

NVIDIA Dynamic Power Software

0.10

NVIDIA Corporation

Oct 20, 2026

Contents

1 MaxLPS Power Pilot Runbooks **3**
1.1 Start Here . . . 3
1.2 Run the Pilot . . . 3
1.3 Supporting Guides . . . 3
1.3.1 MaxLPS Power Pilot . . . 4
1.3.1.1 The Five Pilot Steps . . . 4
1.3.1.2 How the Pilot Fits Together . . . 4
1.3.1.3 Continue With . . . 4
1.3.2 Run the MaxLPS Power Pilot . . . 5
1.3.2.1 Start Here . . . 5
1.3.2.2 Detailed Procedure . . . 5
1.3.2.3 Step 1: Complete Prerequisites and Deploy DPS . . . 6
1.3.2.4 Prerequisites . . . 7
1.3.2.5 Step 2: Set Up Metrics Collection . . . 11
1.3.2.6 Step 3: Create a Topology and Run a BMC Health Check . . . 12
1.3.2.7 Step 4: Prepare the Workload and Comparison Controls . . . 20
1.3.2.8 Step 5: Run the Baseline . . . 22
1.3.2.9 Step 6: Enable MaxLPS . . . 22
1.3.2.10 Step 7: Run the MaxLPS Test . . . 25
1.3.2.11 Step 8: Compare Results . . . 27
1.3.2.12 Cleanup . . . 32
1.3.2.13 Next Steps . . . 33
1.3.3 Collect Pilot Telemetry: 1-Second GPU and CPU Data with DCGM Exporter . . . 35
1.3.3.1 What This Captures . . . 36
1.3.3.2 Before You Begin . . . 36
1.3.3.3 Install and Start DCGM Exporter . . . 37
1.3.3.4 Verify the Requested Metrics . . . 39
1.3.3.5 Capture One Sample per Second to a File . . . 40
1.3.3.6 Create the Shareable Tarball . . . 42
1.3.3.7 Download the Dashboard Skill . . . 42
1.3.3.8 Optional Cleanup . . . 43
1.3.3.9 References . . . 43
1.3.4 BMC Readiness and Health Guide . . . 43
1.3.4.1 Confirm BMC Compatibility . . . 44
1.3.4.2 Connect DPS to BMCs . . . 44
1.3.4.3 Activate the Topology Before Checking BMCs . . . 45
1.3.4.4 Run the BMC Health Check . . . 45
1.3.4.5 Review the Health Report . . . 46
1.3.4.6 Account for Slow BMCs . . . 46
1.3.4.7 BMC Health FAQ . . . 48
1.3.4.8 Collect Evidence Before Escalation . . . 50
1.3.5 Troubleshoot the MaxLPS power pilot . . . 50
1.3.5.1 Why Does DPS Deployment Verification Fail? . . . 51

1.3.5.2 Why Does Topology Validation or Import Fail? . . . 51
1.3.5.3 BMC Health Check . . . 52
1.3.5.4 BMC Health . . . 54
1.3.5.5 What If Topology Activation Fails? . . . 54
1.3.5.6 What If Resource Group Creation Fails? . . . 55
1.3.5.7 What If I Cannot Add Nodes to the Resource Group? . . . 55
1.3.5.8 What If Resource Group Activation Fails? . . . 55
1.3.5.9 How Do I Check Power Steering? . . . 56

2 Deploying DPS 57
2.1 Before you start . . . 57
2.2 Key concepts . . . 57
2.3 Architecture . . . 58
2.4 Deployment steps . . . 58
2.5 After deployment . . . 59
2.5.1 Installation . . . 59
2.5.1.1 In this Section . . . 59
2.5.2 Concepts . . . 104
2.5.2.1 Overview . . . 104
2.5.2.2 Key Abstractions . . . 104
2.5.2.3 Next Steps . . . 105
2.5.3 Operating DPS . . . 207
2.5.3.1 Change what DPS manages . . . 207
2.5.3.2 Watch what it is doing . . . 207
2.5.3.3 Diagnose a problem . . . 207
2.5.3.4 Maintain the deployment . . . 207
2.5.3.5 Connect other systems . . . 208
2.5.4 Integrating DPS . . . 293
2.5.4.1 Integrate with a workload scheduler . . . 293
2.5.4.2 Integrate with an inventory or cluster manager . . . 293
2.5.4.3 Integrate with the electrical grid . . . 294
2.5.4.4 Integrate with BMCs . . . 294
2.5.4.5 Build against the API . . . 294
2.5.4.6 Related . . . 294
2.5.5 Troubleshooting . . . 294
2.5.5.1 Safety and Evidence . . . 294
2.5.5.2 Quick Diagnosis . . . 295
2.5.5.3 Errors and Diagnosis . . . 296
2.5.5.4 Error Code Index . . . 300
2.5.5.5 Escalation . . . 301
2.5.5.6 Related Documentation . . . 302
2.5.6 Reference . . . 302
2.5.6.1 APIs . . . 303
2.5.6.2 Topology Schema Reference . . . 427
2.5.6.3 dpsctl . . . 439

3 DPS SDK and Simulator 521
3.1 Evaluate DPS Without Hardware . . . 521
3.2 Build integrations . . . 521
3.3 Deploying against real hardware . . . 522
3.3.1 SDK Simulator User Guide . . . 522
3.3.1.1 Deployment Profiles . . . 522
3.3.1.2 What's Included . . . 522
3.3.1.3 Use Cases . . . 523

3.3.1.4 Quick Start . . . 523
3.3.1.5 Access Services . . . 524
3.3.1.6 Next Steps . . . 524
3.3.2 Administrator Guides . . . 559
3.3.2.1 Operations . . . 559
3.3.2.2 Partner Integration . . . 590
3.3.3 Developer Guides . . . 662
3.3.3.1 Developer Guide . . . 662

4 DSX OS **673**
4.1 DSX Agent Deployment Guide . . . 673
4.1.1 Connection Overview . . . 673
4.1.2 Prepare the Deployment . . . 674
4.1.3 Select the Chart and Image . . . 674
4.1.4 Create the Secrets . . . 674
4.1.5 Configure the Agent . . . 675
4.1.6 Configure Webhook Delivery . . . 677
4.1.7 Install and Check Startup . . . 677
4.1.8 Verify the Signed Message Flow . . . 678
4.1.9 Troubleshooting . . . 679
4.1.10 Update Credentials and Configuration . . . 679
4.2 Integration Architecture . . . 680
4.3 Integration Responsibilities . . . 680
4.4 Load Targets and Status . . . 680
4.5 Deployment Boundaries and Verification . . . 681

5 Release Notes **683**
5.1 DPS 0.10 Release Notes . . . 683
5.1.1 Upgrade Information . . . 683
5.1.1.1 Review Optional External Authentication . . . 683
5.1.2 Added . . . 683
5.1.3 Improved . . . 684
5.1.4 Fixed . . . 684
5.1.5 Important DSX Configuration Information . . . 684
5.1.5.1 Using DSX Agent Connection Settings . . . 684
5.1.5.2 DSX MQTT Topics . . . 685
5.1.5.3 Nautobot Import Configuration . . . 686
5.2 DPS 0.9.0 Release Notes . . . 686
5.2.1 Important Upgrade Information for MaxLPS Power Pilots . . . 686
5.2.1.1 Upgrading a MaxLPS Pilot Deployment From DPS 0.8.x . . . 686
5.2.1.2 Review the Default Policy Bundle . . . 686
5.2.2 Added . . . 687
5.2.2.1 Policy-Driven Power Management . . . 687
5.2.2.2 MaxLPS Dynamic Power Management . . . 687
5.2.2.3 Hardware, BMC, and Agent Operations . . . 688
5.2.2.4 Authentication, Security, and Observability . . . 688
5.2.2.5 UI and SDKs . . . 688
5.2.3 Improved . . . 688
5.2.4 Fixed . . . 688
5.2.5 Deprecated . . . 689
5.2.6 Known Issues . . . 689
5.2.7 Further Upgrade Information . . . 689
5.2.7.1 Update Telemetry Settings . . . 689
5.2.7.2 Review Power Steering Defaults . . . 689

5.2.7.3 Configure Production Authentication and TLS . . . 689
5.2.7.4 Upgrade Clients and Build Environments . . . 690
5.3 0.8.3 Release Notes . . . 690
5.3.1 Added . . . 690
5.3.2 Improved . . . 690
5.3.3 Fixed . . . 690
5.3.4 Security . . . 691
5.3.5 Deprecated . . . 691
5.3.6 Removed . . . 691
5.3.7 Compatibility and Support Notes . . . 691

6 NVIDIA SOFTWARE LICENSE AGREEMENT 693
6.1 1. License Grants . . . 693
6.1.1 1.1 License Grant to You . . . 693
6.1.2 1.2 License Grant to NVIDIA . . . 693
6.2 2. License Restrictions . . . 694
6.3 3. Authorized Users . . . 694
6.4 4. Pre-Release . . . 694
6.5 5. Third Party Privacy Practices . . . 695
6.6 6. Updates . . . 695
6.7 7. Components Under Other Licenses . . . 695
6.8 8. Ownership . . . 695
6.8.1 8.1 NVIDIA Ownership . . . 695
6.8.2 8.2 Your Ownership . . . 695
6.9 9. Feedback . . . 696
6.10 10. No Publicity . . . 696
6.11 11. Termination . . . 696
6.11.1 11.1 Termination for Convenience . . . 696
6.11.2 11.2 Termination for Cause . . . 696
6.11.3 11.3 Effect of Termination . . . 696
6.11.4 11.4 Survival . . . 697
6.12 12. Disclaimer of Warranties . . . 697
6.13 13. Limitations of Liability . . . 697
6.13.1 13.1 EXCLUSIONS . . . 697
6.13.2 13.2 DAMAGES CAP . . . 697
6.14 14. Use in Mission Critical Applications . . . 697
6.15 15. Governing Law and Jurisdiction . . . 698
6.16 16. General . . . 698
6.16.1 16.1 Indemnity . . . 698
6.16.2 16.2 Independent Contractors . . . 698
6.16.3 16.3 No Assignment . . . 698
6.16.4 16.4 No Waiver . . . 698
6.16.5 16.5 Trade Compliance . . . 699
6.16.6 16.6 Government Rights . . . 699
6.16.7 16.7 Notices . . . 699
6.16.8 16.8 Severability . . . 699
6.16.9 16.9 Construction . . . 700
6.16.10 16.10 Amendment . . . 700
6.16.11 16.11 Entire Agreement . . . 700

7 Look something up 701

Important

Developer Preview Notice

NVIDIA Dynamic Power Software is currently offered as a Developer Preview. Access to DPS is governed by the applicable Developer Preview guidelines.

The software may have limitations, may change significantly, and includes functionality that continues to be refined through testing and feedback. It should not be used in production environments except as part of an authorized collaboration.

Dynamic Power Software (DPS) is a data center power management system that monitors power consumption, enforces power policies, and integrates with BMCs and cluster management systems. It addresses the challenge of optimizing power consumption while maintaining performance and reliability at scale.

Chapter 1. MaxLPS Power Pilot Runbooks

Run this pilot to determine whether four GB200 or GB300 NVL72 racks can deliver more aggregate inference throughput inside the same facility power envelope as a three-rack baseline. DPS manages fleet GPU power while your existing workload scheduler and provisioning systems keep ownership of workload placement and fleet membership.

1.1. Start Here

Pilot at a glance Get the pilot goal, the five steps, the minimum concepts, and links to the right detailed guide. Start here if you are new to DPS or returning to the pilot after working in another guide.

MaxLPS Power Pilot

1.2. Run the Pilot

Run the MaxLPS power pilot Complete the end-to-end procedure: prepare DPS, establish the 3-rack baseline, enable the default `MaxLPS` bundle for the 4-rack managed run, and compare results.

Run the MaxLPS Power Pilot Rehearse the pilot with the SDK Use the SDK Pilot profile to practice topology and resource-group operations, replay simulated workloads, and compare GPU power without physical hardware. Inference performance and physical power compliance require the hardware pilot.

Rehearse the MaxLPS Power Pilot With the SDK Simulator

1.3. Supporting Guides

Collect pilot telemetry Capture the GPU and CPU power and utilization evidence needed to compare the baseline and managed runs.

Collect Pilot Telemetry: 1-Second GPU and CPU Data with DCGM Exporter How MaxLPS works Learn how the topology, resource group, default `MaxLPS` bundle, and Power Steering work together to dynamically allocate GPU power within the facility envelope.

MaxLPS Power Management BMC readiness and health Prepare BMC access, verify Redfish compatibility, run the health gate, and resolve BMC latency or power-control issues before a managed DPS run.

BMC Readiness and Health Guide Troubleshoot the pilot Find targeted recovery steps for deployment verification, topology, BMC health, resource groups, and Power Steering.

Troubleshoot the MaxLPS power pilot

1.3.1. MaxLPS Power Pilot

Compare a three-rack baseline with a four-rack DPS-managed run under the same facility power limit. Run the same workload mix without DPS across three NVL72 racks, then with DPS dynamically managing GPU power across four racks. Compare throughput, latency, and power compliance.

To run this pilot, prepare: four GB200 NVL72 or GB300 NVL72 racks (use three for the baseline and all four for the managed run), BMC access, and the facility power limit to hold fixed for both runs. You also need two representative inference jobs—one lower-power and one higher-power—and metrics collection with a place to retain the results. For details, refer to *pilot prerequisites*.

To practice the workflow before using hardware, follow *Rehearse the MaxLPS Power Pilot With the SDK Simulator*. The SDK Pilot profile compares simulated GPU power across a three-rack baseline and a four-rack managed run. It does not measure inference performance or physical power compliance.

1.3.1.1 The Five Pilot Steps

1. **Get ready.** Deploy DPS, authenticate `dpsctl`, and confirm the site and metrics. Follow the *Deployment and Configuration Guide*, *pilot prerequisites*, and *metrics setup*.
2. **Create the topology and check BMCs.** Import the devices and four-rack topology, activate it, then check every BMC. DPS 0.9 requires an active topology for `dpsctl check connection` and `dpsctl verify bmc-health start`. Deactivate it and verify default GPU limits before the baseline. Follow *Step 3*.
3. **Run the baseline.** Run the workload mix without DPS and capture the comparison metrics. Follow *Step 4* to prepare the run and *Step 5* to run it. *If baseline nodes and metrics are ready, you can run it first. Keep the topology inactive until the baseline finishes, then activate it for BMC checks.*
4. **Enable MaxLPS.** Reactivate the topology, create the resource group with the default `MaxLPS` bundle, and confirm Power Steering is Ready with no reconciliation errors. Follow *Step 6*.
5. **Run the MaxLPS test and compare.** Repeat the workload mix across four racks, confirm GPU limits move under load, then compare throughput, latency, and power compliance. Follow *Step 7* to run the managed test and *Step 8* to compare the results.

1.3.1.2 How the Pilot Fits Together

- A **topology** describes the four racks and the power domain that defines the operating envelope.
- A **resource group** activates its compute nodes for dynamic power management.
- The **`MaxLPS` policy bundle** and Power Steering use live telemetry to manage the resource group's GPU limits. Do not create a pilot-specific policy bundle.

1.3.1.3 Continue With

- *Run the pilot* — follow the complete, step-by-step procedure.
- *Collect telemetry* — capture the metrics needed for the comparison.
- *BMC readiness and health* — prepare BMC access and resolve BMC health issues before the managed run.

- *Troubleshoot* — resolve a failed gate or command.
- *How MaxLPS works* — understand the topology, resource group, bundle, and dynamic power-management model.

1.3.2. Run the MaxLPS Power Pilot

Jump to a section

- *Start Here*
- *Detailed Procedure*
- *Step 1: Complete Prerequisites and Deploy DPS*
- *Prerequisites*
- *Step 2: Set Up Metrics Collection*
- *Step 3: Create a Topology and Run a BMC Health Check*
- *Step 4: Prepare the Workload and Comparison Controls*
- *Step 5: Run the Baseline*
- *Step 6: Enable MaxLPS*
- *Step 7: Run the MaxLPS Test*
- *Step 8: Compare Results*
- *Cleanup*
- *Next Steps*

1.3.2.1 Start Here

Use this procedure after reading *MaxLPS Power Pilot*. Run the same mix of at least two representative inference jobs—one low-power-utilization and one high-power-utilization—on GB200 NVL72 or GB300 NVL72 racks first without, then with, DPS fleet power management. Keep fleet-wide demand and client behavior the same for both runs.

Pilot goal: determine whether four managed racks deliver more aggregate inference throughput inside the same facility power envelope as a three-rack baseline, while maintaining power compliance.

For the MaxLPS operating model and the scheduler-independent configuration used in this runbook, refer to *MaxLPS Power Management*.

1.3.2.2 Detailed Procedure

1. **Complete prerequisites and deploy DPS** — Confirm the pilot site, deploy DPS, install and authenticate `dpsctl`, then verify the deployment (*Step 1*).
2. **Set up metrics collection** — Configure and validate workload, power, GPU-limit, and BMC telemetry before creating the topology or running the baseline. Retain raw data for both comparisons (*Metrics setup*).

3. **Create a topology and run a BMC health check** — Import devices if needed, create and activate the 4-rack topology with a single **PowerDomain** budget, check every BMC, then deactivate the topology for the baseline (*Step 3*).
4. **Prepare the workload and comparison controls** — Stage the low- and high-power inference jobs, fix fleet-wide QPS or concurrency and client behavior, then define the steady-state window, KPIs, and evidence package (*Step 4*).
5. **Run the baseline** — Execute the workload mix **without** DPS in the control path and retain the baseline evidence (*Step 5*).
6. **Enable MaxLPS** — Reactivate the topology, select the default `MaxLPS` policy bundle, then create and activate the resource group with dynamic power management, Power Steering, and **Shared GPU** (*Step 6*).
7. **Run the MaxLPS test** — Repeat the **same** workload mix and fleet-wide QPS or concurrency across **4 racks** under the **same** envelope with DPS enforcing the topology budget (*Step 7*).
8. **Compare results** — Align metrics windows, quantify throughput uplift, and check for envelope breaches or throttling (*Step 8*).

1.3.2.3 Step 1: Complete Prerequisites and Deploy DPS

Back to the five-step pilot overview.

1.3.2.3.1 Confirm Prerequisites

1.3.2.3.2 Scheduler-independent configuration

Here **scheduling and provisioning** means **workload schedulers** (Slurm, Kubernetes, and similar job or pod placement layers) **and provisioning systems** (cluster installers, fleet managers, automation that registers, scales, or re-images nodes)—everything that normally owns *where* nodes run and *which* hardware is in service—not DPS's power plane.

DPS is designed to follow those layers when you want it to: workload schedulers **and** provisioning systems can drive resource group lifecycle and policy through the ***`nvidia.dcpower.v1` resource group API*** and the matching ***`dpsctl resource-group`*** commands so membership stays aligned with jobs, reservations, or capacity changes. *Managing Resource Groups* documents that integrated style of operation step by step.

You can still pursue the MaxLPS pattern **without** that coupling by defining **one or more resource groups** whose members you manage **outside** both workload schedulers **and** provisioning systems—fixed topology entities, `dpsctl resource-group create` / `add` / `activate` only—while inference continues to run through your existing scheduling and provisioning unchanged. **Scheduler-independent configuration**, in this runbook, is that decoupled path: the smallest setup that enforces a **shared GPU power budget** across the fleet while **scheduling and provisioning** never call DPS to add or remove nodes for you.

That configuration consists of:

- A simplified **topology** that models the 4 racks and the operating power domain above them.
- A single **resource group** containing every compute node in the topology, created with three capabilities enabled together:
- **DPM** (Dynamic Power Management) — DPS actively enforces per-node power policy.
- **PRS** (Power Reservation Steering) — DPS's mechanism that redistributes headroom across GPUs based on live telemetry. Refer to *Power Reservation Steering*.

- **Shared GPU** — the resource group's GPU budget is enforced as a group sum rather than per-node, which is what makes operating more racks within one envelope possible. Refer to *Shared GPU Budget*.

If any one of these three is missing, the pilot configuration described here is incomplete and the comparative results in this runbook will not be reproducible. Creating these conditions is covered in the following sections.

1.3.2.3.3 What you will do

1. **Validate the deployment** — Authenticate, run `dpsctl verify`, and interpret failures so you are not debugging topology on a broken server (*Part 1*).
2. **Model the site and gate end-to-end BMC health** — Import devices if needed, build the 4-rack topology with a single **PowerDomain** budget, run the deep BMC health check against every node, and activate the topology once as a temporary DPS-to-BMC end-to-end gate (*Part 2*).
3. **Deactivate before the baseline** — After the BMC health and activation gates in Part 2 pass, deactivate the topology so the unmanaged baseline runs with DPS out of the control path (*Part 3*).
4. **Prepare the baseline run** — Stage inference consistently, define the steady-state window and KPIs, and document the **3-rack** footprint and power context (*Part 4*).
5. **Run the unmanaged baseline** — Execute the workload **without** DPS in the control path and capture baseline metrics (*Part 5*).
6. **Enable fleet power policy** — Re-activate the topology, then create and activate the resource group with the shipped **MaxLPS** bundle and **DPM**, **PRS**, and **Shared GPU** (*Part 6*).
7. **Run the managed comparison** — Repeat the **same** inference profile across **4 racks** under the **same** envelope with DPS enforcing the topology budget (*Part 7*).
8. **Compare and decide** — Align metrics windows, quantify throughput uplift, and watch for envelope breaches or throttling (*Part 8*).

1.3.2.4 Prerequisites

Before starting this runbook, confirm each of the following:

- **Hardware**: Four GB200 NVL72 or GB300 NVL72 racks identified for the pilot. All 18 compute nodes per rack must be powered on and BMC-reachable.
- **BMC preparation for managed DPS**: Before you import the topology, complete the Redfish compatibility, firmware, client-capacity, access, and credential prerequisites in *BMC Readiness and Health Guide*. Step 3 runs the BMC health gate after import and before the managed run.
- **Inference workload and dataset staged** on every node in all 4 racks, launchable from your job scheduler or equivalent orchestration.
- **PowerDomain budget** for the pilot is known and documented — this runbook uses the same 3-rack operating envelope as the baseline.

1.3.2.4.1 Deploy DPS and Install `dpsctl`

Deploy the DPS 0.9.x server to a Kubernetes cluster with the *Deployment and Configuration Guide*, then install a matching 0.9 `dpsctl` client from DPS on NGC. For this MaxLPS pilot, deploy a DPS Agent with mTLS before you import the topology. The agent must be ready and able to reach every assigned BMC or Redfish proxy. Set up the BMC access and credentials as described in *BMC Readiness and Health Guide*.

1.3.2.4.1.1 Deploy the Pilot Agent

Issue an agent certificate whose common name is `maxlps-pilot` from a CA that the DPS server trusts. Create the Secret in the DPS namespace with the CA, certificate, and private-key files. Do not put private keys in the Helm values file or topology JSON.

```
kubectl create secret generic dps-agent-mtls \
  --namespace dps \
  --from-file=ca.crt=<path-to-agent-ca-certificate> \
  --from-file=tls.crt=<path-to-maxlps-pilot-certificate> \
  --from-file=tls.key=<path-to-maxlps-pilot-private-key>
```

Configure the DPS server with TLS or mTLS before enabling the agent, and ensure that DPS trusts the agent certificate chain. Then add the following settings to the Helm values file used for the DPS deployment. This pilot uses `maxlps-pilot` as its agent ID. The certificate common name is allowed separately in `dps.grpcServiceMTLSCN`.

```
dps:
  agent:
    allowHandshakeFallback: false
  grpcServiceMTLSCN:
    nvidia.dcpower.v1.DPSAgentService:
      - "maxlps-pilot"

agent:
  enabled: true
  replicas: 1
  agentID: "maxlps-pilot"
  tls:
    transportSecurity: "mtls"
    existingSecret: "dps-agent-mtls"
```

Set DPS_RELEASE to the selected 0.9 chart version. The following example uses the `dps` release name and namespace.

Render, install, and verify the Agent deployment

```
helm template dps ngc/dps \
  --namespace dps \
  --version "${DPS_RELEASE}" \
  --values values.yaml > dps-rendered.yaml

helm upgrade --install dps ngc/dps \
  --namespace dps \
  --create-namespace \
  --version "${DPS_RELEASE}" \
  --values values.yaml \
  --wait \
  --timeout 15m

kubectl rollout status deployment/dps-server-agent \
  --namespace dps \
  --timeout 5m
kubectl get pods --namespace dps --selector app=dps-server-agent
```

Do not continue until the Agent Pod is Ready. If it does not become Ready, inspect the certificate chain, certificate common name, DPS server trust configuration, and Agent-to-BMC or proxy network path.

Install and authenticate `dpsctl` as described in *Installing dpsctl*. The client must be able to reach the DPS server.

To practice this workflow without physical hardware, follow *Rehearse the MaxLPS Power Pilot With the SDK Simulator*. The SDK Pilot profile provides four simulated GB300 racks, workload-trace replay, and a GPU power comparison in Grafana. It demonstrates configuration and control behavior; inference performance and physical power compliance require this hardware procedure.

1.3.2.4.2 Verify DPS Deployment

Before you configure a topology, confirm the deployment is healthy end-to-end. Treat this as a gate. If any component is unhealthy, stop and fix it before moving to *Step 3*.

> **Note:** If you have not already, export a few `dpsctl` environment variables to keep the commands that follow concise:

```
export DPSCTL_HOST="api.dps.your-domain.com"
export DPSCTL_PORT="443"
```

For a production pilot, use a certificate that the client trusts. When the endpoint uses a private CA, set its certificate path:

```
export DPSCTL_CA_CERT_PATH="/path/to/ca-certificate.pem"
```

Do not set `DPSCTL_INSECURE_TLS_SKIP_VERIFY` for a production endpoint.

Use `DPSCTL_INSECURE=true` only for local bring-up against a deliberately plaintext endpoint. `DPSCTL_INSECURE_TLS_SKIP_VERIFY` permits an encrypted connection without certificate verification; it does not configure a plaintext connection.

1.3.2.4.2.1 Authenticate the CLI

Configure the CLI for your server's setup by following *Configure Server Authentication*. For LDAP or evaluation authentication, run the following command. For OIDC or certificate access, use the corresponding procedure in that guide.

```
dpsctl login -u <username>
```

`dpsctl login` prompts for the password and saves tokens in `~/.dpsctl/credentials.yaml`. Refer to *Installing dpsctl* for installation and *dpsctl login* for login options.

1.3.2.4.2.2 Confirm Server and Client Versions

Confirm authentication succeeded and capture the server version:

```
dpsctl server-version
```

A successful call returns the server's version payload. Confirm the server version matches the latest DPS release available on NGC.

Confirm the local `dpsctl` client matches the server version:

```
dpsctl --version
```

Confirm this reports the same version as the server. If `dpsctl server-version` returns `AUTHENTICATION_FAILED` or `missing token`, follow *Cannot Connect or Log In* for your selected authentication setup before continuing.

1.3.2.4.2.3 Run the Verify Gate

Proceed to *Step 3* only when `status.ok` is `true` and no component reports `healthy: false`. A `skipped: true` component is acceptable — it means that component is not part of this deployment.

If BCM is intentionally not part of the deployment, its result does not block this pilot. Confirm the other components are healthy and continue. Refer to *Deployment Component Unhealthy* for the no-BCM output and to *Troubleshoot the MaxLPS power pilot* for the full component list.

Run `dpsctl verify` with no component flags. It checks the DPS server, database, authentication service, UI ingress, and BCM connectivity in a single call:

```
dpsctl verify
```

Each component (`dps_server`, `database`, `auth`, `ui`, `bcm`) returns `healthy`, `message`, and `skipped`. The `details` field is informational and is mainly useful for troubleshooting.

Example healthy output


```
{
  "auth": {
    "details": "Successfully connected to LDAP server: ldaps://openldap.dps.
↪svc.cluster.local:636. Service account authentication successful",
    "healthy": true,
    "message": "LDAP connectivity and authentication verified",
    "name": "Authentication Service"
  },
  "bcm": {
    "details": "Successfully connected to BCM server",
    "healthy": true,
    "message": "BCM connectivity successful",
    "name": "BCM"
  },
  "database": {
    "details": "Database ping successful",
    "healthy": true,
    "message": "Database connectivity successful",
    "name": "Database"
  },
  "dps_server": {
    "details": "gRPC endpoint accessible, topology server responsive",
    "healthy": true,
    "message": "DPS Server is healthy",
    "name": "DPS Server"
  },
  "status": {
    "diag_msg": "All requested DPS components are healthy",
```


(continues on next page)

(continued from previous page)

```
    "ok": true
  },
  "ui": {
    "details": "URL: https://ui.dps:80. Note: You may need to update your /
↪etc/hosts file to resolve ui.dps to the cluster's IP address. The UI should
↪be tested from an external browser.",
    "healthy": true,
    "message": "UI ingress endpoint retrieved successfully",
    "name": "DPS UI"
  }
}
```

Troubleshooting: If deployment verification fails, see *Why Does DPS Deployment Verification Fail?*.

1.3.2.5 Step 2: Set Up Metrics Collection

Configure and validate metrics before creating the topology or starting either comparison run. This avoids discovering after the baseline that a required signal is missing or clocks cannot be aligned.

Collect the following signals with timestamps from the same time source:

- workload demand, throughput, token rate, latency percentiles, and errors;
- per-GPU power, utilization, and observed power limits; use *Collect Pilot Telemetry: 1-Second GPU and CPU Data with DCGM Exporter* to set up and validate the DCGM capture;
- BMC health and latency, plus rack or facility power for the same boundary as the pilot envelope; and
- the DPS Prometheus metrics used to verify GPU-cap movement during the managed run. Configure the scrape before the managed run as described in *Enable DPS Prometheus Metrics*.

Run a short test capture before creating the topology. Confirm every required signal has a timestamp, expected units, and a known retention location. Record the collector versions, sampling intervals, clock source, and any offsets in the evidence package.

1.3.2.5.1 Retain and Download a Power Steering Dump

You can retain a Power Steering (PRS) telemetry dump to help investigate the managed pilot run. If you need more time to collect it, extend retention before starting the run. Download the dump after the run, before its retention expires, and save it with the *evidence package*.

1.3.2.5.1.1 Extend Dump Retention (Optional)

This setting controls dump retention in the bundled SeaweedFS storage. If your deployment uses external object storage, configure retention through that storage system's lifecycle policy.

Run the following commands on the deployment host with Helm access to the DPS deployment and version 4 of Mike Farah's yq. Use the existing release name, namespace, chart, and values file. Keep `DPS_RELEASE` set to the installed chart version from Step 1. `DPS_RELEASE_NAME` is the Helm release name, not the chart version.

Set the retention period. The following example uses `24h`:

```
NS='<namespace>'
DPS_RELEASE_NAME='<dps-release-name>'
DPS_CHART='<DPS-chart-path-or-OCI-reference>'
DPS_VALUES='/path/to/dps/values.yaml'
TTL='24h'
```

Update the retention setting in your DPS values file:

```
yq -i \
  '.power-steering.global.telemetry.dump.retention = "'"$TTL"'"' \
  "$DPS_VALUES"
```

Apply the values using the same chart version as the installed deployment:

```
helm upgrade "$DPS_RELEASE_NAME" "$DPS_CHART" \
  --namespace "$NS" \
  --version "$DPS_RELEASE" \
  --reuse-values \
  -f "$DPS_VALUES"
```

1.3.2.5.1.2 Download the Dump

After the managed run, run the following commands on the deployment host. Set `CONTROLLER_URL` to the Power Steering controller URL reachable from that host. Download the dump to your home directory:

```
CONTROLLER_URL='https://<power-controller-host>'

curl -fSL \
  -o "$HOME/power-steering-dump.tar" \
  "${CONTROLLER_URL%/}/api/dump?frames=*"
```

Verify that the downloaded file is a valid tar archive:

```
tar -tf "$HOME/power-steering-dump.tar" >/dev/null \
  && echo "Dump ready: $HOME/power-steering-dump.tar"
```

Save the archive with the managed run's evidence package before downloading another dump to the same path.

1.3.2.6 Step 3: Create a Topology and Run a BMC Health Check

Back to the five-step pilot overview.

To manage power effectively, DPS needs to know the structure of the data center in which it is operating. We call this the data center topology. DPS also needs to know of any custom devices it will encounter in that topology.

1.3.2.6.1 Import Device Specifications (Optional)

DPS ships with a set of supported device specifications out of the box. The supported devices include, at time of writing:

- **Compute**: `DGX_GB200`, `DGX_GB300`, `DGX_B200`, `DGX_B300`, `DGX_H100`

- **GPUs**: `GB200`, `GB300`, `B200`, `B300`, `H100`
- **CPUs**: `Grace`, `Grace_GB300`
- **Power infrastructure**: `FloorPDU95` (Generic Floor PDU with 95% efficiency factor) , `RackPDU95_57500W` (Generic Rack PDU with 95% efficiency factor and max load of 57,000 W), `PowerSupply95_3300W` (Generic Power Supply with 95% efficiency factor and max load of 3,300 W)

GB300 GPU minimum power (optional). `minLoadWatts` in the **`GB300`** GPU entry of `devices.yaml` is the lower power bound that Dynamic Power Management can request. Do not derive it from a workload average. Change it only when the BMC-supported minimum differs from the shipped model, and confirm that minimum on a representative system before `dpsctl device upsert`. A value below the supported minimum can make DPS request a limit that the BMC rejects. A value above the supported minimum can prevent expected power reductions. Keep `maxLoadWatts` above `minLoadWatts`, record any approved override, and re-import it per *Managing Devices*.

If your data center uses PDUs, PSUs, or other equipment whose model does **not** match the list above, you must register a specification for it before it can appear in a topology. Follow the full procedure in *Managing Devices*, then import your additions:

```
dpsctl device upsert devices.yaml
```

1.3.2.6.2 Verify Device Specifications

Confirm that every model you reference in your topology appears in the output:

```
dpsctl device list
```

1.3.2.6.3 Define the Topology and Static Load

The topology is how you describe your power-distribution network to DPS. For this runbook, the topology must contain at minimum:

Entity type	Count	Purpose
`PowerDomain`	1	Top-level 3-rack operating envelope applied across all 4 racks.
`PowerDistribution`	4 (one per rack)	Rack PDU.
`ComputerSystem`	72 (18 per rack)	GB200 or GB300 compute nodes, parented under their rack PDU.

The `PowerDomain`'s `PowerValue` is the top-level operating envelope DPS enforces across the 4-rack pilot topology. Set it to the same 3-rack envelope used by the baseline: **375 kW** for GB200 NVL72 or **405 kW** for GB300 NVL72.

Each GB300 NVL72 rack contains 9 NVSwitch trays. Their draw is **not** managed by DPS, but it must still be reflected in the rack PDU and `PowerDomain` budget.

The runbook represents the per-rack NVSwitch aggregate with the **`StaticLoad`** field on each `FloorPDU-Rack0X` (`PowerDistribution`) entity. `StaticLoad` declares fixed power consumption from unmanaged devices and is propagated up the power-distribution network automatically—see *Entities Static Load*.

Use **7,416 W per rack** for a standard GB300 NVL72 9-switch aggregate (replace with measured or vendor-specified values when available). The example topology in this step applies this value on every rack PDU. For a standard GB200 NVL72 rack, we recommend a 9-switch aggregate power value of **11,160 W per rack**.

1.3.2.6.4 Prepare the Topology File

Download the reference topology that matches your hardware:

- **GB300 topology**: `maxlps-pilot-gb300.json` — 4 GB300 NVL72 racks, 72 `DGX_GB300` nodes, 4 `FloorPDU95` rack PDUs at 135 kW (`StaticLoad` 7,416 W), and a `PDU-Root` `PowerDomain` at the 3-rack envelope of 405 kW.
- **GB200 topology**: `maxlps-pilot-gb200.json` — 4 GB200 NVL72 racks, 72 `DGX_GB200` nodes, 4 `FloorPDU95` rack PDUs at 125 kW (`StaticLoad` 11,160 W), and a `PDU-Root` `PowerDomain` at the 3-rack envelope of 375 kW.

These DPS 0.9 topology files intentionally contain no `Policies` section. DPS 0.9 rejects legacy topology policies; bind the default `MaxLPS` policy bundle when you create the resource group in *Step 6*.

Replace the BMC URL and `SecretName` values with those for your fleet before importing. The downloadable GB200 and GB300 examples assign the pilot agent at `PDU-Root`, so all child devices inherit `AgentId: maxlps-pilot`. Replace this value only if it matches the deployed agent's `agent.agentID`. Configure the certificate common name separately in `dps.grpcServiceMTLSCN`. Refer to *BMC Readiness and Health Guide* for BMC credential mapping and proxy guidance.

Set `TOPOLOGY_FILE` to the file you downloaded:

```
TOPOLOGY_FILE=maxlps-pilot-gb300.json # Use maxlps-pilot-gb200.json for GB200.
```

Preview: topology root, one rack PDU, and the PowerDomain

```
{
  "Topology": {
    "Name": "maxlps-pilot",
    "Entities": [
      {
        "Name": "FloorPDU-Rack01",
        "Children": [
          "gb300-nvl-001-compute01",
          "...",
          "gb300-nvl-001-compute18"
        ]
      },
      {
        "Name": "PDU-Root",
        "Children": [
          "FloorPDU-Rack01",
          "FloorPDU-Rack02",
          "FloorPDU-Rack03",
          "FloorPDU-Rack04"
        ]
      }
    ]
  },
```

(continues on next page)

(continued from previous page)

```
  "Entities": [
    {
      "OperatingLimit": { "PowerValue": { "Type": "W", "Value": 135000 } },
      "Model": "FloorPDU95",
      "Name": "FloorPDU-Rack01",
      "Redfish": { "URL": "https://localhost" },
      "StaticLoad": { "Type": "W", "Value": 7416 },
      "Type": "PowerDistribution"
    },
    {
      "OperatingLimit": { "PowerValue": { "Type": "W", "Value": 405000 } },
      "AgentId": "maxlps-pilot",
      "Name": "PDU-Root",
      "Redfish": { "URL": "https://localhost" },
      "Type": "PowerDomain"
    },
    {
      "Model": "DGX_GB300",
      "Name": "gb300-nvl-001-compute01",
      "Redfish": {
        "SecretName": "gb300-nvl-001-compute01",
        "URL": "https://10.0.0.1"
      },
      "Type": "ComputerSystem"
    }
  ]
}
```

For additional topology examples, refer to *Import a Topology* and *Managing Topologies*.

1.3.2.6.5 Validate and Import the Topology

The topology JSON cannot carry a typed bundle selection and must not contain a top-level `Policies` array. During import, the DPS 0.9 server resolves the current `global_policy_bundle` and persists that selection on the new topology. The later resource-group step binds the shipped `MaxLPS` bundle with `--policy-bundle`; it does not read the topology's stored bundle. Refer to *Import a Topology*.

If you want the imported topology's `policy_bundle` to be `MaxLPS`, confirm `settings.global_policy_bundle` with `dpsctl settings list` before import. Do not change the global default unless it is empty or points at a different bundle. If you do change it for import, restore the recorded value immediately after the import command returns, even if import failed.

```
dpsctl topology validate "$TOPOLOGY_FILE"
dpsctl topology import "$TOPOLOGY_FILE"
```

These commands only register configuration, topology, and inventory. No policy is applied to a BMC until topology activation.

Troubleshooting: If topology validation or import fails, see *Why Does Topology Validation or Import Fail?*.

1.3.2.6.6 Activate the Topology Before Checking BMCs

Before activating, run the deep BMC prerequisite health check against every BMC in the imported pilot topology. This check is broader than a reachability ping: it validates Redfish endpoint reachability, firmware inventory, power-limit write and read-back behavior, telemetry sampling, GPU power-policy state, and in-band/out-of-band power drift across every node. Passing `--topology` runs the check against every node in that topology in a single call. The topology only needs to be imported — not active — for this check to run. Complete the write-safety preflight in *Run a BMC Health Check* before you use `--force-writes`; write mode changes power limits temporarily and can disable active WPPS profiles.

On large clusters, the probe can take hours. `dpsctl verify bmc-health start` starts the check asynchronously and returns a `task_id`. The server task continues after `dpsctl` exits, so retain the ID.

First, run and retain the full read-only baseline report. Resolve any failures before you continue:

```
dpsctl verify bmc-health start --topology maxlps-pilot --skip-writes --
↪expected-edpp-pct 100 --samples-per-telemetry 2500 --telemetry-interval
↪500ms --per-node-timeout 1h
dpsctl verify bmc-health report <read-only-task-id> --wait --output json >
↪bmc-health-maxlps-pilot-read-only.json
```

Next, validate write behavior on one representative node with one in-flight node probe. Confirm that the report passes before you continue:

```
dpsctl verify bmc-health start --topology maxlps-pilot --nodes <node-name> --
↪force-writes --concurrency 1 --expected-edpp-pct 100 --samples-per-
↪telemetry 2500 --telemetry-interval 500ms --per-node-timeout 1h
dpsctl verify bmc-health report <node-write-task-id> --wait --output json >
↪bmc-health-write-probe.json
```

After both checks pass, start the fleet-wide write-enabled check and query its status:

```
dpsctl verify bmc-health start --topology maxlps-pilot --force-writes --
↪expected-edpp-pct 100 --samples-per-telemetry 2500 --telemetry-interval
↪500ms --per-node-timeout 1h
# Query the status until the health check is done
dpsctl verify bmc-health status <fleet-write-task-id>
# Once done, generate a report
dpsctl verify bmc-health report <fleet-write-task-id> --wait --output json >
↪bmc-health-maxlps-pilot.json
dpsctl verify bmc-health report <fleet-write-task-id> --summary-only
```

Example Full GB300 Report: All Nodes Healthy

```
{
  "status": {
    "ok": true,
    "diag_msg": "all nodes healthy"
  },
  "node_reports": [
    {
      "node": "node-001",
```

(continues on next page)

(continued from previous page)


```
      "reachable": true,
      "firmware": [
        { "id": "FW_BMC_0", "version": "25.2.0", "node": "node-001" }
      ],
      "endpoint_stats": [
        {
          "endpoint": "device.StdMetrics",
          "method": "POST",
          "attempts": 2500,
          "successes": 2500,
          "duration": { "count": 1024, "min_ms": 420, "max_ms": 640, "avg_ms
↪": 500, "p50_ms": 500, "p95_ms": 590, "p99_ms": 625 }
        },
        {
          "endpoint": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}
↪/EnvironmentMetrics",
          "method": "GET",
          "attempts": 64,
          "successes": 64,
          "duration": { "count": 64, "min_ms": 620, "max_ms": 980, "avg_ms":
↪800, "p50_ms": 790, "p95_ms": 940, "p99_ms": 980 }
        }
      ],
      "power_settings": [
        { "component": "GPU_0", "set_point_w": 700, "allowable_max_w": 700,
↪"default_set_point_w": 700, "power_w": 104.3 }
      ],
      "power_writes": [
        { "component": "GPU_0", "original_w": 700, "decrement_cycles": 3,
↪"restored": true, "restored_value_w": 700, "increment_cycles": 3 }
      ],
      "edpp": [
        { "processor": "GPU_0", "current_pct": 100, "reference_pct": 100,
↪"reference_available": true }
      ],
      "wpps": [
        { "processor": "GPU_0", "accessible": true, "enforced_mask": "0x0",
↪"requested_mask": "0x0", "supported_mask": "0xff" }
      ],
      "telemetry_samples": 2500,
      "telemetry": [
        { "component": "GPU_0", "power_w": { "count": 2500, "min": 92.4, "max
↪": 119.8, "mean": 104.7, "p50": 104.5, "p95": 116.9, "p99": 119.1 } }
      ]
    },
    { "node": "node-002", "reachable": true, "telemetry_samples": 2500 },
    { "node": "node-003", "reachable": true, "telemetry_samples": 2500 },
    { "node": "node-004", "reachable": true, "telemetry_samples": 2500 },
    { "node": "node-005", "reachable": true, "telemetry_samples": 2500 }
  ],
  "cluster_summary": {
    "total_nodes": 5,
```


(continues on next page)

(continued from previous page)

```
    "passed": 5,
    "endpoint_latency_ms": { "count": 5600, "min_ms": 420, "max_ms": 1100,
↪"avg_ms": 800, "p50_ms": 790, "p95_ms": 980, "p99_ms": 1080 },
    "telemetry_latency_ms": { "count": 5120, "min_ms": 420, "max_ms": 640,
↪"avg_ms": 500, "p50_ms": 500, "p95_ms": 590, "p99_ms": 625 },
    "nodes_attempted": 5
  }
}
```

1.3.2.6.7 Step 7: Review BMC health results

This run is also where you collect BMC latency metrics for the pilot. From `bmc-health-maxlps-pilot.json`, capture the following latency summaries for every node:

- `endpoint_latency_ms`
- `telemetry_latency_ms`
- `power_get_latency_ms`
- `power_set_latency_ms`
- `edpp_get_latency_ms`
- `wpps_get_latency_ms`

For MaxLPS readiness, target p50 below 0.5 seconds, p95 below 0.7 seconds, p99 below 0.8 seconds, and max near 1 second for read and write paths. If p99 is greater than 1 to 2 seconds, document the latency measurements and assess cluster-wide BMC client connection pressure before activation.

Prepare other BMC clients before you run this check. Confirm the BMC client connection requirements from the prerequisites are still in place:

- For GB200 and GB300 systems, the NVIDIA BMC and DPS teams recommend no more than four BMC client connections, including two DPS client connections.
- Use session token authentication instead of basic authentication.
- Use keep-alive to maintain the connection.

In the summary report generated in Step 6, confirm:

- `status.ok` is `true`.
- `cluster_summary.total_nodes` is 72, matching the node count in `maxlps-pilot`.
- `cluster_summary.failed` is `0`.
- `cluster_summary.nodes_unreachable` is `0`.
- `issues` contains no `SEVERITY_ERROR` entries.

Known B200/B300 firmware-validation exception: on systems with B200 or B300 GPUs, DPS 0.8.x can report a `FIRMWARE_VALIDATION_FAILED` issue with `message: firmware validation failed` even when the BMC is usable. If this is the only `SEVERITY_ERROR` for the affected B200 or B300 node, `cluster_summary.nodes_unreachable` is still `0`, and the rest of the health report is clean, document the exception and continue with the runbook.

If any of these do not hold, other than the known B200/B300 firmware-validation exception described here, the `issues` list identifies what to fix. Each entry includes the affected `node`, the `resource` within that node, an issue `code`, the `observed` value, the `threshold` it violated, and a human-readable

`message`. Fix the affected nodes and re-run the health check until the report is clean before activating the topology.

Troubleshooting: For issue-specific remediation, see *BMC Health Check*.

1.3.2.6.8 Step 8: Activate the topology

Activate the topology as a temporary end-to-end gate. Activation exercises the full DPS-to-BMC set-policy path against every node so any device-model or policy-application issue that the BMC health review in Step 7 could not catch surfaces now, at setup time, instead of immediately before the 4-rack managed run in *Part 7*. The topology will be deactivated in *Part 3* so the unmanaged baseline in *Part 5* runs with DPS out of the control path, and re-activated later in this runbook.

```
dpsctl topology activate --topology maxlps-pilot # Must match Topology.Name
↪in the imported file.
```

Troubleshooting: If topology activation fails, see *What If Topology Activation Fails?*.

1.3.2.6.9 Step 9: Verify topology activation

```
dpsctl topology activate --topology maxlps-pilot
dpsctl tp list --active
```

Confirm the topology appears, `leaf_node_names` lists all 72 compute nodes, and `policy_bundle` matches the `global_policy_bundle` that was in effect at import.

```
dpsctl check connection --topology maxlps-pilot
```

The check must succeed for every pilot node; otherwise stop and correct the agent ID assignment, agent-to-BMC network path, or BMC credentials before the health gate. For activation failures, refer to *What If Topology Activation Fails?*.

1.3.2.6.10 Complete the BMC Health Gate

Run the full BMC health check against the active topology, review its latency profile, and resolve every error before continuing. Keep the topology active until the check finishes and you have saved its report. Refer to *BMC Readiness and Health Guide* for the procedure, slow-BMC guidance, and issue-specific recovery.

1.3.2.6.11 Deactivate the Topology for the Baseline

After saving the health report, deactivate the topology to remove DPS from the baseline power-control path:

```
dpsctl topology deactivate --topology maxlps-pilot
dpsctl tp list --active
```

Confirm that no topology is active. Deactivation cleanup is best effort. On all 54 baseline compute nodes, check every GPU:

```
nvidia-smi --query-gpu=index,power.default_limit,power.limit --format=csv
```

Every GPU must report its default limit before you start the baseline. For limits that remain changed, follow the reset procedure in *Step 3: Verify per-GPU caps returned to default*, then recheck every af-

fected GPU. Do not run either BMC command while the topology is inactive in 0.9. If you need another check, repeat the activation, check, and deactivation sequence before the baseline.

1.3.2.7 Step 4: Prepare the Workload and Comparison Controls

1.3.2.7.1 Workload Requirements

What "baseline" means here. The baseline run is the unmanaged reference. DPS is installed and running, but the imported topology and resource group are not active, so DPS does not enforce power policy on any node. The topology is deactivated after the BMC health check. Confirm that `dpsctl tp list --active` returns no rows and `dpsctl rg list` returns no resource groups before you launch the workload.

For the 3-rack baseline and the 4-rack DPS-managed run to be directly comparable, the workload mix must meet these requirements:

- **Each job is an inference workload.** Training workloads, synthetic power stressors (`gpu-burn`, etc.), and pure benchmarks do not exhibit the bursty, non-sustained power profile that envelope-based inference pilots target and will produce misleading results.
- **It includes at least two complementary power profiles.** Run at least one low-power-utilization and one high-power-utilization representative inference job.
- **Each job is representative of the production workload.** Ideally it uses the actual model, framework, and request pattern (batch size, concurrency, input length distribution) you plan to run once the pilot succeeds.
- **It runs long enough to reach a steady-state comparison window.** The "steady-state window" is the contiguous slice of the run that you and the 4-rack managed run will be compared on in *Step 8*. Identify it as follows, and apply the same procedure on every run:
  - Run the workload mix for at least **one hour total** so the run is not dominated by ramp-up.
  - **Trim ramp-up and cool-down** from the start and end of the run.
  - Within the trimmed interval, require that **both per-rack power draw and aggregate tokens/sec have visibly flattened** on your time-series plots — no monotonic climb, no large step changes, only normal short-term workload jitter.
  - The remaining window must be **at least 30 minutes long and contiguous**; if it is shorter or broken up, extend the run rather than relying on a noisy window.
  - Record the start and end timestamps of this window and use **exactly those timestamps** when capturing every metric in *What to Collect*. The same procedure applies to the 4-rack managed run in *Step 7* and to the comparison in *Step 8*.
- **The 3 pilot racks are under load in isolation.** No other workloads should be sharing these racks or competing for the same facility power budget during the baseline window; otherwise the "3-rack baseline" no longer describes what the 3 racks alone can do.

1.3.2.7.2 Comparison Controls

Fix the client-side controls before starting the baseline, and retain them for the managed run:

- **Fleet-wide demand.** For open-loop testing, use the same aggregate QPS schedule across all load generators for each inference job. Do not change per-node QPS just because the managed run has more nodes. For closed-loop testing, use the same aggregate configured concurrency and record the configured concurrency for each generator and the total.

- **Client behavior.** Keep request timeouts, retry and backoff behavior, cancellation policy, warm-up, total duration, and stop conditions the same.
- **Workload placement.** Record the job-to-node placement and the nodes under load. Do not add unrelated workload to the pilot racks during either steady-state window.

1.3.2.7.3 What to Collect

Before you run the baseline inference test, capture the following for the duration of the baseline run. You will compare these signals against the 4-rack managed run in *Step 7* and interpret them using *Step 8*:

- **Aggregate workload throughput** — requests/sec, queries/sec, or whichever KPI matches the workload.
- **Aggregate token throughput** (tokens per second across the cluster) and **latency percentiles** (refer to Step 8 for the KPI set).
- **Workload metadata and configuration** - application, model, ISL/OSL, and any other relevant configuration parameters.
- **Per-rack and aggregate power draw**, sampled at a consistent interval. Keep both the time series and a steady-state average.
- **Wall-clock duration** of the run and of the steady-state window within it.
- **Per-GPU power draw distribution** across the 3 racks, if available — useful for showing how headroom is being left on the table by static provisioning.
- **EDPp settings** per node. The same EDPp setting should be used across the entire fleet.
- **BMC health report and latency profile** — retain the full JSON report from *Step 3*, including node-level read, telemetry, and power-write latencies.

1.3.2.7.4 Retain an Evidence Package for Every Run

For both baseline and managed runs, retain an immutable package that includes:

- the workload manifest or launch command, image and model version, dataset or prompt profile, batching, aggregate QPS or configured concurrency, load generator allocation, and client timeout, retry, and stop settings;
- raw workload, power, GPU-limit, and BMC telemetry series; their UTC run and steady-state bounds; and the clock source and any known offsets;
- topology and resource-group state, active node lists, DPS and `dpsctl` versions, deployment or Helm values relevant to Power Steering, and the BMC health report; and
- the source used for every aggregate in the comparison.

Do not put BMC credentials, tokens, or other secrets in this package. Redact hostnames and endpoint details where required by your site's retention policy.

Use whatever stack you already trust for time-series capture (BMC or rack telemetry, DCGM or vendor exporters, inference server metrics, facility BMS). The important part is **consistent sampling**, **aligned clocks**, and **the same definitions** on baseline and managed runs.

The baseline run establishes the reference numbers the 4-rack DPS-managed run will be compared against. It is run **without DPS in the control path**, after deactivating the topology used for health checks and before creating a resource group.

This runbook does **not** prescribe *how* you launch the workload. Whatever you already use to orchestrate AI inference on these racks — Slurm, a Kubernetes inference operator, a Triton deployment, your own job runner — is exactly what you should use here. DPS cares about the power side of the run, not the scheduling side.

1.3.2.8 Step 5: Run the Baseline

Launch the workload mix across 3 of the 4 pilot racks (54 compute nodes) using your existing orchestration tooling. DPS should not be in the control path during this run.

Record the number of compute nodes that are actually under load during the baseline run. You will compare that footprint with the 4-rack managed run in *Part 8*.

The topology remains imported but inactive, and no resource group exists yet, so DPS is not in the control path during this baseline run.

1.3.2.9 Step 6: Enable MaxLPS

Back to the five-step pilot overview.

1.3.2.9.1 Confirm Power Steering Settings

Before enabling MaxLPS, confirm that the Helm values used for the deployment enable dynamic power management and Power Steering:

```
dps:
  prs:
    enabled: true
power-steering:
  enabled: true
```

Both values default to `true` in DPS 0.9. Keep them explicit in the pilot overlay so a chart customization cannot omit the Power Steering controller.

1.3.2.9.2 Reactivate the Topology

The topology passed BMC health checks and was deactivated for the baseline. Reactivate it now before creating the resource group:

```
dpsctl topology activate --topology maxlps-pilot
```

Troubleshooting: If topology activation fails, see *What If Topology Activation Fails?*.

1.3.2.9.3 Verify the Topology Is Active

```
dpsctl tp list --active
```

Confirm `maxlps-pilot` appears as the active topology and its `policy_bundle` matches the `global_policy_bundle` that was in effect at import.

1.3.2.9.4 Select the Default MaxLPS Policy Bundle

Use the default `MaxLPS` policy bundle. Do not create a pilot-specific policy bundle. The bundle contains model-scoped limits for dynamic power management and Power Steering on the supported systems; its GPU values are aggregate watts for all GPUs in one `ComputerSystem`, not per-GPU watt values.

Confirm the default bundle is available before creating the resource group:

```
dpsctl policy bundle list
```

Confirm that the output lists `MaxLPS`. If it does not, stop and reconcile the DPS installation before continuing. Do not replace the default bundle with a pilot-specific bundle.

Bind the shipped `MaxLPS` bundle on the resource group. Do not create a custom percentage bundle for this pilot. Confirm the name with `dpsctl policy bundle list` before you create the group.

`MaxLPS` does not cap each node at a fixed fraction of MaxP. It sets a per-model operating band and an aggregate GPU target below the GPU ceiling: for `DGX_GB300`, a 100% node ceiling with a 4,108 W GPU target against a 5,600 W GPU ceiling, and for `DGX_GB200`, a 100% node ceiling with a 3,648 W GPU target against a 4,800 W ceiling.

The 3-rack envelope is held by the topology instead. The `PDU-Root PowerDomain` limit set in *Part 2* bounds all four racks, and DPM, PRS, and the shared-GPU budget redistribute power under that bound as demand moves. That is what lets 72 managed nodes run inside an envelope sized for 54: DPS admits nodes at their effective policy floor when the target does not fit, then steers power toward the nodes actually under load. Expect activation to report some nodes below their policy target — that is the mechanism working, not a failure. Refer to *What If Resource Group Activation Fails?*.

1.3.2.9.5 Create the Resource Group

Create one resource group containing every compute node in the topology, with dynamic power management, Power Steering, and Shared GPU all enabled. The `--shared-gpu-enable` flag is what binds the resource group's GPU power consumption to a shared, group-wide budget rather than per-node caps. Set `--policy-bundle` to the default `MaxLPS` bundle selected in Step 3.

```
dpsctl resource-group create \
  --resource-group maxlps-pilot \
  --external-id 1 \
  --dpm-enable \
  --prs-enabled \
  --shared-gpu-enable \
  --policy-bundle MaxLPS
```

Use `MaxLPS` for both GB200 and GB300 pilots.

Troubleshooting: If resource group creation fails, see *What If Resource Group Creation Fails?*.

`--external-id` is **required** and must be a 64-bit integer. It is the handle a workload scheduler or provisioning system would normally use to correlate this resource group with one of its own records. In the scheduler-independent configuration this runbook describes, no external system owns the resource group — pick any positive integer and keep a note of it for your own records. DPS does not enforce uniqueness on this value, but using a distinct ID per resource group makes *`dpsctl rg list`* output easier to read once you have more than one.

1.3.2.9.6 Add Every Compute Node

Add every compute node in the topology to the resource group. Substitute your real node names for `gb300-nvl-001-compute01,gb300-nvl-001-compute02,...`:

```
dpsctl resource-group add \
  --resource-group maxlps-pilot \
  --entities "gb300-nvl-001-compute01,gb300-nvl-001-compute02,gb300-nvl-001-
↪compute03..."
```

See *`dpsctl resource-group add`* for additional flags (partial-update behavior, strict policy, etc.).

Troubleshooting: If adding resources fails, see *What If I Cannot Add Nodes to the Resource Group?*.

1.3.2.9.7 Activate the Resource Group

Activate the resource group. Use `--sync` so the command blocks until every node has its policy applied, `--strict-policy` so an infeasible target does not fall back to its effective floor, and `--allow-reprovision=false` to prevent the pilot from changing any other resource group:

```
dpsctl resource-group activate \
  --resource-group maxlps-pilot \
  --strict-policy \
  --allow-reprovision=false \
  --sync
```

See *`dpsctl resource-group activate`* for the full activation flag set, including strict-policy and partial-timeout options.

1.3.2.9.8 Verify Resource Group Activation

```
dpsctl rg list --active
```

Confirm the `maxlps-pilot` resource group appears with:

- `activation_status: active`
- `dpm_enable: true`
- `prs_enabled: true`
- `shared_gpu_enable: true`
- `resource_names` listing all 72 compute nodes

Troubleshooting: If resource group activation fails, see *What If Resource Group Activation Fails?*.

1.3.2.9.9 Verify Power Steering Readiness

After the resource group is active, confirm the Power Steering controller is Ready and has no reconciliation errors. The 0.9 Power Steering 2.1 controller does not use older controller log text as a stable interface, so controller readiness and reconciliation health are the acceptance evidence for this step.

```
kubectl get pods -n dps \
  -l app.kubernetes.io/component=controller,app.kubernetes.io/part-of=power-
↪steering
kubectl logs -n dps \
```

(continues on next page)

(continued from previous page)

```
  -l app.kubernetes.io/component=controller,app.kubernetes.io/part-of=power-
↪steering \
  -c controller --since=10m
```

Troubleshooting: If the Power Steering controller is not Ready or reports errors, refer to *How Do I Check Power Steering?*.

1.3.2.10 Step 7: Run the MaxLPS Test

Back to the five-step pilot overview.

The **only intended changes** between *Step 5* and this step are the number of racks under load and that DPS is now actively managing their power. Workload mix, model, framework, batching, fleet-wide demand, client behavior, controller timing, and the operating envelope are unchanged. If anything else differs, the comparison in *Step 8* is invalid.

The two runs differ on exactly two dimensions:

Dimension	Baseline (Step 5)	Managed (Step 7)
Racks Under Load	3 racks (54 compute nodes)	4 racks (72 compute nodes)
DPS Control Path	Not in control path; no topology and no resource group active	In control path; topology `maxlps-pilot` active and resource group `maxlps-pilot` active
Operating Power Envelope	Same	Same
Workload, Model, Framework, and Batching	Same	Same
Fleet-wide Demand	Same aggregate QPS schedule (open loop) or configured concurrency (closed loop)	Same
Client Behavior	Same timeout, retry, cancellation, warm-up, duration, and stop rules	Same
Power Steering Timing	Not controlling power	Same Power Steering scheduler interval recorded for the comparison

The **Operating Power Envelope** row is held by the topology in the managed run: the active `PDU-Root` `PowerDomain` already sets the same 3-rack operating limit used by the baseline, and DPS enforces that limit across all 4 racks through the resource group's shared-GPU policy.

With the topology active and the **pilot resource group** (`maxlps-pilot`) in place—created and maintained **without** integration to your workload schedulers **or** provisioning systems—re-run the **same** workload mix you used for the *3-rack baseline*—same jobs, model, framework, and request pattern—but now across **all 4 racks** (72 nodes), with DPS in the control path.

As in *Step 5*, this runbook does **not** prescribe *how* you launch the workload mix. Use the same orchestration tooling (Slurm, Kubernetes inference operator, Triton, your own job runner) you used for the baseline. The only thing that changes between the two runs is the number of racks under load

and whether DPS is enforcing power policy. Everything else must stay the same so the comparison is meaningful.

1.3.2.10.1 Workload Requirements

The 4-rack run must be directly comparable to the 3-rack baseline. In addition to the baseline's requirements:

- **It is the same workload mix as the baseline.** Run the same low-power-utilization and high-power-utilization inference jobs with the same model, framework, batch size, and input distribution. If anything about the workload mix changes, the throughput numbers are no longer comparable.
- **It uses the same fleet-wide demand and client behavior.** Use the aggregate QPS schedule or aggregate configured concurrency, timeout, retry, cancellation, warm-up, duration, and stop rules recorded under *Comparison controls*. Do not compensate for the larger fleet by changing per-node QPS.
- **All 4 pilot racks are under load in isolation.** No other workloads should be sharing these racks or competing for the same facility power budget during the run. The same operating envelope that constrained the 3-rack baseline should constrain the 4-rack run.
- **DPS is in the control path.** Confirm the pilot topology from *Step 3* and the active resource group from *Step 6* are in place (`dpsctl tp list`, `dpsctl rg list`) and that a read-only BMC health recheck still passes (`dpsctl verify bmc-health start --topology maxlps-pilot --skip-writes --wait --summary-only`) before launching the workload.
- **It produces a comparable steady-state window.** DPS's per-GPU caps are set dynamically from telemetry, so the run needs enough duration for the control loop to settle into steady-state redistribution, not just the workload itself. Apply the **same steady-state window procedure** defined in *Step 4* under "Workload Requirements", and make the resulting window **at least as long** as the one used for the 3-rack baseline.

1.3.2.10.2 Enable DPS Prometheus Metrics

In DPS 0.9, `dps-server` exposes Prometheus text metrics at `/metrics` on the dedicated `<release>-server-metrics` Service (for example, `dps-server-metrics`) at port `9090`, with service port name `dps-metrics`. The direct listener uses HTTP and is separate from the main HTTP API Service. Enable scraping for the pilot so the DPS-side signals called out in *What to Record* — per-GPU cap movement and group-sum allocation versus observed aggregate draw — land in the same time-series store as the workload and BMC telemetry you captured in *Step 4*, and can be cross-checked against the steady-state window with PromQL. The chart-managed `ServiceMonitor` (`dps.serviceMonitor.enabled: true`) is the recommended path for in-cluster Prometheus and is applied through the Helm upgrade flow in the *Deployment and Configuration Guide*; refer to *Prometheus Monitoring* for the full setup, including external Prometheus and protected-metrics configurations. If you enable `dps.metrics.authentication`, it requires `dps.auth.oidc.enabled=true` and a complete OAuth2 or authorization profile for the ServiceMonitor with the required `dps:metrics:read` scope.

> **Note:** Land this Helm change **once, before** the 4-rack managed run starts so the steady-state window in *What to record* is fully covered by Prometheus samples.

1.3.2.10.3 Run the Workload

Launch the workload mix across all 4 pilot racks (72 compute nodes) using the same orchestration tooling and submission flow you used for the 3-rack baseline. DPS is now in the control path and

will actively redistribute per-GPU power headroom across the resource group in response to live BMC telemetry.

1.3.2.10.4 Verify GPU Limit Movement

After the workload reaches steady state, confirm that the BMC or DCGM limit time series shows at least one GPU limit change while aggregate power remains within the operating envelope. Do not mark the pilot healthy solely because a pod is Running or a legacy log phrase appears. If limits do not move, refer to *How Do I Check Power Steering?*.

1.3.2.10.5 What to Record

Capture the **same metrics you captured for the 3-rack baseline**, over a comparable steady-state window, so the two runs can be compared directly. Additionally, capture the **DPS-managed run** observations below.

Additionally, specific to the DPS-managed run:

- **Per-GPU cap movement during the run** — the per-GPU `power_max` that DPS is actively setting, sampled over time. This demonstrates DPS redistributing budget live, rather than statically provisioning.
- **GPU power allocations vs. observed aggregate draw** — confirms the group sum is being enforced rather than per-node caps leaving headroom on the table.
- **DPS-provided Prometheus metrics** — once scraping is enabled per *Enable DPS Prometheus metrics*, capture the dedicated DPS metrics service's `/metrics` series for the same steady-state window so per-GPU cap movement and group-sum enforcement can be cross-checked against PromQL.

To sample **per-GPU caps** and utilization over time, we recommend using DCGM on a representative node set on a fixed interval (for example every 60 seconds via a simple loop or your automation), or export the same fields from your existing BMC/telemetry pipeline. Store `set_limit` / `max_limit` alongside observed GPU power so cap movement can be plotted against workload phase.

1.3.2.11 Step 8: Compare Results

Compare the raw captures from *Step 4*, *Step 5*, and *Step 7*. **Compare runs only on their steady-state windows** — at minimum 1 hour total run time, ramp-up and cool-down trimmed, both per-rack power and aggregate tokens/sec visibly flattened, contiguous window of at least 30 minutes, and the exact start/end timestamps recorded. The full procedure is defined in *Step 4* under "Workload Requirements." **Do not compare headline numbers unless the steady-state windows, workload mix, fleet-wide demand, client behavior, and controller timing match** between the 3-rack baseline and the 4-rack managed run.

1.3.2.11.1 Pilot success quick-check

Before reading the KPI tables, confirm the run satisfies every pass condition below. A miss on any row means the pilot did not meet its goals on this configuration.

Check	Pass condition	Source KPI
Compliance	Zero new compliance events in the managed run's steady-state window	Compliance events (*Data center power and compliance KPIs*)
Throughput uplift	Aggregate output tokens/sec strictly greater than baseline at matched load	Aggregate output tokens/sec (*Inference service KPIs*)
Service stability	Error rate matches baseline within ±0.1 percentage points	Success / error rate (*Inference service KPIs*)
Envelope adherence	Overall observed power binding data center constraint for ~100% of steady-state samples	Overall observed power (*Data center power and compliance KPIs*)
Experiment validity	Workload mix, fleet-wide demand, client behavior, controller timing, and steady-state selection match; both evidence packages are complete	*Comparison controls* and *evidence package*

1.3.2.11.2 Time alignment

- **Clock authority.** Synchronize every sampling agent — BMC/BMS exporter, DCGM, inference server, load generator — to the **cluster's NTP/chrony source** (typically the cluster head node or site-wide NTP server) before either run. All timestamps in this step are interpreted relative to that authority. If a sampling agent cannot sync to the cluster source, document the offset and treat its data as advisory, not authoritative.
- Use the **same wall-clock sampling interval** (or finer on the managed run) for power and service metrics.
- If you change *anything* about the client (QPS, concurrency, prompt distribution, batching, timeout, retry, or stop behavior), treat the run as a new experiment—do not attribute deltas to DPS.

1.3.2.11.3 Interpret offered load correctly

- **Open-loop tests:** The offered request rate is the configured aggregate QPS schedule, independent of completion rate. Report offered QPS, completed QPS, success and timeout counts, queueing, and TTFT. A capacity shortfall can appear as a growing queue and worse latency even when offered QPS is fixed.
- **Closed-loop tests:** New requests replace completed requests, so observed QPS changes with system capacity. Report configured and observed aggregate concurrency, completion rate, success and timeout counts, and latency.

Do not compare an open-loop run with a closed-loop run, or compare results from different aggregate demand settings as a MaxLPS result.

1.3.2.11.4 Data center power and compliance KPIs

The definitions below assume you can obtain **an overall facility or pilot-segment power budget** (the *data center constraint*), **provisioned electrical or thermal capacity** for the GPUs or systems under test (from nameplate TDP, derated caps, or your capacity model), and **metered or telemetry-derived actual draw** for the same boundary. Map each to your monitoring source (BMS, rack PDU, BCM, or aggregated BMC).

KPI	Definition	What to expect in a successful pilot
Compliance events	Distinct count of intervals (or events) where **overall observed power** exceeds **any applicable data center constraint** (contract limit, breaker headroom, pilot `PowerDomain` budget—use the tightest binding constraint you track).	**Target: zero** new compliance events in the managed run's steady-state window. The baseline may already be clean; the managed run must not introduce excursions.
Overall provisioned power	**Total system power (provisioned)** — sum (or budgeted envelope) of provisioned capacity for the systems in scope (for example aggregate GPU or node TDP limits you *could* enforce statically).	**Between runs:** provisioned **total** for the 4-rack fleet can be **higher** than for the 3-rack baseline because more hardware sits in the topology, while the **same pilot envelope** caps *observed* aggregate draw. You are measuring whether you *provisioned* more compute than naive sizing would have allowed under that envelope.
Overall observed power	**Total system power (observed utilization)** — metered or telemetry-summed draw for the same boundary as provisioned and constraints.	**Must remain** □ the binding data center constraint for essentially **100%** of steady-state samples on the managed run (allow rare spikes only if your constraint definition already includes agreed crest factor). Expect observed **at or below** the pilot envelope you set in topology.
Capacity efficiency	*Power budget* minus *total system power (provisioned)*.	Interpret as **headroom between envelope and what you provisioned on paper**. With MaxLPS-style sizing, this number is often **smaller** than in conservative MaxP-only planning because you deliberately **provision closer to the envelope**—that is expected if compliance stays clean.
Provisioning efficiency	*Total system power (provisioned)* minus *total system power (observed utilization)*.	Interpret as **stranded provisioned headroom** not appearing as draw. You often want this gap to **shrink** when inference loads the fleet more evenly; a **large persistent gap** under load suggests underutilization or caps binding unevenly.
Resource group efficiency	*Aggregated average power (W)* divided by *aggregated provisioned TDP limit (W)* for all members of the pilot resource group, using the same time window.	Ratios in roughly **0.35–0.85** are common for bursty inference once warm; values **sustained below 0.25** under declared steady load suggest the workload or caps are not exercising the fleet, while values **sustained above 0.95** leave little margin for spikes—watch compliance.
Node efficiency	*Node average power (W)* divided by *node provisioned TDP limit (W)* per compute node, then compare distributions across racks.	Expect a **similar spread** to baseline at matched load, with managed runs sometimes showing **higher** averages on more nodes because Power Steering moves limits toward demand. Flag nodes **stuck below 0.2** while peers approach limits—possible mapping, cooling, or telemetry issues.

If your tooling cannot yet compute one of the aggregates exactly as defined, document the **substitute signal** (for example sum of BMC inlet metrics vs. rack PDU) and apply it **consistently** on both runs.

1.3.2.11.5 Inference service KPIs

These metrics come from the inference stack (vLLM, Triton, TensorRT-LLM, proprietary routers, etc.), not from DPS. Prefer **server-reported** token and latency counters over client-only estimates.

KPI	Definition	What to expect in a successful pilot
Aggregate output tokens/sec	Total **output** tokens completed per second across all replicas in scope (cluster-wide), measured over the steady-state window. Exclude failed generations if your stack exposes **goodput** separately.	**Strictly greater** than the 3-rack baseline steady-state value when load generator settings are unchanged—this is the headline uplift. Relative uplift depends on how much headroom the baseline left; **single-digit to mid-double-digit percent** improvements are plausible when the fleet was envelope-starved, but **your measured baseline** is the reference, not a fixed percentage from this table.
Aggregate total tokens/sec	**Input + output** tokens per second if your economics or SLAs track both.	Should move **roughly in proportion** to output tokens/sec unless the prompt length distribution changed.
Tokens/sec per concurrent user (or per active session)	Aggregate output tokens/sec divided by a **defined concurrency** measure (open sessions, active clients, or in-flight requests—pick one and keep it fixed). Proxy for **interactive saturation**.	At **fixed** concurrency, expect **within ~±15% of the baseline** value if power is not the dominant bottleneck; a **large sustained drop** warrants checking for cap-induced queueing or straggler GPUs.
Time to first token (TTFT)	Latency from request accept to first output token. Report **p50/p95/p99** over the window.	Expect **p95 within ~1.2× of baseline p95** and **p99 not more than ~1.5× baseline p99** when configuration is healthy—Power Steering can redistribute headroom and introduce modest tail movement. Investigate if p99 regresses **beyond ~2×** baseline.
Inter-token latency (ITL) / **decode token interval**	Spacing between successive output tokens (stream quality).	Similar tolerance band as TTFT: **p95 ~1.2× baseline**, **p99 ~1.5× baseline** as a first-pass gate; tighten to your product SLO if stricter.
End-to-end request latency	Wall time for full response.	Same percentile bands as TTFT/ITL for streaming workloads; for non-streaming, this is often the primary user-facing metric—use the same **~1.2× / ~1.5×** p95/p99 gates unless your SLO says otherwise.
Success / error rate	Fraction of requests completing without server-side error.	**Match baseline within ±0.1 percentage points** on error rate; regressions here invalidate throughput comparisons.

The **±15%**, **1.2×**, and **1.5×** bands above are **starting heuristics** for a first pilot read—not substitute contractual SLOs. Replace them with your own targets where you have historical production data.

1.3.2.11.6 How to document the outcome

1. **Compliance table** — List each binding constraint, observed max, and count of excursions (baseline vs. managed).
2. **Throughput and latency table** — Baseline vs. managed for aggregate tokens/sec, tokens/sec per concurrent user, and p50/p95/p99 for TTFT (or TTFT + E2E).
3. **Efficiency plots** — Time series for overall observed vs. constraint, resource group efficiency, and (optional) per-rack node efficiency histograms.
4. **DPS overlay** — Example plots: per-GPU `set_limit` vs. observed GPU power for a sample of nodes, keyed to the same timestamps as inference metrics.

1.3.2.11.7 Recommended report layout

Use the following one-page layout for the final comparison output. Each section should be a single paragraph, table, or plot — anything longer belongs in an appendix.

1. **Run metadata** — date, operator, topology name, resource group name, policy, workload identifier, model, framework, fleet-wide demand, client controls, and Power Steering scheduler interval.
2. **Steady-state windows** — start/end timestamps for the baseline run and the managed run, plus total run duration.
3. **Clock authority** — name of the NTP/chrony source used and any sampling agents whose offset was outside it.
4. **Pilot success quick-check** — a copy of the *Pilot success quick-check* table from this step with the actual pass/fail per row.
5. **Compliance summary** — binding constraints, observed maxima, and excursion counts (baseline vs. managed).
6. **Power KPI table** — baseline vs. managed values for each row of *Data center power and compliance KPIs*.
7. **Service KPI table** — baseline vs. managed values for each row of *Inference service KPIs*.
8. **Plots** — overall observed power vs. constraint over time; per-GPU `set_limit` vs. observed GPU power for a sample of nodes; (optional) per-rack node-efficiency histograms.
9. **Evidence package** — location and retention reference for the raw telemetry, BMC health report, configuration, and workload records.
10. **Verdict and next action** — pass / fail / partial pass, plus the single next change to make if not a pass (per the discipline in *Next Steps*).

If compliance stays clean, aggregate output tokens/sec rises materially, and latency percentiles remain inside your agreed bands, you have a documented case that the pilot configuration met its goals. If not, capture the same package of telemetry before changing topology, policy, or workload shape so the next iteration is evidence-driven.

1.3.2.12 Cleanup

When the pilot is complete, reverse the configuration in the opposite order you applied it.

1.3.2.12.1 Step 1: Delete the resource group

DPS does not expose a standalone `resource-group deactivate` subcommand — deleting an active resource group automatically deactivates it first. See *dpsctl resource-group delete*.

```
dpsctl resource-group delete --resource-group maxlps-pilot
```

1.3.2.12.2 Step 2: Deactivate the topology

```
dpsctl topology deactivate --topology maxlps-pilot
```

See *dpsctl topology deactivate* for details.

At this point DPS is no longer in the power-control path for the pilot racks.

1.3.2.12.3 Step 3: Verify per-GPU caps returned to default

During topology deactivation, DPS asks each managed node to unset its DPS-managed policy. The GB200 and GB300 device plugins apply a configured idle policy or restore BMC default limits. Device cleanup is best effort, so an unreachable BMC, a manual `nvidia-smi -pl ...` command after deactivation, or another management tool can leave a nondefault limit. Confirm the result on a representative compute node from each pilot rack:

```
nvidia-smi --query-gpu=index,power.default_limit,power.limit --format=csv
```

Every GPU should report `power.limit` equal to `power.default_limit` (within rounding). If any GPU shows a lower `power.limit`, reset it explicitly before moving on:

```
sudo nvidia-smi -i <gpu-index> -pl <default-limit-watts>
```

Repeat on each affected GPU. Do not proceed to *Next Steps* or to the optional uninstall below until every sampled GPU reads back its default limit.

1.3.2.12.4 Step 4: (Optional) Uninstall DPS

If you have more tuning passes planned (different policies or additional workloads from *Next Steps*), **skip this section**. DPS can remain installed indefinitely; with the resource group deleted and the topology deactivated, it is not in the power-control path and is safe to leave in place between runs.

Only if you are done with the pilot and want to remove DPS entirely:

```
helm uninstall dps -n dps
```

Optionally remove the namespace and any BMC secrets created for the pilot:

```
kubectl delete namespace dps
```

1.3.2.13 Next Steps

The pilot topology and decoupled resource group are intentionally small so you can prove behavior end-to-end. After *Step 8* shows the outcome you want, work the following three priorities in order, re-running the compliance and service KPI captures from Step 8 after each change.

1.3.2.13.1 1. Model a larger, more complex topology

Expand beyond the 4-rack sketch to **full rows, feeds, and redundancy paths** your facility actually runs: additional `PowerDomain` boundaries, floor and rack PDUs, diversity groups, and any devices that were simplified or omitted here. The goal is to **practice the same import activate verify loop** on data that resembles a future data center rollout, so surprises surface in the lab—not the first production change window.

Refer to *Topologies* for the entity model, *Import a Topology* for validation and import, and *Managing Topologies* for activation and other lifecycle operations. Rerun the deployment-health gate from *Part 1* and the BMC health gate (`dpsctl verify bmc-health start --topology <name> --wait --summary-only`) from *Part 2* after each major topology change.

1.3.2.13.2 2. Tune power budgets and overprovisioning carefully

Many pilots still show **observed draw below 100% of the binding budget** during steady inference: that is normal when burstiness or conservative caps leave slack. Tune the topology model before you expand the pilot:

- **Topology `PowerDomain`** — The operating envelope is expressed as the domain's **`OperatingLimit`** (the cap DPS enforces against).
- **Topology overprovisioning** — **Overprovisioning** is **not** a Helm chart setting: it is the optional numeric field **`OverProvisioningPercentage`** on the `PowerDomain` entity in your topology JSON. Per the *Topology Schema Reference* (`PowerDomain` definition), provisioning headroom scales with the operating limit as **provisioning limit = operating limit × (1 + `OverProvisioningPercentage` / 100)**. The schema defines the exact type and requires **`OperatingLimit`** when **`OverProvisioningPercentage`** is present. After editing the topology, give the replacement a unique name and import it by following *Import a Topology*. Deactivate the previous topology and activate the replacement by following *Managing Topologies*, then revalidate metrics before comparing runs.

Use small steps on **`OperatingLimit`** and **`OverProvisioningPercentage`** so DPS can hand out **a little more** effective GPU budget—**only** while aggregate telemetry and facility constraints prove you remain safe.

Treat this as an experiment matrix, not a single knob turn:

- Change **one family of settings at a time** (`PowerDomain`'s `OperatingLimit` or `PowerDomain`'s `OverProvisioningPercentage`), apply it through the topology workflow, and re-run a comparable workload window.
- **Stay within roughly 10% of the true electrical or contractual limit** on the binding constraint when you first push outward—enough to learn sensitivity without jumping past derated margins you have not modeled here.
- **Observe heavily**: the same compliance, efficiency, and inference tables from *Step 8* should improve or at least not regress; any increase in **compliance events** or tail latency is a signal to **roll back** that change before stacking the next one.

1.3.2.13.3 3. Experiment with different workloads

Repeat measurement passes with **other inference models**, **batch sizes**, **concurrency levels**, and **prompt length distributions** that matter for your road map—including bursty or seasonal patterns if you can simulate them. The goal is to learn **where token throughput and latency stay acceptable** as power policy moves, not to optimize a single demo model forever. Non-inference loads (training, stress tools) remain poor substitutes for the envelope story this runbook targets.

Once those three are stable, common production-readiness follow-ons are: integrating scheduling and provisioning (*Managing Resource Groups*); promoting Step 8 plots into dashboards and SLOs; running multi-hour or multi-day soaks; validating the topology model against as-built power; defining change control for `PowerDomain` `OperatingLimit` and `OverProvisioningPercentage`; and rehearsing failure and maintenance drills with DPS active. Sequence them by risk — topology fidelity and compliance first, then throughput tuning, then integration and operations.

1.3.3. Collect Pilot Telemetry: 1-Second GPU and CPU Data with DCGM Exporter

Use this support guide to collect the power and utilization evidence for the *Run the MaxLPS Power Pilot*. Return to *MaxLPS Power Pilot* to choose another pilot step.

Jump to a section

- *What This Captures*
- *Before You Begin*
- *Install and Start DCGM Exporter*
- *Verify the Requested Metrics*
- *Capture One Sample per Second to a File*
- *Create the Shareable Tarball*
- *Download the Dashboard Skill*
- *Optional Cleanup*
- *References*

This guide collects a short, shareable power and utilization trace from an NVIDIA GB200, GB300, B200, or H100 system. It uses a pinned DCGM Exporter release and its CSV/CLI configuration interface only. No YAML configuration file is required.

This playbook is pinned to:

- DCGM Exporter `4.8.2`
- Paired DCGM version `4.5.3`
- Container image `nvcr.io/nvidia/k8s/dcgm-exporter:4.5.3-4.8.2-distroless`
- Helm chart `4.8.2`

1.3.3.1 What This Captures

Requested signal	DCGM field used by this release	Scope	Unit
Device-reported GPU power management limit	DCGM_FI_DEV_POWER_MGMT_LIMIT	GPU	W
Device-reported enforced GPU power limit	DCGM_FI_DEV_ENFORCED_POWER_LI	GPU	W
GPU utilization	DCGM_FI_DEV_GPU_UTIL	GPU	%
GPU power draw	DCGM_FI_DEV_POWER_USAGE	GPU	W
CPU power draw	DCGM_FI_DEV_CPU_POWER_UTIL_CU	Grace CPU	W
CPU utilization	DCGM_FI_DEV_CPU_UTIL_TOTAL	Grace CPU core	%

DCGM CPU telemetry is supported for NVIDIA Grace CPUs. Expect all six metric families on a GB200/Grace or GB300/Grace system. On a B200 or H100 node with a non-Grace host CPU, the four GPU metric families should be present and the two CPU metric families will normally be absent. That is expected: DCGM Exporter is not a general-purpose x86 or non-NVIDIA CPU exporter.

The two DCGM power-limit fields are device-side observations. `DCGM_FI_DEV_POWER_MGMT_LIMIT` does **not** show the requested target, and neither DCGM field shows whether a request was clamped before dispatch. Correlate this capture with the per-update debug evidence required by the parent pilot runbook.

1.3.3.1.1 Non-Grace Host CPU Utilization

On B200, H100, or another node with a non-Grace host CPU, collect host CPU utilization through node exporter or an equivalent host collector at the same 1-second interval. Retain both aggregate and per-core data for the complete workload window. `node_cpu_seconds_total` is a cumulative counter; calculate utilization from its non-idle rate rather than treating the raw value as a percentage.

If the host does not already expose CPU metrics, install or enable a supported host collector before the pilot. For a bounded diagnostic when a metrics pipeline is unavailable, `mpstat -P ALL 1` from the `sysstat` package is an acceptable fallback. Record the collector, version, sampling interval, and exact command. Do not report non-Grace CPU power unless a supported platform-specific source provides it.

1.3.3.2 Before You Begin

Run this procedure on the node you need to investigate, or target that node when using Kubernetes.

You need:

- A working NVIDIA driver; `nvidia-smi -L` must list the target GPUs.
- `curl`, `tar`, and Bash.
- Root or `sudo` access for the package and Docker methods.
- Exactly one of the three installation methods in this playbook.
- For Docker, a working NVIDIA Container Toolkit installation.
- For Helm, a Kubernetes cluster that can schedule GPU workloads and `kubectl`/Helm access.

Avoid running the package and Docker methods at the same time because both publish host port `9400` in this playbook.

Create a timestamped capture directory and the collector CSV:

```
export CAPTURE_DIR="${HOME}/dcgm-sre-capture-$(hostname -s)-$(date -u +%Y%m%dT
↪%H%M%SZ)"
mkdir -p "${CAPTURE_DIR}"

export METRICS_CSV="${CAPTURE_DIR}/sre-power-metrics.csv"
tee "${METRICS_CSV}" >/dev/null <<'EOF'
DCGM_FI_DEV_POWER_MGMT_LIMIT, gauge, Device-reported GPU power management
↪limit in watts.
DCGM_FI_DEV_ENFORCED_POWER_LIMIT, gauge, Enforced GPU power limit in watts.
DCGM_FI_DEV_GPU_UTIL, gauge, GPU utilization in percent.
DCGM_FI_DEV_POWER_USAGE, gauge, GPU power draw in watts.
DCGM_FI_DEV_CPU_POWER_UTIL_CURRENT, gauge, Grace CPU power draw in watts.
DCGM_FI_DEV_CPU_UTIL_TOTAL, gauge, Grace CPU core utilization in percent.
EOF

sed -n '1,20p' "${METRICS_CSV}"
```

Each row has exactly three values: DCGM field, Prometheus type, and HELP text. Keep commas out of unquoted HELP text.

1.3.3.3 Install and Start DCGM Exporter

Choose **one** method.

1.3.3.3.1 Method 1: Native DEB or RPM Package

These commands assume that the appropriate NVIDIA CUDA network repository is already configured for the operating system and architecture. Refer to Install DCGM if it is not.

Set `CUDA_MAJOR` to the CUDA user-mode driver major version installed on the node:

```
export CUDA_MAJOR=12  # Change to 13 when the installed user-mode driver is
↪CUDA 13.
```

On Ubuntu or Debian, install the pinned, tested pair:

```
sudo apt-get update
apt-cache madison "datacenter-gpu-manager-4-cuda${CUDA_MAJOR}" datacenter-gpu-
↪manager-exporter
sudo apt-get install --yes --install-recommends \
  "datacenter-gpu-manager-4-cuda${CUDA_MAJOR}=1:4.5.3-1" \
  "datacenter-gpu-manager-exporter=4.8.2-1"
```

On RHEL, Rocky Linux, or another supported RPM-based distribution, install the same pair:

```
sudo dnf --showduplicates list \
  "datacenter-gpu-manager-4-cuda${CUDA_MAJOR}" \
  datacenter-gpu-manager-exporter
sudo dnf install --assumeyes --setopt=install_weak_deps=True \
  "datacenter-gpu-manager-4-cuda${CUDA_MAJOR}-4.5.3-1" \
  "datacenter-gpu-manager-exporter-4.8.2-1"
```

Install the collector CSV and add a systemd override that selects it and changes the DCGM watch interval to 1,000 milliseconds:

```
sudo install -d -m 0755 /etc/dcgm-exporter
sudo install -m 0644 "${METRICS_CSV}" /etc/dcgm-exporter/sre-power-metrics.csv

sudo install -d -m 0755 /etc/systemd/system/nvidia-dcgm-exporter.service.d
sudo tee /etc/systemd/system/nvidia-dcgm-exporter.service.d/sre-capture.conf >
↪/dev/null <<'EOF'
[Service]
ExecStart=
ExecStart=/usr/bin/dcgm-exporter -f /etc/dcgm-exporter/sre-power-metrics.csv -
↪c 1000 -p f
EOF

sudo systemctl enable --now nvidia-dcgm.service
sudo systemctl daemon-reload
sudo systemctl enable nvidia-dcgm-exporter.service
sudo systemctl restart nvidia-dcgm-exporter.service
sudo systemctl --no-pager --full status nvidia-dcgm-exporter.service
```

Use this endpoint for the capture steps:

```
export EXPORTER_URL=http://127.0.0.1:9400
```

1.3.3.3.2 Method 2: Docker Container

The container includes the paired DCGM version, so do not install a separate DCGM package for this method. Run the current release image with the collector CSV mounted read-only:

```
docker run --detach --rm \
  --name dcgm-exporter-sre \
  --gpus all \
  --cap-add SYS_ADMIN \
  --publish 9400:9400 \
  --volume "${METRICS_CSV}:/etc/dcgm-exporter/sre-power-metrics.csv:ro" \
  nvcr.io/nvidia/k8s/dcgm-exporter:4.5.3-4.8.2-distroless \
  -f /etc/dcgm-exporter/sre-power-metrics.csv \
  -c 1000 \
  -p f

docker logs dcgm-exporter-sre
export EXPORTER_URL=http://127.0.0.1:9400
```

On an SELinux-enforcing host, use `:ro,Z` rather than `:ro` on the volume mount if the container cannot read the CSV.

1.3.3.3.3 Method 3: Helm on Kubernetes

This method uses `--set-file` and CLI arguments, so it does not require a values or manifest YAML file. It installs one exporter pod on each eligible node.

```
helm repo add gpu-helm-charts https://nvidia.github.io/dcgm-exporter/helm-
↪charts
helm repo update

helm upgrade --install dcgm-exporter gpu-helm-charts/dcgm-exporter \
  --namespace gpu-monitoring \
  --create-namespace \
  --version 4.8.2 \
  --set-string image.tag=4.5.3-4.8.2-distroless \
  --set serviceMonitor.enabled=false \
  --set-file customMetrics="${METRICS_CSV}" \
  --set-string 'arguments[0]=-m' \
  --set-string 'arguments[1]=gpu-monitoring:exporter-metrics-config-map' \
  --set-string 'arguments[2]=-c' \
  --set-string 'arguments[3]=1000' \
  --set-string 'arguments[4]=-p' \
  --set-string 'arguments[5]=f'

kubectl rollout status --namespace gpu-monitoring daemonset/dcgm-exporter
kubectl get pods --namespace gpu-monitoring --output wide
```

Select the exporter pod on the node being investigated. Replace the placeholder with the Kubernetes node name shown by `kubectl get nodes`:

```
export TARGET_NODE='<target-kubernetes-node-name>'
export EXPORTER_POD="$(kubectl get pods \
  --namespace gpu-monitoring \
  --selector app.kubernetes.io/name=dcgm-exporter \
  --field-selector "spec.nodeName=${TARGET_NODE}" \
  --output jsonpath='{.items[0].metadata.name}')"

test -n "${EXPORTER_POD}" || { echo "No DCGM Exporter pod found on ${TARGET_
↪NODE}" >&2; exit 1; }

kubectl port-forward \
  --namespace gpu-monitoring \
  "pod/${EXPORTER_POD}" \
  9400:9400 \
  >"${CAPTURE_DIR}/kubectl-port-forward.log" 2>&1 &
export PORT_FORWARD_PID=$!
export EXPORTER_URL=http://127.0.0.1:9400
sleep 2
```

1.3.3.4 Verify the Requested Metrics

Check the endpoint before starting the timed capture:

```
curl --fail --silent --show-error --connect-timeout 5 --max-time 5 "$
↪{EXPORTER_URL}/metrics" \
  | grep -E '^DCGM_FI_DEV_(POWER_MGMT_LIMIT|ENFORCED_POWER_LIMIT|GPU_
↪UTIL|POWER_USAGE|CPU_POWER_UTIL_CURRENT|CPU_UTIL_TOTAL)(\{|[[:space:]])'
```

Confirm that:

- GPU sample lines exist for requested limit, enforced limit, utilization, and power draw.
- CPU power and utilization lines exist on GB200/Grace or GB300/Grace.
- CPU utilization has one series per Grace CPU core rather than one host-wide aggregate.
- Missing CPU series on a non-Grace B200/H100 host are expected.

If a GPU series is absent, check the exporter log and confirm that the GPU, driver, and pinned DCGM/exporter pair are healthy before collecting data. For the package or Docker method, start with:

```
nvidia-smi -L
```

For the package method:

```
sudo journalctl --unit nvidia-dcgm-exporter.service --since '10 minutes ago' -
↪-no-pager
```

For Docker:

```
docker logs dcgm-exporter-sre
```

For Helm:

```
kubectl logs --namespace gpu-monitoring "pod/${EXPORTER_POD}" --tail=200
```

1.3.3.5 Capture One Sample per Second to a File

Choose the capture duration in seconds. The default records five minutes. DCGM Exporter watches the fields every 1,000 ms because each installation method passed `-c 1000`; the loop also scrapes the exporter once per second.

```
export CAPTURE_SECONDS=300
export METRICS_FILE="${CAPTURE_DIR}/dcgm-exporter-power-utilization.prom"
export METRIC_PATTERN='^DCGM_FI_DEV_(POWER_MGMT_LIMIT|ENFORCED_POWER_
↪LIMIT|GPU_UTIL|POWER_USAGE|CPU_POWER_UTIL_CURRENT|CPU_UTIL_TOTAL)(\
↪{|[[:space:]])'

: >"${METRICS_FILE}"
export CAPTURE_START_UTC="$(date -u +%Y-%m-%dT%H:%M:%SZ)"
```

On a non-Grace host with node exporter, start a matching host CPU capture before the DCGM loop. Repeat this on every target node or point HOST_EXPORTER_URL at the corresponding node-exporter endpoint:

```
export HOST_EXPORTER_URL=http://127.0.0.1:9100
export HOST_CPU_FILE="${CAPTURE_DIR}/host-cpu-utilization.prom"
: >"${HOST_CPU_FILE}"

(
  for ((sample = 1; sample <= CAPTURE_SECONDS; sample++)); do
    printf '# sample=%d timestamp_utc=%s\n' \
      "${sample}" \
      "$(date -u +%Y-%m-%dT%H:%M:%S.%NZ)" \
      >>"${HOST_CPU_FILE}"
```

(continues on next page)

(continued from previous page)

```
    if host_metrics="$(curl --fail --silent --show-error --connect-timeout 5 -
↪-max-time 5 "${HOST_EXPORTER_URL}/metrics")"; then
      printf '%s\n' "${host_metrics}" \
        | grep -E '^node_cpu_seconds_total(\{|[[:space:]])' \
        >>"${HOST_CPU_FILE}"
    else
      printf '# sample=%d ERROR: host CPU scrape failed\n' "${sample}" \
        >>"${HOST_CPU_FILE}"
    fi

    sleep 1
  done
) &
export HOST_CPU_CAPTURE_PID=$!
```

Run the DCGM capture:

```
for ((sample = 1; sample <= CAPTURE_SECONDS; sample++)); do
  printf '# sample=%d timestamp_utc=%s\n' \
    "${sample}" \
    "$(date -u +%Y-%m-%dT%H:%M:%S.%NZ)" \
    >>"${METRICS_FILE}"

  if exporter_metrics="$(curl --fail --silent --show-error --connect-timeout
↪5 --max-time 5 "${EXPORTER_URL}/metrics")"; then
    printf '%s\n' "${exporter_metrics}" \
      | grep -E "${METRIC_PATTERN}" \
      >>"${METRICS_FILE}"
  else
    printf '# sample=%d ERROR: GPU and CPU scrape failed\n' "${sample}" \
      >>"${METRICS_FILE}"
  fi

  sleep 1
done

if test -n "${HOST_CPU_CAPTURE_PID:-}"; then
  wait "${HOST_CPU_CAPTURE_PID}"
fi

wc --lines "${METRICS_FILE}"
tail -30 "${METRICS_FILE}"
```

The resulting file is a sequence of raw Prometheus samples. Each block begins with a UTC timestamp comment and retains the exporter labels needed to distinguish GPUs, CPUs, and CPU cores. It is a diagnostic capture log, not a single valid Prometheus exposition response.

Record enough system context to interpret the trace:

```
{
  printf 'capture_start_utc=%s\n' "${CAPTURE_START_UTC}"
  printf 'hostname=%s\n' "$(hostname -f 2>/dev/null || hostname)"
  printf 'dcgm_exporter_release=%s\n' '4.5.3-4.8.2'
```

(continues on next page)

(continued from previous page)

```
  printf 'capture_seconds=%s\n' "${CAPTURE_SECONDS}"
  printf 'exporter_url=%s\n' "${EXPORTER_URL}"
  uname -a
  if command -v nvidia-smi >/dev/null 2>&1; then
    nvidia-smi -L
  fi
  if test -n "${TARGET_NODE:-}"; then
    printf 'kubernetes_target_node=%s\n' "${TARGET_NODE}"
    printf 'kubernetes_exporter_pod=%s\n' "${EXPORTER_POD}"
    kubectl get node "${TARGET_NODE}" --output wide
  fi
} >"${CAPTURE_DIR}/capture-info.txt" 2>&1
```

1.3.3.6 Create the Shareable Tarball

Package the raw samples, the exact collector CSV, capture metadata, and any port-forward log into one archive:

```
export ARCHIVE="${CAPTURE_DIR}.tar.gz"
tar --create --gzip \
  --file "${ARCHIVE}" \
  --directory "$(dirname "${CAPTURE_DIR}")" \
  "$(basename "${CAPTURE_DIR}")"

sha256sum "${ARCHIVE}" | tee "${ARCHIVE}.sha256"
tar --list --gzip --file "${ARCHIVE}"
printf 'Share this archive: %s\n' "${ARCHIVE}"
```

Share the `.tar.gz` file and, when useful, its `.sha256` checksum. Review the captured labels before sending the archive outside the intended support channel; hostnames, GPU UUIDs, device identifiers, and Kubernetes workload labels can be operationally sensitive.

1.3.3.7 Download the Dashboard Skill

Download the `DCGM performance dashboard skill` to analyze one or more capture archives. The optional skill retains every valid telemetry point. It does not require a particular scheduler, topology, or workload naming convention.

For Codex, save the downloaded file as `<home>/.agents/skills/dcgm-perf-dashboard/SKILL.md`, creating the directories first. Invoke `$dcgm-perf-dashboard` in your prompt and supply the paths to your capture archives. Codex detects new skills automatically. If the skill does not appear, restart Codex. Refer to Build Skills for supported skill locations. For another agent, follow its documented skill installation and invocation procedure.

For an experiment-wide dashboard, supply an `inventory.csv` file with the columns `hostname,group,role,domain`. The `hostname` column lists the expected nodes. Collect a separate 1-second capture and metadata for each expected node over the experiment window. The skill compares the captures with the inventory to report nodes with missing capture data. Without an expected-node inventory, the output is a partial capture view. Continue to use this capture procedure and collect the required per-update debug evidence.

1.3.3.8 Optional Cleanup

For the package method, stop the one-off exporter configuration and remove the override:

```
sudo systemctl disable --now nvidia-dcgm-exporter.service
sudo rm -f /etc/systemd/system/nvidia-dcgm-exporter.service.d/sre-capture.conf
sudo systemctl daemon-reload
```

For Docker:

```
docker stop dcgm-exporter-sre
```

For Helm:

```
kill "${PORT_FORWARD_PID}" 2>/dev/null || true
helm uninstall dcgm-exporter --namespace gpu-monitoring
```

Do not delete `CAPTURE_DIR` until the archive has been verified and delivered.

1.3.3.9 References

- DCGM Exporter release 4.5.3-4.8.2
- DCGM Exporter versioning
- DCGM CPU monitoring and Grace requirements
- DCGM field identifiers
- DCGM installation
- DCGM Exporter Helm chart

1.3.4. BMC Readiness and Health Guide

Use this guide to prepare Baseboard Management Controllers (BMCs) for DPS and resolve BMC health-check issues. Complete the BMC health gate before you enable DPS power control for a pilot or production resource group.

An unmanaged baseline can proceed when its compute nodes and metrics collection are ready. Do not include a node with an unresolved BMC health issue in a managed DPS run.

Jump to a section

- *Confirm BMC Compatibility*
  - *Limit BMC Client Pressure*
- *Connect DPS to BMCs*
- *Activate the Topology Before Checking BMCs*
- *Run the BMC Health Check*
- *Review the Health Report*
- *Account for Slow BMCs*

- *Adjust DPS Redfish Timeouts*
- *BMC Health FAQ*
  - *Why Is the BMC Unreachable?*
  - *Why Did Firmware Validation Fail?*
  - *Why Do In-Band and Out-of-Band Limits Differ?*
  - *Why Did a Power Write or Restore Fail?*
  - *Why Does the Report Flag WPPS or EDPp?*
  - *Why Does the Report Flag Latency?*
  - *Why Did the Report Probe No Nodes?*
  - *Why Are Metrics Missing or a Setpoint Write Unauthorized?*
- *Collect Evidence Before Escalation*

1.3.4.1 Confirm BMC Compatibility

Before you deploy or configure DPS, confirm that every target BMC exposes the Redfish endpoints required for its hardware. Refer to *Redfish API* for the endpoints DPS uses to collect telemetry and control GPU, CPU, module, or node power.

For GB200 NVL and GB300 NVL systems, use BMC firmware `25.0.0` or later. Record the BMC service's advertised `RedfishVersion` and verify support for the required endpoints; endpoint support is the readiness gate for the pilot. Record the following for each BMC:

- Manufacturer and BMC type, such as OEM-based or NVBMC-based.
- BMC firmware build and Redfish version.
- Supported and unsupported Redfish endpoints.

If an endpoint is missing or behaves differently, capture a sanitized API response and engage NVIDIA support before you add that node to a managed DPS group. Tools such as `redfish-crawler` can help collect that evidence.

1.3.4.1.1 Limit BMC Client Pressure

For GB200 and GB300 systems, use no more than four simultaneous BMC client connections, including two DPS connections. Configure every client to use session-token authentication and keep-alive connections instead of basic authentication or repeated session creation. Refer to the DMTF Redfish Host Interface Specification for connection guidance.

1.3.4.2 Connect DPS to BMCs

For direct BMC access, create a Kubernetes credential Secret for every node and reference its name from that node's `Redfish.SecretName` field in the topology. Set `Redfish.URL` to the BMC endpoint. Do not put BMC credentials in topology JSON. Refer to *Credentials and Secrets Configuration* for the supported credential stores and Secret formats.

DPS 0.9 routes device-management requests through a connected DPS Agent. The chart leaves `agent.enabled` as `false`, so enable and configure an agent before you run a BMC health check or apply hardware policy. Set the topology entities' `AgentId` fields to the matching `agent.agentID`. For a MaxLPS pilot, follow the agent deployment and mTLS configuration in *Step 1 of the pilot runbook*.

If you use a Redfish proxy, complete *Redfish Proxy Configuration* instead. That guide covers proxy routing, authentication, TLS, and topology routing fields. When the proxy owns backend BMC authentication, do not create DPS-owned BMC credential Secrets for that path.

1.3.4.3 Activate the Topology Before Checking BMCs

Note

DPS 0.9 requires an active topology for `dpsctl verify bmc-health start` and `dpsctl check connection`. Activate the target topology first, even for read-only checks or checks scoped with `--topology` or `--nodes`.

Check the active topology with `dpsctl topology list --active`. If the imported target topology is inactive, activate it before either BMC command:

```
dpsctl topology activate --topology <topology-name>
dpsctl topology list --active
```

Activation applies power policies to the nodes. For the MaxLPS pilot, perform it before the baseline workload and leave resource-group creation until after the health gate and baseline. Follow *Activate the Topology Before Checking BMCs*.

1.3.4.4 Run the BMC Health Check

With the target topology active, run the BMC health check. Keep it active until the check finishes and you have saved the report. The check validates Redfish reachability, firmware inventory, power-limit write and readback behavior, telemetry sampling, GPU power-policy state, and in-band and out-of-band power drift.

The full check creates a server task and can write and restore BMC power limits. Obtain authorization for those writes before you start it. It is single-flight across the server, so record the returned `task_id` and use it to query an active health check rather than starting an overlapping one.

```
dpsctl verify bmc-health start \
  --topology <topology-name> \
  --force-writes \
  --expected-edpp-pct 100 \
  --samples-per-telemetry 2500 \
  --telemetry-interval 500ms \
  --per-node-timeout 1h

dpsctl verify bmc-health status <task-id>
dpsctl verify bmc-health report <task-id> --wait --output json \
  > bmc-health-report.json
dpsctl verify bmc-health report <task-id> --summary-only
```

Full checks can take several hours. `--per-node-timeout` applies to each node, not the whole check. Size it for the sample count multiplied by the telemetry interval, plus the firmware, power, EDPp, WPPS, and write phases. Fleet duration also depends on node count and concurrency. For this 2,500-sample command, sampling alone takes about 21 minutes per node, so the default `5m` is too short. Refer to *Run a BMC Health Check* for the operator procedure and report interpretation, and to *bmc-health* for every flag and the asynchronous task model. The server task continues if the local `dpsctl` process exits. Use `status` or `report` with the `task_id` to query its progress.

Before an unmanaged baseline, deactivate the topology and verify default GPU limits as described in *Deactivate the Topology for the Baseline*. For subsequent health checks in 0.9, reactivate the topology first.

Example: Healthy BMC Health Summary

```
{
  "status": { "ok": true, "diag_msg": "all nodes healthy" },
  "cluster_summary": {
    "total_nodes": 72,
    "failed": 0,
    "nodes_unreachable": 0
  },
  "issues": []
}
```

1.3.4.5 Review the Health Report

Save the full JSON report and record these per-node latency summaries:

- `endpoint_latency_ms`
- `telemetry_latency_ms`
- `power_get_latency_ms`
- `power_set_latency_ms`
- `edpp_get_latency_ms`
- `wpps_get_latency_ms`

For a MaxLPS-ready BMC, target p50 below `0.5` s, p95 below `0.7` s, p99 below `0.8` s, and maximum latency near 1 s on read and write paths. If p99 exceeds 1 to 2 s, record the measurements and assess BMC client pressure before you start the managed workload.

Before continuing, confirm all of the following:

- `status.ok` is `true`.
- The report covers every expected node.
- `cluster_summary.failed` and `cluster_summary.nodes_unreachable` are `0`.
- `issues` contains no `SEVERITY_ERROR` entries.

Each issue records a node, resource, code, observed value, threshold, and message. Resolve the issue and rerun a scoped check before you rerun the full topology check.

1.3.4.6 Account for Slow BMCs

Slow BMCs delay the telemetry used for dynamic power management and the readback that verifies power-limit changes. First run a node-scoped read-only check with a longer telemetry interval:

```
dpsctl verify bmc-health start \
  --topology <topology-name> \
  --nodes <node-name> \
  --skip-writes \
  --samples-per-telemetry 10 \
```

(continues on next page)

(continued from previous page)

```
  --telemetry-interval 5s \
  --wait --summary-only
```

`--telemetry-interval` spaces telemetry reads; it does not extend the underlying request deadline. `--write-resolution-timeout` applies only to the post-PATCH readback window of a write-cycle check. It is capped at `60 s` and does not make an unreachable BMC respond faster.

For a confirmed write or readback issue, an authorized operator can run a node-scoped write-cycle check with a longer readback window:

```
dpsctl verify bmc-health start \
  --topology <topology-name> \
  --nodes <node-name> \
  --force-writes \
  --expected-edpp-pct 100 \
  --samples-per-telemetry 2500 \
  --telemetry-interval 5s \
  --write-resolution-timeout 5s \
  --per-node-timeout 4h \
  --wait --summary-only
```

1.3.4.6.1 Adjust DPS Redfish Timeouts

For an agent-enabled DPS deployment, the Helm chart exposes separate Redfish operation timeouts for reads and power-control writes. The rendered default is `1500ms` for both. Use the health report to adjust only the path that is consistently slow but healthy:

- `dps.redfish.connection.get.operationTimeout` covers health checks, telemetry, and other GET calls.
- `dps.redfish.connection.actuation.operationTimeout` covers power-limit and workload-profile writes.

Set a measured value just above the affected path's healthy-node p99, while keeping `dps.redfish.connection.requestTimeout` above either operation timeout. For example, if both read and write latency are healthy but exceed the default:

```
dps:
  redfish:
    connection:
      get:
        operationTimeout: 3s
      actuation:
        operationTimeout: 3s
```

Use `dialTimeout` only when connecting to the BMC is slow, and `sessionCreateTimeout` only when Redfish session creation or refresh is slow. Do not raise these limits to hide an unreachable BMC, authentication failure, or a failed power write. Apply the values through the approved DPS Helm release procedure, then rerun the scoped BMC health check and record the effective values with the report.

The chart renders these values into `dps-agent`; they apply when `agent.enabled: true`. For a Redfish proxy, also verify the proxy and proxy-to-BMC timeouts. Proxy mode uses a shared DPS-to-proxy connection path, so its connection limits are configured separately.

For materially slow but otherwise healthy BMCs, set and record the Power Steering scheduler interval through the approved Helm deployment procedure. Use `0` to retain the `10 s` Power Steering default,

or a value of at least 5 seconds. If BMC latency p99 exceeds 10 s, start with 10; when it is near 30 s, start with 30 and document the choice. Do not change this setting as a BMC preflight; use it only after the health report confirms slow BMCs.

```
power-steering:
  enabled: true
dps:
  prs:
    enabled: true
    schedulerIntervalSeconds: 10
```

Verify that the controller is ready and that the BMC or DCGM limit time series changes under load. Record the scheduler interval with the health report. If it changes after an unmanaged baseline, repeat the baseline and managed pair before comparing results.

1.3.4.7 BMC Health FAQ

Warning

Resetting a BMC, restarting a node, or changing any GPU power setting through an in-band or out-of-band path can interrupt service or change active power limits. Before you take any of these actions, obtain explicit authorization, schedule a site-approved maintenance window, and prepare a documented rollback plan.

1.3.4.7.1 Why Is the BMC Unreachable?

Issue code: `BMC_UNREACHABLE`

The BMC did not respond to the Redfish service root, or the fallback BMC ping failed. In 0.9, first confirm that the target topology is active and the DPS Agent is ready; checks require both before they can reach a BMC.

1. Confirm BMC network reachability from the cluster.
2. Confirm the credential Secret and its `Redfish.SecretName` mapping.
3. Recheck the BMC compatibility inventory and the required Redfish endpoints.
4. Attempt a BMC reset after compatibility and credentials are confirmed.
5. If a different hardware model exposes the required endpoints, capture the evidence and engage NVIDIA support about a custom device plugin.
6. Exclude the node from the managed group if the issue remains unresolved.

1.3.4.7.2 Why Did Firmware Validation Fail?

Issue code: `FIRMWARE_VALIDATION_FAILED`

The fallback health check could not validate the BMC firmware for the selected device model. Recheck the affected node's model, BMC firmware, Redfish version, and required endpoints against the compatibility inventory. If the inventory is correct and the issue persists, retain the sanitized report and engage NVIDIA support before adding the node to a managed group.

The B200 and B300 false-failure exception documented for DPS 0.8.x does not apply to this DPS 0.9 guide. The GB200 and GB300 deep health check records firmware inventory rather than using it as a

health-check gate. Do not treat a FIRMWARE_VALIDATION_FAILED error from a fallback check as an exception.

1.3.4.7.3 Why Do In-Band and Out-of-Band Limits Differ?

Issue code: IB_OOB_LIMIT_DRIFT

1. Preserve the original report, then run the node-scoped write-cycle check to rule out slow readback.
2. Query the current, default, minimum, and maximum GPU power limits:

```
nvidia-smi -q -d POWER | grep -E 'Default Power Limit|Max Power Limit|Min
↪Power Limit|Current Power Limit'
```

3. With authorization, reset the affected in-band limit to the default or maximum TGP value.

```
sudo nvidia-smi -pl <watts>
sudo nvidia-smi -i 0 -pl <watts>
```

4. Rerun the node-scoped check. If drift remains, attempt a BMC reset or node restart. Exclude the node if it cannot return to a healthy state.

1.3.4.7.4 Why Did a Power Write or Restore Fail?

Issue codes: POWER_WRITE_READBACK_MISMATCH, POWER_LIMIT_RESTORE_FAILED

1. Run the authorized node-scoped write-cycle check with a longer telemetry interval and readback window.
2. If it still fails, use the BMC credentials to test the Redfish setpoint write on that node. Refer to *Redfish API*.
3. Verify TGP, and TMP or TCP where the platform supports them.
4. Attempt a BMC reset if the write does not read back correctly. Exclude the node if it remains unresolved.

1.3.4.7.5 Why Does the Report Flag WPPS or EDPp?

Issue codes: WPPS_PROFILES_ACTIVE, WPPS_INACCESSIBLE, WPPS_RESET_FAILED, EDPP_BELOW_REFERENCE

The pilot expects no active workload power profile and an EDPp value of 100%. Reset WPPS through in-band and out-of-band paths where applicable. Reset EDPp where the platform supports it, then rerun the health check. If the settings persist, record the node and values because they affect managed-run performance and baseline comparison.

1.3.4.7.6 Why Does the Report Flag Latency?

Issue codes include TELEMETRY_LATENCY_MAX_HIGH, TELEMETRY_LATENCY_AVG_HIGH, POWER_GET_LATENCY_AVG_HIGH, POWER_SET_LATENCY_AVG_HIGH, POWER_LATENCY_HIGH, EDPP_GET_LATENCY_AVG_HIGH, and EDPP_LATENCY_HIGH.

Use the saved report to identify the affected node and endpoint. The issue-code thresholds are coarse failure thresholds: 5 s average for power GET, power SET, EDPp GET, and telemetry latency, and 10 s maximum telemetry latency. Follow *Account for Slow BMCs*, then reduce BMC client pressure before you change DPS settings.

1.3.4.7.7 Why Did the Report Probe No Nodes?

Issue code: `NO_NODES_PROBED`

Confirm the topology name, requested node names, and that those nodes belong to the imported topology. Then rerun the scoped health check.

1.3.4.7.8 Why Are Metrics Missing or a Setpoint Write Unauthorized?

Confirm that the selected device model supports the required Redfish endpoints and that the BMC credentials have read and write access. Capture the report's node, endpoint, and error message, then check routing and any Redfish proxy configuration. Do not weaken authentication or force another write to diagnose an authorization failure.

1.3.4.8 Collect Evidence Before Escalation

Keep the full BMC health JSON report, its `task_id`, DPS and `dpsctl` versions, the relevant Helm values, and a sanitized window of controller or server logs. Also retain the BMC firmware and Redfish inventory, latency summaries, and affected node names. Redact passwords, tokens, BMC credentials, certificate private keys, and sensitive endpoint details before sharing them.

For control-plane metrics collection, refer to *Prometheus Monitoring*. Prometheus metrics can supplement the health report, but they do not replace the per-node BMC health gate.

For the full MaxLPS workflow, return to *Run the MaxLPS Power Pilot*.

1.3.5. Troubleshoot the MaxLPS power pilot

Choose the question that matches the failed pilot step. Return to *MaxLPS Power Pilot* to choose another pilot step. For failures outside this pilot, refer to *Troubleshooting*.

Jump to a section

- *Why Does DPS Deployment Verification Fail?*
- *Why Does Topology Validation or Import Fail?*
- *BMC Health Check*
- *BMC Health*
- *What If Topology Activation Fails?*
- *What If Resource Group Creation Fails?*
- *What If I Cannot Add Nodes to the Resource Group?*
- *What If Resource Group Activation Fails?*
- *How Do I Check Power Steering?*

1.3.5.1 Why Does DPS Deployment Verification Fail?

Use this section when `dpsctl verify` reports an unhealthy component during *Step 1* of the runbook.

To recheck a single component after a fix, pass its flag — for example `dpsctl verify --database`. Any combination of `--dps-server`, `--database`, `--auth`, `--ui`, and `--bcm` is valid.

- **`dps_server` unhealthy** — Check the DPS server pod is `Running`; `kubectl -n dps logs statefulset/dps-server` usually shows the underlying error.
- **`database` unhealthy** — Check the Postgres pod and the database connection secret; a failed ping is almost always reachability or credentials, not load.
- **`auth` unhealthy** — For LDAP, verify `ServerURL`, `BindDN`, `BindPassword`, and CA/client certificates. For JWT, verify the private key file is mounted and readable.
- **`ui` unhealthy or missing** — Confirm the `dps-ui` Ingress exists in the same namespace as the DPS server with at least one rule and a non-empty host.
- **`bcm` unhealthy** — If BCM is expected, confirm the credential Secret, network path, and API access. If BCM is intentionally omitted, refer to *Deployment Component Unhealthy* for the no-BCM output and gate this step on the other components.

1.3.5.2 Why Does Topology Validation or Import Fail?

Use this section when `dpsctl topology validate <topology-file>.json` or `dpsctl topology import <topology-file>.json` fails during *Part 2, Step 5* of the runbook.

Local JSON syntax errors and JSON-schema failures print instance location (or line, column, and offset) and do not produce `validation_errors`. A non-empty top-level `Policies` array is rejected before the server call with `legacy topology policies are not supported; create policy bundles with dpsctl policy bundle upsert`.

When `dpsctl topology validate` reaches `ValidateTopology`, it prints `ValidationError` entries. Match failures by the `error` field:

- **`invalid_model`** — an entity `Type`/`Model` combination or topology structure is not valid for the device registry. Refer to *Import a Topology*.
- **`device_not_found`** — an entity's `Type`/`Model` pair is not in the DPS device registry (typo in `Model`, or the device was not seeded). Run `dpsctl device list` and correct the `Model` string to an exact match.
- **`invalid_name`** — an entity or topology name does not satisfy the 3–63-character alphanumeric and hyphen rule.
- **`legacy topology policies are not supported`** — the JSON contains a non-empty legacy `Policies` array. Remove it; DPS 0.9 rejects topology-embedded policies.
- **`invalid_secret_name`** — the `Redfish.SecretName` is not a valid Kubernetes Secret name. Use 2–253 lowercase letters, digits, hyphens, or dots, and start and end with a lowercase letter or digit.
- **`duplicate_entity`** — two entities share the same `Name` in the file. Make each node name unique.
- **`referenced_entity_not_found`** / **`referenced_topology_entity_not_found`** — a topology entry lists a child that has no matching entity block. Compare the `Topology.Entities[].Children` list against the top-level `Entities` list.

- **`self_reference` / `circular_dependency`** — a topology entity lists itself or creates a cycle through its children. Rebuild the parent/child chain so every leaf is reached exactly once from the root.
- **`disconnected_graph`** — one or more compute nodes are not reachable from the topology root. Every compute node must be a descendant of the topology root entity.
- **`invalid_connection`** — a parent entity cannot legally have the given child device type (for example, a rack entity listing another rack as a child). Correct the parent/child relationship to match the reference topology examples.

Fix the file, then rerun `dpsctl topology validate` until it prints `Topology validation passed`.

If `dpsctl topology import <topology-file>.json` fails:

- **`failed to create topology`** — Run `dpsctl topology list` first. If `maxlps-pilot` already exists, verify whether it is the expected topology; retrying the unchanged import cannot resolve the name collision. If the name is absent, collect the server and database logs.
- **`failed to add topology entities`** — Check the DPS server and `dps-postgresql` logs, then retry only after identifying and correcting the failure.
- Entity create stream errors — `dpsctl topology import` creates missing entities before topology creation. Inspect the returned gRPC status and server logs.

1.3.5.3 BMC Health Check

Use this section when `dpsctl verify bmc-health` reports issues during *Part 2, Step 6* or *Part 2, Step 7* of the runbook.

Use the `issues` list in the `dpsctl verify bmc-health` report as the starting point. Each issue includes a `node`, `resource`, `code`, `observed` value, `threshold`, and `message`. The *BMC health check guide* covers safe mode selection, report interpretation, and recovery. The *`dpsctl verify bmc-health` reference* documents every command flag and the asynchronous task model.

After fixing an affected node, scope the retry to that node before rerunning the full topology check:

```
dpsctl verify bmc-health start --topology maxlps-pilot --nodes <node-name> --
↪force-writes --expected-edpp-pct 100 --samples-per-telemetry 2500 --
↪telemetry-interval 500ms --per-node-timeout 1h --wait --summary-only
```

If a power-write failure looks like slow asynchronous BMC readback rather than a hard incompatibility, rerun the affected node with a larger write-resolution timeout. Keep the telemetry interval unchanged; increasing it extends the time between telemetry samples and does not extend the power-write readback window.

```
dpsctl verify bmc-health start --topology maxlps-pilot --nodes <node-name> --
↪force-writes --expected-edpp-pct 100 --samples-per-telemetry 2500 --
↪telemetry-interval 500ms --write-resolution-timeout 5s --per-node-timeout
↪1h --wait
```

1.3.5.3.1 BMC Unreachable or Firmware Validation Failed

Issue codes: `BMC_UNREACHABLE`, `FIRMWARE_VALIDATION_FAILED`

The BMC did not respond to the Redfish service root, the fallback BMC ping failed, or firmware validation failed.

On systems with B200 or B300 GPUs, a standalone `FIRMWARE_VALIDATION_FAILED` issue with `message: firmware validation failed` is a known DPS 0.8.x health-check false failure. If the affected B200 or B300 node is still reachable, this is the only `SEVERITY_ERROR` for that node, and no other BMC health checks failed, document the exception and continue the runbook. This exception will be removed in a future DPS version. Treat `FIRMWARE_VALIDATION_FAILED` on other hardware, or any firmware failure combined with `BMC_UNREACHABLE` or another `SEVERITY_ERROR`, as a blocker.

To remediate:

1. Confirm BMC network reachability from the cluster.
2. Confirm the Kubernetes BMC credential secret for the affected node.
3. Confirm the Day 0 BMC compatibility assessment from the runbook prerequisites.
4. Attempt a BMC reset if compatibility and credentials are correct.
5. Review the hardware model if the compatibility assessment shows the required GB200 or GB300 Redfish endpoints but the model is different. DPS might support the node through a custom device plugin. For example, a Supermicro BMC based on NVBMC and OEM extensions for HGX B300 can expose the environment metrics endpoints needed for TGP power settings and metrics, and might be supportable under the GB200 plugin with a custom device definition.
6. Contact the NVIDIA DPS support team if the node appears supportable through a custom device definition.
7. Exclude the node if the BMC remains unreachable. DPS cannot operate on the affected node.

1.3.5.3.2 Power Limit Drift

Issue code: `IB_OOB_LIMIT_DRIFT`

The in-band and out-of-band BMC power limits do not match.

To remediate:

1. Rerun a node-scoped read-only health check to confirm the drift. Telemetry interval and write-resolution settings do not change the in-band and out-of-band comparison.
2. Query the current, default, minimum, and maximum power limits before choosing the reset value:

```
nvidia-smi -q -d POWER | grep -E 'Default Power Limit|Max Power Limit|Min
↪Power Limit|Current Power Limit'
```

3. Reset the in-band GPU power limits to the default or maximum TGP:

```
sudo nvidia-smi -pl <watts>      # Set all GPUs.
sudo nvidia-smi -i 0 -pl <watts> # Set one GPU.
```

4. Rerun the node-scoped health check.
5. Attempt a BMC reset or node restart if drift persists.
6. Exclude the node if the issue cannot be resolved. DPS cannot operate on the affected node.

1.3.5.3.3 Power Write or Restore Mismatch

Issue codes: `POWER_WRITE_READBACK_MISMATCH`, `POWER_LIMIT_RESTORE_FAILED`

DPS wrote a power limit through the BMC, but the readback did not converge to the requested value, or the probe could not verify restoration to the original limit.

To remediate:

1. Rerun the affected node with `--write-resolution-timeout 5s` and inspect the full `power_writes` evidence. Do not increase `--telemetry-interval` for a write-readback failure.
2. Use the DPS BMC credentials to manually test Redfish setpoint writes on the affected node if the issue remains.
3. Follow the *Redfish API guide* examples for `PowerLimitWatts.SetPoint`.
4. Verify TGP, TMP where available, and TCP where available.
5. Attempt a BMC reset if manual writes fail or do not read back correctly.
6. Exclude the node if the issue cannot be resolved. DPS cannot operate on the affected node.

1.3.5.3.4 WPPS or EDPp State

Issue codes: `WPPS_PROFILES_ACTIVE`, `WPPS_INACCESSIBLE`, `WPPS_RESET_FAILED`, `EDPP_BELOW_REFERENCE`

Workload power profile settings (WPPS) or EDPp settings differ from the expected default state. No workload power profile should be active, and EDPp should report the expected current value of `100%` for this pilot gate. WPPS can also affect the EDPp setpoint.

To remediate:

1. Reset WPPS through both in-band and out-of-band paths where applicable.
2. Reset EDPp directly where supported if EDPp remains below `100%`.
3. Attempt a BMC reset if the settings continue to persist.
4. Document the node, active settings, and measured values if no immediate fix is available. These settings affect MaxLPS performance metrics and baselines.

1.3.5.4 BMC Health

In DPS 0.9, both `dpsctl check connection` and `dpsctl verify bmc-health start` require an active topology. If either reports `no agent configuration available`, confirm the topology is active and the agent is configured. Follow *Activate the Topology Before Checking BMCs*, then *Deactivate the Topology for the Baseline* before running the unmanaged baseline.

For BMC compatibility, access mapping, health-check commands, latency tuning, and issue-specific recovery, refer to *BMC Readiness and Health Guide*.

1.3.5.5 What If Topology Activation Fails?

Use this section when `dpsctl topology activate --topology maxlps-pilot` fails during *Step 3* or *Step 6* of the runbook.

- **`topology maxlps-pilot is already active`** — activation was rerun after it succeeded. Run `dpsctl tp list --active` to verify, then continue without deactivating the topology.
- **`topology <name> not found`** / load-topology error — the value passed to `--topology` does not match any imported topology. Confirm the `Topology.Name` in the imported file and rerun `dpsctl tp list`.

1.3.5.6 What If Resource Group Creation Fails?

Use this section when `dpsctl resource-group create` fails during *Step 6* of the runbook.

- **`resource group already exists`** — a resource group named `maxlps-pilot` is already in the database. Run `dpsctl rg list` and verify its configuration. Reuse it only if it is the expected pilot group; otherwise, stop and choose a different name.
- **`resource-group name validation error`** — the `--resource-group` value contains disallowed characters. Use lowercase letters, digits, and `-`, matching the naming convention used elsewhere in this runbook.
- **`policy bundle "MaxLPS" not found`** — Run `dpsctl policy bundle list` and confirm that `MaxLPS` appears. If it does not, stop and reconcile the DPS installation before retrying. Do not replace it with a pilot-specific bundle.
- **`failed to create resource group`** — generic database failure. Check the DPS pod logs and the `dps-postgresql` pod, then retry once the database is healthy.

1.3.5.7 What If I Cannot Add Nodes to the Resource Group?

Use this section when `dpsctl resource-group add` fails during *Step 6* of the runbook.

- **`no active topologies`** — the topology activation step in *Step 6* was skipped or rolled back. Run `dpsctl tp list --active`, then return to that runbook step instead of activating from this troubleshooting page.
- **`some entities are not in the active topology`** — one or more node names in `--entities` are not part of the active topology. Compare the list against `dpsctl tp list --active` and correct the request. If the active topology omitted an intended node, create and activate a corrected replacement topology.
- **`some entities are already in the resource group`** — those nodes are already members of `maxlps-pilot`. Remove the duplicate names from the add request.

1.3.5.8 What If Resource Group Activation Fails?

Use this section when `dpsctl resource-group activate` fails during *Step 6* of the runbook.

- **`resource group maxlps-pilot is already active`** — activation was rerun after it succeeded. Run `dpsctl rg list --active` to verify, then continue without deleting the resource group.
- **`resource group maxlps-pilot has no devices`** — the `resource-group add` step did not land. Run `dpsctl rg list` and confirm `resource_names` is populated before retrying.
- **`power budget exceeded for resource group maxlps-pilot, but reprovision is false`** — admission detected a new or worsened overload in the proposed device or ancestor allocation. Capture the effective policy, active resource groups, topology limits, and reported device metadata. The Power Steering headroom setting does not determine admission.
- **`failed to activate resource group with per-node failures in the --sync response`** — run `dpsctl verify bmc-health start --topology maxlps-pilot --nodes <node1>,<node2> --skip-writes --wait --summary-only` against the failing nodes. If overall `status.ok` is `false`, the server cleans up partially activated nodes. If the request succeeds under its partial-activation rule, accepted nodes remain active. Query `dpsctl rg list --active` before deciding which nodes require remediation.

1.3.5.9 How Do I Check Power Steering?

Use this section when the Power Steering controller is not Ready, reports reconciliation errors, or the observed BMC/DCGM limit time series does not move during *Step 6* of the runbook.

Do not use the older `New power limits:` log phrase as a health check. Instead:

1. Verify the resource group is active with Power Steering enabled.
2. For the bundled deployment, confirm the Power Steering settings are enabled in the Helm values. Refer to *DPS Power Steering Controller*.
3. Verify the controller pod is Ready, then check its logs for errors:

```
kubectl get pods -n dps \
  -l app.kubernetes.io/component=controller,app.kubernetes.io/part-
↪of=power-steering
kubectl logs -n dps \
  -l app.kubernetes.io/component=controller,app.kubernetes.io/part-
↪of=power-steering \
  -c controller --since=15m --tail=200
```

4. Confirm at least one GPU limit changes in the BMC or DCGM time series while the workload is running. If limits remain static, capture the resource-group configuration, controller logs, and matching telemetry window before changing policy or workload settings.

Chapter 2. Deploying DPS

This guide takes you from a Kubernetes cluster to DPS enforcing power policies against your hardware.

Tip

Evaluating DPS rather than deploying it? The *DPS SDK and simulator* runs the complete system with no data center hardware required.

Tip

Running the NVL72 MaxLPS power pilot? Start with *MaxLPS Power Pilot*. It links the pilot preparation, baseline, managed run, and results steps without making you search through the deployment guide first.

2.1. Before you start

You need:

- A Kubernetes cluster to host the DPS services
- Network access to the BMCs of every node you intend to manage
- An understanding of your data center's power distribution architecture, from the utility feed down to individual compute nodes

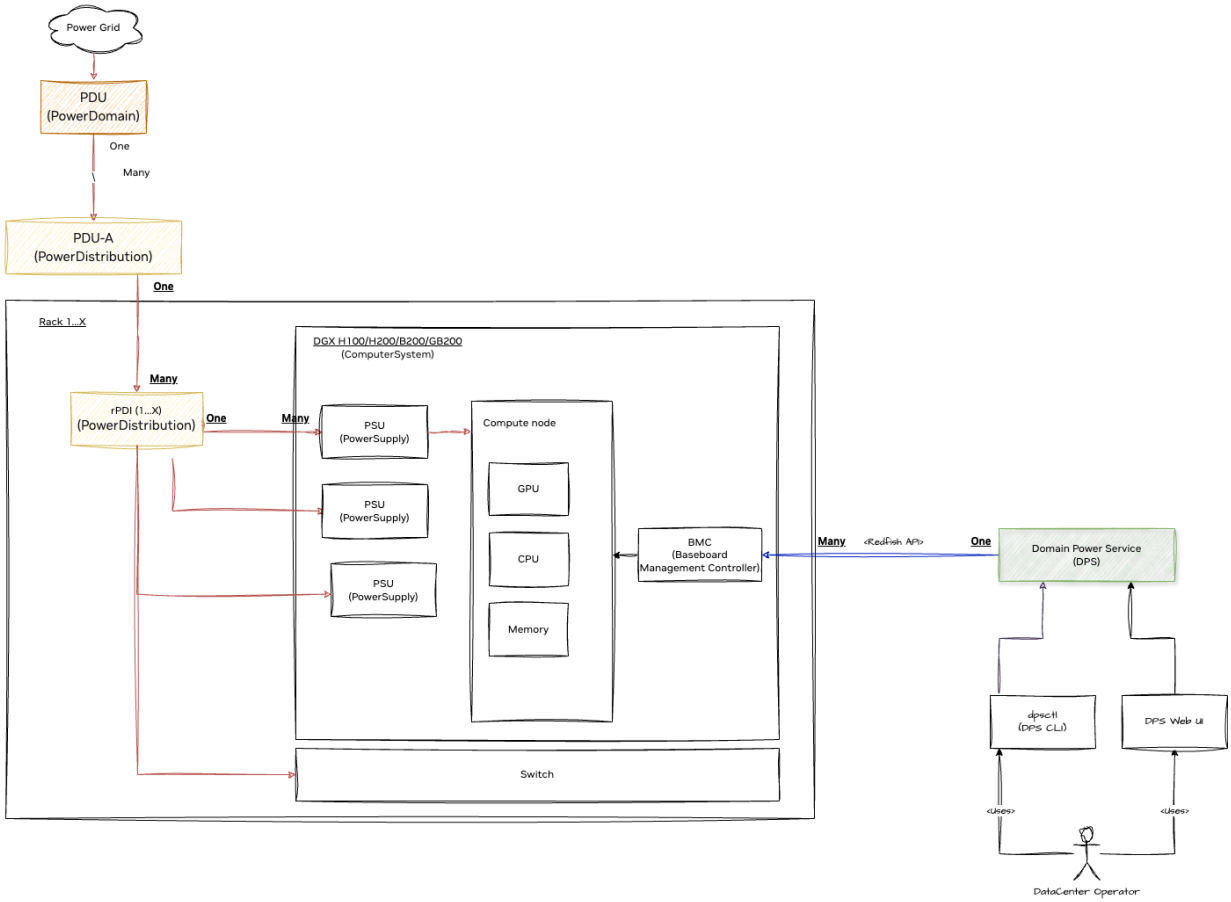
2.2. Key concepts

Four concepts carry most of the deployment:

- ***Topology*** — a representation of your data center's power distribution network, from utility feeds down to individual compute nodes
- ***Power policies*** — rules that define power limits and management strategies for different workload scenarios
- ***Resource groups*** — collections of compute resources that share power budgets and policies
- ***dpsctl*** — the command-line interface for managing DPS

Refer to *Concepts* for the complete set.

2.3. Architecture



2.4. Deployment steps

1. ***Deploy DPS*** — set up the server infrastructure using Helm charts.
2. ***Install dpsctl*** — install the command-line client used for every step that follows.
3. ***Model your topology*** — describe your power distribution network as a topology file.
4. ***Import the topology*** — validate that file and load it into DPS. Imported topologies stay inactive. Activate the topology after import.
5. ***Configure power policies*** — inspect the built-in bundles or create a custom bundle, then select the default to use for the topology import.

Note

Step 3 is the one that cannot be templated. Every deployment needs a custom topology that accurately represents its own power distribution network. Refer to *Managing Topologies* for the full procedure.

2.5. After deployment

If a deployment or operations step fails, use *Troubleshooting* for diagnosis steps and escalation criteria.

- *Operating DPS* — day-to-day procedures: managing devices, resource groups, and topologies
- *Integrating DPS* — connect DPS to schedulers, inventory systems, BMCs, and the grid
- *MaxLPS Power Pilot Runbooks* — the MaxLPS power pilot overview, detailed procedure, telemetry collection, and troubleshooting
- *Troubleshooting* — operator diagnosis and escalation criteria
- *Reference* — the gRPC API, the `dpsctl` command reference, and the topology schema
- *Prometheus monitoring* — collect and visualize power metrics

2.5.1. Installation

Installation covers the first two steps of *Deploying DPS*, which describes the full sequence from a Kubernetes cluster to enforced power policies.

1. ***Deploy and configure DPS*** - Select matching release artifacts, check platform and firmware requirements, prepare security and storage, install with Helm, verify with `dpsctl`, and plan upgrades or recovery
   - To choose CPU, memory, and storage resources, refer to ***System Requirements***
   - To select and configure the server's authentication mode, refer to ***Configure Server Authentication***
   - To sign in through the UI or CLI, refer to ***Sign In to DPS***
   - For network-isolated environments, refer to ***Air-Gapped Deployment***
   - To route Redfish traffic through a proxy, refer to ***Redfish Proxy Configuration***
   - To scrape `dps-server` metrics with Prometheus, refer to ***Prometheus Monitoring***
2. ***Install dpsctl*** - Install the command-line client for administration

With DPS installed, continue with modelling your topology, validating and importing it, and configuring power policies. Refer to *Deploying DPS* for those steps, and to *Import a Topology* for the import procedure itself.

If installation or the first login fails, refer to *Troubleshooting*.

2.5.1.1 In this Section

2.5.1.1.1 Deployment and Configuration Guide

Use this guide to install Dynamic Power Software (DPS) on Kubernetes, secure its external connections, verify the deployment with `dpsctl`, and plan lifecycle operations.

For a network-isolated cluster, prepare the mirrored artifacts in *Air-Gapped Deployment* before you continue.

The deployment is ready for configuration when the pinned Helm release is healthy, the `dpsctl` client and server versions match, the server, database, and identity checks pass, and enabled external endpoints are reachable. For a deployment beyond evaluation, also establish an approved backup and

recovery procedure. Do not enable hardware writes until the topology-specific connection and BMC health gates also pass.

2.5.1.1.1.1 Choose a Release and Its Artifacts

Choose one DPS release for the Helm chart, container images, and `dpsctl`.

1. Review *Release Notes* for release-specific behavior and upgrade notes.
2. Locate the release in the DPS collection on NGC.
3. Add the public NGC Helm repository. The DPS chart does not require an NGC API key:

   ```
   helm repo add ngc https://helm.ngc.nvidia.com/nvidia
   helm repo update ngc
   helm search repo ngc/dps --versions
   ```

   The published DPS chart and container images are public and do not require NGC credentials.
4. Set the selected chart version in your shell. Use the chart version shown by NGC exactly.

   ```
   export DPS_RELEASE="<release-version>"
   ```

5. Inspect the chart metadata before installation:

   ```
   helm show chart ngc/dps --version "${DPS_RELEASE}"
   ```

Install the matching `dpsctl` release by using *Installing dpsctl*.

2.5.1.1.1.2 Requirements

DPS 0.9 has the following minimum requirements:

Area	Requirement
Kubernetes	Version 1.31.x or later.
Helm	Version 3.7 or later. Power Steering requires version 3.12 or later.
PostgreSQL	For a deployment beyond evaluation, use an externally managed service. When Power Steering is enabled, use PostgreSQL 17 or later. The chart's bundled `postgres:18.1-alpine` default is for evaluation, not a production support policy.
Storage	Provide storage for the chart-managed persistent volume claims (PVCs). The chart creates a DPS server PVC and, when the bundled dependencies are enabled, separate PostgreSQL and SeaweedFS PVCs. For dynamic provisioning, configure a default `StorageClass` or component-specific storage classes. Inspect the selected chart's values for their sizes.
Managed systems	Use a topology model implemented by DPS 0.9 and firmware at or later than the validation threshold in *Topology Models and Firmware Checks*.

Before installation, ensure that you have:

- A Kubernetes context for the target cluster and permission to install the chart and its required resources.

- When exposing DPS through Ingress or Gateway API, an Ingress controller or Gateway API implementation, client-resolvable hostnames for the API and enabled UI, and certificates issued by a CA that the clients trust.
- Network access from the installation workstation to `helm.ngc.nvidia.com` and from the cluster nodes to `nvcr.io`. The published DPS chart and images are public and do not require pull credentials. If you copy the artifacts to a private registry mirror, configure the credentials that the mirror requires.
- Network paths from the cluster to PostgreSQL, the selected LDAP or OIDC identity provider, each managed baseboard management controller (BMC) or the Redfish proxy, and any external object storage, Zapp, Alertmanager, or monitoring endpoints you enable.
- Capacity and a backup policy for the external PostgreSQL service. The chart-managed PostgreSQL instance is intended for evaluation only.

2.5.1.1.1.3 Evaluation Quickstart

After you select DPS_RELEASE, install DPS Core and an Agent for evaluation or testing:

```
helm upgrade --install dps ngc/dps \
  --namespace dps \
  --create-namespace \
  --version "${DPS_RELEASE}" \
  --set agent.enabled=true \
  --wait \
  --timeout 15m

kubectl rollout status statefulset/dps-server --namespace dps
kubectl rollout status deployment/dps-server-agent --namespace dps
```

Note

The quickstart configuration is intended for evaluation only. It uses chart-managed dependencies, permissive authentication, default hostnames, and plaintext gRPC and HTTP connections. TLS is not enabled for the API or UI by default. Complete the security preparation before connecting DPS to a management or power-control network.

2.5.1.1.1.4 Inspect the Versioned Configuration

The selected chart's `values.yaml` is authoritative for configuration keys and defaults. The chart metadata is authoritative for dependency versions. Save a versioned copy for review rather than copying values from another release:

```
helm show values ngc/dps \
  --version "${DPS_RELEASE}" > dps-values-reference.yaml
```

Store deployment overrides in a separate values file, and keep Secret values out of it.

2.5.1.1.1.5 Prepare a Secure Deployment

2.5.1.1.1.6 Select Authentication and Transport Security

The chart defaults to plaintext gRPC with `dps.serverTLS.transportSecurity: insecure`, and it does not enable HTTPS for the HTTP API or UI. Supply TLS through your service mesh or other platform infrastructure, or terminate gRPC TLS directly on the DPS server. Authentication configuration does not enable encryption by itself.

For deployments beyond evaluation:

- To terminate gRPC TLS on the server, configure `dps.serverTLS.transportSecurity` as `tls` or mutual TLS (`mtls`) and provide `dps.serverTLS.existingSecret`. Configure matching agent transport settings. For external authentication and authorization, configure TLS at the proxy or surrounding platform. Deploy the proxy as a sidecar in the server Pod.
- Configure HTTPS for both API and UI ingress. Browser authentication requires HTTPS. If an upstream load balancer terminates TLS, document and protect the remaining network path.
- Choose LDAP, OIDC, or external authentication and authorization by following *Choose an Authentication Mode*. That guide explains server configuration. For UI and `dpsctl` sign-in, refer to *Sign In to DPS*. Use *Configure Service-Specific mTLS* when machine clients need certificate access to specific gRPC services.
- For LDAP-issued DPS tokens, provide a persistent RSA private key through `dps.auth.jwt.existingSecret`.
- Restrict `dps.cors.allowedOrigins` to the UI and other approved browser origins. The default wildcard is not appropriate when credentials are allowed.
- Keep certificate verification enabled for PostgreSQL, LDAP or OIDC, BMCs, proxies, and optional integrations. Mount private CA certificates when the container trust store does not contain the issuing CA.
- Use least-privilege identity groups and separate user and service accounts. Refer to *User Accounts* and *Service Accounts*.

2.5.1.1.1.7 Create Secrets Outside Helm

Create the namespace before external Secret controllers reconcile their objects:

```
kubectl create namespace dps
```

Use your approved secret manager, External Secrets operator, sealed-secret workflow, or another auditable process. Create the applicable database, transport, authentication, and BMC Secrets by following *Credentials and Secrets Configuration*. Do not place passwords, tokens, or private keys on command lines, in shell history, in a Helm release's values, or in a Git repository.

2.5.1.1.1.8 Prepare PostgreSQL and Object Storage

For deployments beyond evaluation, set `postgresql.enabled: false` and point `global.postgresql` at an externally managed PostgreSQL service. Pre-create the database named by `global.postgresql.auth.database` and grant the configured role ownership or the required schema and migration privileges.

PostgreSQL 17 is the minimum supported version when Power Steering is enabled; it is not an open-ended compatibility commitment for every later major release. Operate high availability, backup, and recovery outside the chart, and validate the selected external PostgreSQL version with the pinned DPS chart before a production upgrade.

Power Steering has additional database, object-storage, and network requirements. If you enable it, follow *Deployment and Configuration*.

2.5.1.1.1.9 Create a Values Overlay

The following overlay is a security-focused starting point for an NGINX Ingress, external PostgreSQL, LDAP, and direct BMC connections. Replace every example hostname, Secret, storage class, and identity value. Complete the credential procedures in *Credentials and Secrets Configuration*. If PostgreSQL uses TLS, add the values described in *Database TLS*.

Example secure values overlay

```yaml
global:
  defaultStorageClass: "<storage-class>"
  postgresql:
    host: "postgres.example.com"
    service:
      ports:
        postgresql: "5432"
    auth:
      username: "dps"
      database: "dps"
      existingSecret: "dps-postgresql"

postgresql:
  enabled: false

dps:
  ingress:
    className: "nginx"
    hostname: "api.dps.example.com"
    tls:
      - hosts:
          - "api.dps.example.com"
        secretName: "dps-api-ingress-tls"
  cors:
    allowedOrigins: "https://ui.dps.example.com"
  serverTLS:
    transportSecurity: "tls"
    existingSecret: "dps-server-tls"
  auth:
    authenticatorType: "ldap"
    auth_required_groups: "<dps-admin-group>,<dps-automation-group>"
    jwt:
      existingSecret: "dps-jwt-keys"
  ldap:
    enabled: true
    serverUrl: "ldaps://ldap.example.com:636"
    bindDn: "<bind-dn>"
    baseDn: "<base-dn>"
    defaultRole: ""
    groupRoleMapping: "<dps-admin-group>=admin,<dps-automation-group>
↪=automation"
```

(continues on next page)

(continued from previous page)

```
    bindPassword:
      existingSecret: "dps-ldap-bind"
      existingSecretKey: "bind-password"
  bmcTLS:
    caCertPath: "/secrets/bmc/ca.crt"
  redfish:
    tls:
      insecureSkipVerify: false
  secrets:
    - name: "dps-bmc-ca"
      secretKey: "ca.crt"
      mountPath: "/secrets/bmc/ca.crt"
  prs:
    enabled: false

ui:
  ingress:
    className: "nginx"
    hostname: "ui.dps.example.com"
    tls:
      - hosts:
          - "ui.dps.example.com"
        secretName: "dps-ui-ingress-tls"

power-steering:
  enabled: false
```

Complete the LDAP setup in *Configure LDAP Mode* and the Secret configuration in *LDAP Credentials*.

For OIDC, replace the LDAP authentication settings with the overlay and client setup in *Configure OIDC Mode*. For a deployment-managed authentication proxy, follow *Configure External Authentication and Authorization*.

When the gRPC Ingress class is `nginx`, the chart selects a GRPCS backend for server TLS. For another controller, configure its supported secure gRPC backend or TLS-passthrough mode. TLS passthrough is required when public `dpsctl` clients authenticate with mTLS; refer to *Configure Service-Specific mTLS* for certificate and ingress requirements.

2.5.1.1.1.10 Configure DPS Core and an Agent

DPS Core runs in the `dps-server` workload and routes device operations to DPS Agents. An Agent initiates the connection to Core and performs Redfish operations for its assigned devices, so it needs network access to those BMCs or the configured Redfish proxy.

Deploy Core and its Agent in the same Helm release for each data hall. Core is enabled by default; enabling the Agent creates one logical Agent. Add the following values to the secure overlay before you install the release:

```
dps:
  agent:
    allowHandshakeFallback: false
  grpcServiceMTLSCN:
    nvidia.dcpower.v1.DPSAgentService:
```

(continues on next page)

(continued from previous page)

```
      - "data-hall-a-agent"

agent:
  enabled: true
  replicas: 1
  agentID: "data-hall-a-agent"
  tls:
    transportSecurity: "mtls"
    existingSecret: "dps-agent-data-hall-a-tls"
```

Create the referenced Agent Secret with `ca.crt`, `tls.crt`, and `tls.key`. When `agent.serverAddr` is empty, the Agent connects to the Core service in the same release. Use a distinct Agent ID for each data hall. The `agent.agentID` must match the topology `AgentId` that routes the Agent's assigned devices. The Agent certificate common name must appear in the `dps.grpcServiceMTLSCN` allowlist for `nvidia.dcpower.v1.DPSAgentService`; it does not need to match the Agent ID. Set that `AgentId` on each managed device or an appropriate parent entity; child devices inherit it. The Agent must reach the assigned BMCs or Redfish proxy and their credential and CA Secrets before you import the topology. By default, the Agent inherits `dps.credentialsStore`; set `agent.credentialsStore` only when it requires a different store. Do not scale multiple Agent replicas with the same `agentID`; deploy each Agent independently for the devices it manages. Refer to *Managing Topologies* for the topology workflow.

2.5.1.1.1.11 Review the Rendered Resources

Render the chart before changing the cluster. Treat the rendered file as sensitive because it can contain Secret references and deployment metadata.

```
helm template dps ngc/dps \
  --namespace dps \
  --version "${DPS_RELEASE}" \
  --values values.yaml > dps-rendered.yaml
```

Review the following settings:

- Image tags, dependency versions, resource requests, and persistent volume claims.
- Ingress or Gateway hosts, TLS and Secret references, RBAC, security contexts, and NetworkPolicies.
- External endpoints and enabled optional components.

2.5.1.1.1.12 Install DPS

Install the pinned chart and wait for Kubernetes readiness:

```
helm upgrade --install dps ngc/dps \
  --namespace dps \
  --create-namespace \
  --version "${DPS_RELEASE}" \
  --values values.yaml \
  --wait \
  --timeout 15m
```

2.5.1.1.1.13 Verify the Installation

First verify the Helm release and Kubernetes resources:

```
helm status dps --namespace dps
kubectl get pods,services,ingresses,persistentvolumeclaims --namespace dps
```

All required workloads must be ready, and PVCs must be bound. When you expose the API or UI, confirm that its DNS name resolves to the intended endpoint. If you use Gateway API, inspect the Gateway and Route status instead of Ingress resources.

Install the matching client by using *Installing dpsctl*, then configure its endpoint. For a private CA, set the CA path instead of disabling certificate verification:

```
export DPSCTL_HOST="api.dps.example.com"
export DPSCTL_PORT="443"
export DPSCTL_CA_CERT_PATH="/path/to/api-ca.crt"
```

Authenticate through the procedure for your server's setup in *Configure Server Authentication*. Then check the versions and deployment:

```
dpsctl server-version
dpsctl --version
dpsctl verify --dps-server --database
```

Confirm the client and server releases match and each requested component check is healthy. For LDAP or local JWT configuration, also run `dpsctl verify --auth`; this check does not validate OIDC or external proxy authentication. Refer to *Cannot Connect or Log In* for access failures.

For an Ingress deployment, run `dpsctl verify --ui`, which checks the `dps-ui` Ingress. For a Gateway API deployment, inspect the Gateway and Route status and test the UI endpoint directly.

If you enabled the DPS Agent, verify that its workload is ready before you import a topology:

```
kubectl rollout status deployment/dps-server-agent --namespace dps
kubectl get pods --namespace dps --selector app=dps-server-agent
```

If you prepared BMC credentials, continue with a topology. Import it only after validation and site approval. The following commands list the devices, validate the topology file, and import it:

```
dpsctl device list
dpsctl topology validate "<topology-file>"
dpsctl topology import "<topology-file>"
```

On a new installation, import does not route the topology's nodes to an Agent, so the connection and BMC health checks cannot reach them yet. Topology activation creates that routing and applies each managed node's power policy. Activate the topology through *Managing Topologies* only after your site approves power changes. After the topology appears in `dpsctl topology list --active`, run the read-only connection and health checks:

```
dpsctl check connection --topology "<topology-name>"
dpsctl verify bmc-health start \
  --topology "<topology-name>" \
  --skip-writes \
  --wait
```

These checks exercise the route from DPS Core through the assigned Agent to each BMC. Review the report before rerunning a health check with write probes.

2.5.1.1.1.14 Continue Configuration

Installation creates the control plane; it does not define your facility power model or authorize control of hardware. Continue with *Deploying DPS* to model and import the topology and configure power policies.

For a first managed pilot with explicit safety gates, use *Run the MaxLPS Power Pilot*.

2.5.1.1.1.15 Configure Optional Integrations

Configure only the integrations required for your deployment.

2.5.1.1.1.16 Configure Zapp Telemetry

> **Scope:** This section covers DPS integration and the Zapp timing settings that affect DPS behavior. It assumes that Zapp is already deployed and operational. If you have not deployed Zapp, follow the Zapp deployment instructions before configuring this integration.

The current chart selects `bmc` or `zapp` for public metrics through `dps.telemetry.metrics.source`. Excursion mitigation always consumes the Zapp aggregator stream when a strategy is enabled.

For Zapp-backed metrics or excursion mitigation, configure `zapp.aggregatorService.url`, select the matching TLS mode, and provide the CA or mTLS Secret required by that mode. Do not use a `grpc://` endpoint or `insecureSkipVerify` on an untrusted network.

```yaml
zapp:
  aggregatorService:
    url: "grpcs://zapp-aggregator-client.zapp.svc.cluster.local:9999"
    tls:
      transportSecurity: "mtls"
      existingSecret: "dps-zapp-aggregator-mtls"
dps:
  telemetry:
    metrics:
      source: zapp
```

The `dps-zapp-aggregator-mtls` Secret is a placeholder name. Provide a Kubernetes Secret that contains the certificate authority certificate, client certificate, and client key required for mTLS.

If DPS is not configured to consume Zapp telemetry, topology management, resource group policies, and power allocation continue to function. Features that require live Zapp-backed power telemetry, such as excursion detection and mitigation, do not receive telemetry readings.

2.5.1.1.1.17 Zapp Telemetry Polling Configuration

Zapp controls how often DPS receives updated telemetry. The packaged Zapp sensor catalog uses these polling intervals:

Zapp sensor poll	Default	DPS data affected
`gpuPower`	`1s`	GPU power and requested or set GPU power limits
`platform`	`30s`	Chassis power and other component power, energy, temperature, and power limit readings

For all supported Helm values and sensor configuration options, refer to the Zapp configuration reference.

To change these defaults, copy the complete Zapp `known-sensors.yaml` catalog into a ConfigMap and update each `sensor_polls[].interval`. The ConfigMap must contain the `known-sensors.yaml` key. Set `zapp.knownSensors.existingConfigMap` in the Zapp Helm values to the ConfigMap name. The `sensor_polls` list must contain exactly the `gpuPower` and `platform` polls, and each interval must be positive. Configure either `known_sensors_file` or inline `sensor_polls`, not both.

DPS retains the latest value for each metric. A value remains fresh for the longer of `15s` or three times the polling interval advertised in the Zapp sensor manifest. With the default intervals, `gpuPower` readings expire after `15s`, and `platform` readings expire after `90s` when Zapp does not provide an update. Public metrics responses omit expired or unavailable readings instead of reporting them as zero. A Zapp stream disconnect invalidates retained readings immediately.

Zapp polling intervals control how often Zapp collects readings from Redfish. They do not control how often DPS or the DPS Power Steering Controller requests metrics. The `gpuPower` interval refreshes the per-GPU readings that are available for metrics queries and Power Steering. The `platform` interval refreshes the chassis power readings that DPS uses for excursion detection and mitigation. The `dps.prs.schedulerIntervalSeconds` setting independently controls Power Steering control cycles. The `interval` entry in `dps.excursionMitigation.params` independently controls excursion mitigation checks. When either consumer runs more frequently than the corresponding Zapp poll, consecutive requests can use the same reading.

Shorter polling intervals make new readings available sooner and improve excursion detection and verification response time. They also increase traffic to the Redfish endpoints or proxy. Longer intervals reduce that traffic but delay updates and extend the time that DPS considers the last reading fresh.

2.5.1.1.1.18 DPS Agent Zapp Capability

Enable the DPS Agent Zapp capability when DSX hosts must register with Zapp and stream telemetry through Zapp services:

```yaml
agent:
  enabled: true
  capabilities:
    zapp:
      enabled: true
      metricsManagementService:
        url: "grpcs://zapp-metrics.ns.example:9090"
        tls:
          transportSecurity: "mtls"
          existingSecret: "dps-agent-zapp-metrics-mtls"
      zappService:
        url: "grpcs://zapp-service.ns.example:50051"
        tls:
          transportSecurity: "mtls"
          existingSecret: "dps-agent-zapp-service-mtls"
```

Use `metricsManagementService` for DSX host registration and metrics management. Use `zappService` for sensor manifests and telemetry subscription.

The URL scheme and TLS Secret rules are the same as the server aggregator connection. The Secret names are placeholders. For `transportSecurity: "tls"`, provide `ca.crt`. For `transportSecurity: "mtls"`, provide `ca.crt`, `tls.crt`, and `tls.key`. For Zapp installation and service endpoint configuration, refer to the Zapp deployment and configuration guide.

2.5.1.1.1.19 Prometheus Metrics

`dps-server` exposes Prometheus metrics at `/metrics` through the dedicated `<release>-server-metrics` Service on port `9090`, separate from the HTTP API Service. The chart ships a `ServiceMonitor` template (disabled by default) for in-cluster Prometheus, and the metrics endpoint can also be scraped from outside the cluster. Refer to *Prometheus Monitoring* for the endpoint, authentication, and both deployment setups.

2.5.1.1.1.20 Configure Alertmanager

Configure an HTTPS Alertmanager endpoint whose certificate chain is trusted by the `dps-server` container, and keep certificate verification enabled.

```
dps:
  monitoring:
    enabled: true
    alertmanager:
      url: "https://alertmanager.example.com"
      timeout: "5s"
      tls:
        insecureSkipVerify: false
```

2.5.1.1.1.21 Configure Excursion Mitigation

After configuring Zapp and Alertmanager, follow *Excursion Mitigation* to validate the remaining prerequisites and select a strategy.

```
dps:
  excursionMitigation:
    strategy: "simple"
    params: ""
```

2.5.1.1.1.22 Configure Nautobot

Enable the Nautobot integration only after provisioning a read-capable token through an existing Secret. Do not put the token in `values.yaml`.

```
dps:
  nautobot:
    enabled: true
    url: "https://nautobot.example.com"
    tls:
      insecureSkipVerify: false
    tokenSecret:
      existingSecret: "dps-nautobot-token"
      key: "token"
```

2.5.1.1.1.23 Upgrade, Roll Back, or Uninstall

2.5.1.1.1.24 Upgrade

1. Review the target release notes and compare its chart defaults with the version you currently run.

2. Capture the current release metadata and effective values. Protect the output according to your Secret-handling policy because an earlier install might have stored sensitive values in the Helm release:

```
helm status dps --namespace dps
helm get values dps --namespace dps --all > pre-upgrade-values.yaml
```

3. Back up external PostgreSQL and any external Power Steering object storage. Confirm that restore procedures have been tested.
4. Update the overlay for renamed, removed, or newly required values. Render and review the target chart.
5. Upgrade to an explicit version:

```
helm upgrade dps ngc/dps \
  --namespace dps \
  --version "<target-release-version>" \
  --values values.yaml \
  --wait \
  --timeout 15m
```

6. Repeat the Kubernetes, version, component, and BMC verification gates.

2.5.1.1.1.25 Roll Back

Use rollback only when the target release notes confirm compatibility with the current database and persistent data:

```
helm history dps --namespace dps
helm rollback dps "<revision>" --namespace dps --wait --timeout 15m
```

Helm rollback restores Kubernetes manifests. It does not restore an external database, object storage, or their schemas. If the failed upgrade changed persistent data incompatibly, use the release-specific recovery procedure and your tested backups instead of assuming that Helm rollback is sufficient.

2.5.1.1.1.26 Uninstall

Capture required evidence and backups before uninstalling:

```
helm uninstall dps --namespace dps
```

The command removes Helm-managed resources. Externally managed Secrets, databases, object storage, and some retained persistent volumes can remain. Review them individually under your retention policy. Delete the namespace or persistent data only through an approved, recoverable decommission procedure.

2.5.1.1.1.27 Troubleshooting

Preserve the rendered chart, Helm status, events, component health output, and a short time-bounded log sample before changing values. Redact passwords, tokens, private keys, BMC credentials, and unnecessary workload identifiers. Use *Deployment Component Unhealthy* for installation failures and *Cannot Connect or Log In* for login failures. For BMC connectivity or health failures, use *BMC or Redfish Calls Fail*. For database certificate errors, use *Troubleshooting*.

2.5.1.1.2 System Requirements

Use this guide to choose the CPU, memory, and storage resources for running Dynamic Power Software (DPS) on Kubernetes. It is for cluster administrators, and its values are recommended starting points.

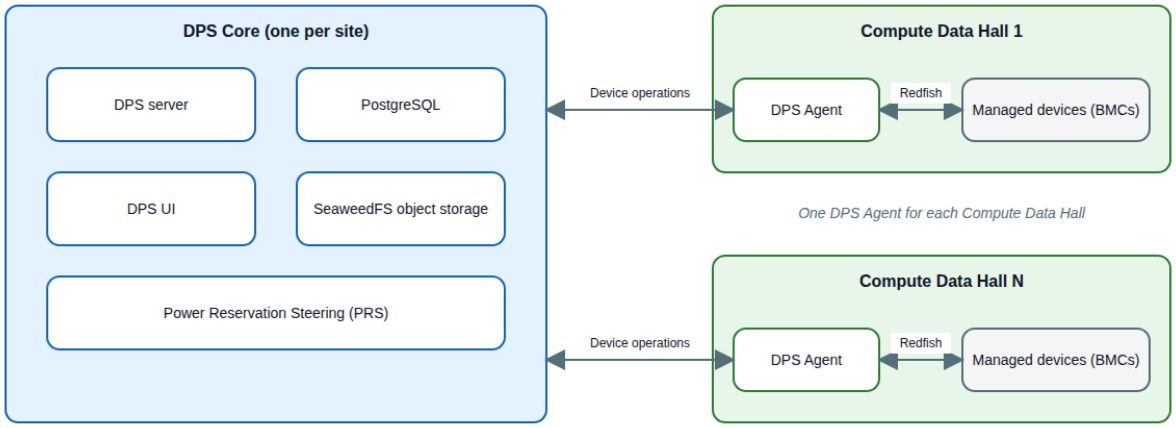
Actual usage depends on your inventory, how often you collect telemetry, and your hardware. Monitor usage after you deploy, and adjust the resources as needed. Refer to *Validate and Monitor*.

2.5.1.1.2.1 How DPS Is Deployed

DPS has two parts:

- **DPS Core:** one deployment per site. It includes the DPS server, the DPS UI, Power Steering, PostgreSQL, and SeaweedFS object storage.
- **DPS Agent:** one deployment for each Compute Data Hall, the facility area that contains your compute scalable units (SUs). It collects telemetry from the devices in its hall and applies the power limits that DPS sets, through the devices' baseboard management controllers (BMCs). Run one replica for each DPS Agent.

DPS Core and the DPS Agents can run in the same Kubernetes cluster or in separate clusters. The resource values are the same in both cases. The values in this guide assume Power Steering is enabled, which is the default.



2.5.1.1.2.2 Before You Begin

Before you size your cluster, confirm that your environment meets the Kubernetes, Helm, PostgreSQL, storage, and network requirements in *Deployment and Configuration Guide*.

2.5.1.1.2.3 Size Your Deployment

A Scalable Unit (SU) is the smallest repeatable IT building block in the DSX facilities architecture. One SU contains 14 NVIDIA Vera Rubin NVL72 racks. For details, refer to the DSX Facilities Infrastructure Reference Design.

To choose the resources for your deployment:

1. Determine how many SUs DPS manages across your site, and how many SUs each DPS Agent manages.

2. Select the DPS Core row that matches your site in *Reserve Cluster Capacity*. If your number of SUs falls between two rows, select the larger row.
3. Select the row that matches each DPS Agent in the same section, using the same rule.
4. Reserve that capacity in your cluster, and add capacity for other workloads. Refer to *Additional Capacity*.
5. Configure each container with the values in *Configure Container Resources*.
6. Verify the cluster. Refer to *Validate and Monitor*.

2.5.1.1.2.4 Reserve Cluster Capacity

Size the cluster by allocatable capacity, not node capacity. Allocatable capacity is the CPU and memory that Kubernetes can give to pods, after it reserves resources for the operating system and system daemons.

CPU values are Kubernetes CPU units. One unit is one CPU core, and `250m` is a quarter of a unit. The capacity values include headroom. If you reserve the listed capacity, the combined requests of the containers use no more than 80 percent of it.

The following table lists the total requests of the DPS Core containers and the capacity to reserve. The capacity is the total requests divided by 0.8. The container requests are listed in *Configure Container Resources*.

Total SUs	Total CPU Requests	Total Memory Requests	CPU Units to Reserve	Memory to Reserve	Persistent Storage
1 SU	3.15	4,544 MiB	4	6 GiB	30 GiB
3 SUs	3.35	6,336 MiB	5	8 GiB	30 GiB
4 SUs	3.35	7,104 MiB	5	9 GiB	30 GiB
9 SUs	3.65	10,688 MiB	5	14 GiB	30 GiB

The following table lists the total requests and the capacity to reserve for each DPS Agent (`dps-server-agent-<suffix>`). If DPS Core and an Agent run in separate clusters, reserve this capacity in the Agent's cluster, in addition to that cluster's other workloads:

SUs per DPS Agent	CPU Requests	Memory Requests	CPU Units to Reserve	Memory to Reserve
1 SU	0.3	384 MiB	1	1 GiB
3 SUs	0.8	768 MiB	1	1 GiB
4 SUs	1	896 MiB	2	2 GiB

The DPS Agent table covers up to 4 SUs per Agent. For an Agent that manages more SUs, start with the 4-SU row and add capacity for each additional SU as described in *Scaling*.

In a single-cluster deployment, add the DPS Core row from the first table and one DPS Agent row from the second table for each Agent. These values cover DPS only.

The following examples show the calculation for a single cluster:

- **3 SUs and one DPS Agent:** DPS Core needs 5 CPU units, 8 GiB, and 30 GiB. The Agent adds 1 CPU unit and 1 GiB. The total is 6 CPU units, 9 GiB, and 30 GiB.
- **9 SUs and three DPS Agents that manage 3 SUs each:** DPS Core needs 5 CPU units, 14 GiB, and 30 GiB. The three Agents add 3 CPU units and 3 GiB. The total is 8 CPU units, 17 GiB, and 30 GiB.

2.5.1.1.2.5 Use an External PostgreSQL Service

The values in this guide include the PostgreSQL service that DPS deploys. If you use an externally managed PostgreSQL service, reduce the persistent storage for DPS Core by 10 GiB. Keep the CPU and memory values. The bundled PostgreSQL service accounts for at most 0.25 CPU units and 0.375 GiB of requests.

2.5.1.1.2.6 Configure Container Resources

The capacity tables in *Reserve Cluster Capacity* tell you how much capacity to reserve in the cluster. The tables in this section tell you how to configure each container.

Set the requests of each container to at least the values in the tables. Set the memory limit of each container to the listed value.

CPU limits are optional. If you set a CPU limit on a container, set it to at least twice the CPU request of that container. For example, the DPS server requests 2 CPU units, so its CPU limit must be at least 4 CPU units.

Do not set a CPU limit lower than 1 CPU unit on the OpenTelemetry sidecars in the Power Steering controller and configuration server pods. A sidecar uses CPU in short bursts. A limit of twice its request (100m) throttles it.

Kubernetes rejects a pod whose request exceeds its limit.

> **Note**
>
> A limit is a ceiling, not a reservation. The memory limits can add up to more than the allocatable capacity in the capacity tables. Plan node memory for the requests, and watch for memory pressure on the node.

In the tables, `Gi` and `Mi` are the Kubernetes notation for GiB and MiB.

2.5.1.1.2.7 DPS Core

DPS Core runs the following pods. Their requests add up to the total requests in the DPS Core capacity table in *Reserve Cluster Capacity*. A pod that a Deployment manages has a generated suffix after the name.

Component	Pod Name	Resources Depend on SUs
DPS server	`dps-server-0`	Yes
Power Steering controller, with its OpenTelemetry sidecar	`dps-power-controller-<`	Yes
PostgreSQL	`dps-postgresql-0`	Yes
DPS UI	`dps-ui-<suffix>`	No
Power Steering configuration server, with its OpenTelemetry sidecar	`dps-power-config-<suf1`	No
SeaweedFS master	`dps-seaweedfs-master-(`	No
SeaweedFS volume server	`dps-seaweedfs-volume-(`	No
SeaweedFS filer	`dps-seaweedfs-filer-0`	No
SeaweedFS S3 gateway	`dps-seaweedfs-s3-<suf1`	No

The following tables list the requests and limits for the three pods whose resources depend on the total number of SUs.

The following table lists the DPS server (`dps-server-0`) resources:

Setting	1 SU	3 SUs	4 SUs	9 SUs
CPU request	2	2	2	2
Memory request	1Gi	2688Mi	3456Mi	6784Mi
Memory limit	10Gi	10Gi	10Gi	10Gi

The following table lists the Power Steering controller (`dps-power-controller-<suffix>`) resources:

Setting	1 SU	3 SUs	4 SUs	9 SUs
CPU request	250m	450m	450m	750m
Memory request	1Gi	1Gi	1Gi	1280Mi
Memory limit	2Gi	2Gi	2Gi	2Gi

The following table lists the PostgreSQL (`dps-postgresql-0`) resources:

Setting	1 SU	3 SUs	4 SUs	9 SUs
CPU request	250m	250m	250m	250m
Memory request	256Mi	384Mi	384Mi	384Mi
Memory limit	2Gi	2Gi	2Gi	2Gi

The following table lists the remaining pods, and the OpenTelemetry sidecar in two of them, whose resources are the same for every number of SUs:

Component	Pod Name	CPU Request	Memory Request	Memory Limit
DPS UI	`dps-ui-<suffix>`	`250m`	`384Mi`	`1Gi`
Power Steering configuration server	`dps-power-config-<suffix>`	`100m`	`384Mi`	`512Mi`
Power Steering OpenTelemetry sidecar (in each of the controller and configuration server pods)	`dps-power-controller-<suf` and `dps-power-config-<suffix>`	`50m`	`128Mi`	`1Gi`
SeaweedFS master	`dps-seaweedfs-master-0`	`50m`	`64Mi`	`64Mi`
SeaweedFS volume server	`dps-seaweedfs-volume-0`	`50m`	`256Mi`	`256Mi`
SeaweedFS filer	`dps-seaweedfs-filer-0`	`50m`	`768Mi`	`1Gi`
SeaweedFS S3 gateway	`dps-seaweedfs-s3-<suffix>`	`50m`	`128Mi`	`256Mi`

2.5.1.1.2.8 DPS Agent

The following table lists the requests for each DPS Agent (`dps-server-agent-<suffix>`):

SUs per DPS Agent	CPU Request	Memory Request
1 SU	`300m`	`384Mi`
3 SUs	`800m`	`768Mi`
4 SUs	`1`	`896Mi`

Set the DPS Agent memory limit to at least twice the memory request, or leave it unset.

2.5.1.1.2.9 Storage

Keep at least 20 percent of the persistent volume capacity free. The following table lists the persistent volumes for DPS Core:

Volume	Size
PostgreSQL data	8 GiB
SeaweedFS volume server	10 GiB
SeaweedFS filer	1 GiB
SeaweedFS master	1 GiB
DPS server data	1 GiB
Total	**21 GiB**

The persistent volumes in the preceding table total 21 GiB. This is the sum of the PostgreSQL, SeaweedFS, and DPS server data volumes. Provision at least 30 GiB of storage capacity for DPS Core, which leaves room to grow beyond that total. A DPS Agent needs no persistent volume.

Ephemeral storage is separate from persistent storage. The following table lists the ephemeral-storage requests and limits:

Component	Request	Limit
DPS UI	`100Mi`	`100Mi`
DPS server	`1Gi`	`10Gi`
PostgreSQL	`500Mi`	`2Gi`

The capacities in this section do not include database backups, PostgreSQL write-ahead log (WAL) archives, replicas, snapshots, container images, logs, or monitoring data. Increase the volume sizes to match your data-retention requirements.

2.5.1.1.2.10 Data That Grows Over Time

- **Power Steering dump (SeaweedFS volume server).** Power Steering writes a record of each cycle to SeaweedFS. DPS deletes it after the retention period, set by `power-steering.global.telemetry.dump.retention` (default 5h). The dump needs about 80 MB per SU per hour. Size the SeaweedFS volume server for: SUs × 80 MB × retention in hours, plus 25 percent. With the default retention, 9 SUs need about 4.5 GB. With 24-hour retention, 9 SUs need about 22 GB, so increase the SeaweedFS volume above its default 10 GiB.
- **PostgreSQL.** PostgreSQL holds the configuration and state of DPS and Power Steering. Some tables keep history, and DPS does not delete it automatically. The write-ahead log uses up to about 1 GB with the default PostgreSQL settings.
- **Monitoring.** Monitor the used space of each volume. Expand a volume before it is 80 percent full. Use a StorageClass that allows volume expansion.

2.5.1.1.2.11 Additional Capacity

Reserve additional CPU, memory, and storage for:

- Multiple replicas or high availability.
- PostgreSQL or SeaweedFS replication.
- Backup or disaster recovery services.
- Ingress controllers, monitoring, log collection, service meshes, or security agents.
- Other workloads on the same nodes.

2.5.1.1.2.12 Validate and Monitor

Confirm that the cluster offers the required allocatable capacity:

```
kubectl describe node <node-name> | grep -A 8 "Allocatable"
```

The allocatable CPU and memory must be at least the values in the capacity tables. For a cluster with more than one node, add the allocatable CPU and memory of all schedulable nodes. At least one node must also have enough allocatable capacity for the largest pod, the DPS server (`dps-server-0`).

After you install DPS, confirm that every DPS pod is scheduled and that its requests match the values in *Configure Container Resources*:

```
kubectl get pods -n <namespace> -o wide
kubectl describe pod <pod-name> -n <namespace> | grep -A 6 "Requests"
```

Monitor usage during normal operation:

- Compare actual usage with requests by using `kubectl top pod -n <namespace> --containers`. This command requires a Metrics API implementation, such as metrics-server, in the cluster.
- Check for restarts caused by out-of-memory kills. Use `kubectl get pods -n <namespace>` to find pods with restarts, then use `kubectl describe pod <pod-name> -n <namespace>` and look for a last state of `OOMKilled`.
- Check for CPU throttling with the `container_cpu_cfs_throttled_seconds_total` metric in your monitoring system.

To scrape DPS server metrics with Prometheus, refer to *Prometheus Monitoring*.

Increase a container's memory limit when its usage exceeds 80 percent of the limit, or when it restarts with `OOMKilled`. Increase CPU requests or limits when throttling occurs repeatedly during normal operation. Watch DPS server memory first, because its usage grows with the number of SUs.

2.5.1.1.2.13 Scaling

- To add SUs to a site, move DPS Core to the next row in the Core capacity table. Each additional SU adds about 1 GiB of allocatable memory. For each DPS Agent, move to the next row in the Agent capacity table if it manages more SUs. Each additional SU adds about 0.3 CPU units and 0.2 GiB of allocatable capacity to that DPS Agent.
- To add a Compute Data Hall, deploy one more DPS Agent, and review the DPS Core row for the new total number of SUs.
- To add a site, deploy one more DPS Core and one DPS Agent for each Compute Data Hall.

2.5.1.1.2.14 Related Topics

- *Deployment and Configuration Guide*
- *Redfish Proxy Configuration*
- *Prometheus Monitoring*
- *Troubleshooting*

2.5.1.1.3 Configure Server Authentication

DPS supports Lightweight Directory Access Protocol (LDAP), OpenID Connect (OIDC), service-specific mutual TLS (mTLS), and external authentication and authorization. For evaluation, you can use permissive authentication. Choose the authentication mode before deployment.

This guide is for administrators configuring the server and its UI integration. For client sign-in procedures, refer to *Sign In to DPS*. For applications and scripts, refer to *Service Accounts*.

Authentication verifies an identity. Authorization determines whether that identity can access the requested API. Transport Layer Security (TLS) encrypts the connection and verifies the server identity. Configure both access controls and transport security.

2.5.1.1.3.1 Choose an Authentication Mode

Use the following table to select the server's authentication mode and prepare the required identity infrastructure.

Authentication mode	Server behavior	Requirements
LDAP	Validates directory credentials and issues DPS tokens.	LDAP directory, bind credentials, access groups, and a persistent signing key.
OIDC	Validates tokens issued by an identity provider.	Issuer, signing keys, API audience, scopes, and registered clients.
Service-specific mTLS	Accepts trusted client certificates for permitted gRPC services.	Server and client certificates, service common name (CN) allowlists, and a TLS path that preserves client identity.
External authentication and authorization	Delegates caller authentication and API access decisions to an external proxy service.	Proxy service, access policies, verified identity headers, and request routing.
Permissive evaluation authentication	Accepts arbitrary username/password credentials and issues tokens.	An isolated evaluation environment.

LDAP and built-in OIDC validation are alternative server modes. Service-specific certificate authentication can accompany either mode. External authentication and authorization delegate both checks to the external integration. This mode controls caller access to DPS; it does not configure credentials for an outbound proxy connection.

The UI uses the HTTP API; `dpsctl` uses gRPC. A certificate accepted for a gRPC service does not provide a browser sign-in method.

2.5.1.1.3.2 Configure LDAP Mode

In LDAP mode, the server checks usernames and passwords against your LDAP directory. It then issues access and refresh tokens for the UI and `dpsctl`. You need a directory bind account, a search base covering your users and groups, and a persistent RSA signing key.

Use the following overlay with the bind and certificate Secret configuration in *LDAP Credentials*.

LDAP Authentication Overlay

```
dps:
  auth:
    authenticatorType: ldap
    auth_required_groups: "dcpower-users,automation-users"
    jwt:
      existingSecret: dps-jwt-keys
```

(continues on next page)

(continued from previous page)

```
  ldap:
    enabled: true
    serverUrl: "ldaps://ldap.example.com:636"
    bindDn: "<bind-dn>"
    baseDn: "<base-dn>"
    userFilter: "(&(objectClass=person)(uid={username}))"
    groupProfile: groupOfNames
    groupAttribute: cn
    defaultRole: ""
    bindPassword:
      existingSecret: dps-ldap-bind
      existingSecretKey: bind-password
ui:
  auth:
    provider: dps
```

Replace the example group names with groups from your directory. For protected API requests, an identity must belong to exactly one group in `dps.auth.auth_required_groups`. Membership in zero or multiple listed groups fails authorization, even if username/password login succeeds.

Choose the group resolver that matches your directory.

`dps.ldap.groupProfile`	Group membership lookup
Empty or `custom`	Uses your configured `groupFilter` and `groupAttribute`.
`groupOfNames`	Uses groups whose `member` attribute contains the user DN.
`rfc2307`	Resolves POSIX secondary groups through `memberUid` and primary groups through `gidNumber`. Ignores `groupFilter`.
`activeDirectory`	Uses Active Directory groups whose `member` attribute contains the user DN.

For `groupOfNames` and `activeDirectory`, a custom `groupFilter` overrides the profile's default filter. Match `userFilter` to your directory's username attribute. Refer to *User Accounts* and *Service Accounts* for account examples.

The default token lifetimes are 15 minutes for access tokens and 24 hours for refresh tokens. Configure them with `dps.auth.jwt.accessTokenTTL` and `dps.auth.jwt.refreshTokenTTL`. For client login and refresh behavior, refer to *dpsctl login*.

2.5.1.1.3.3 Configure OIDC Mode

In OIDC mode, your identity provider issues tokens. The server validates them against the configured issuer, JSON Web Key Set (JWKS), audience, and required scopes. DPS username/password token issuance is disabled.

Register separate identity-provider clients for the UI, interactive `dpsctl`, and automation. Configure the provider to issue tokens for your API audience and grant the API scope only to identities that need access.

Register the UI and interactive `dpsctl` as public clients without a client secret. Neither sends a client secret during token exchange. Register service-account clients separately for the client-credentials grant.

Use the following client registration settings.

Client	Client type	Grant or flow	Client secret
UI	Public	Authorization code with Proof Key for Code Exchange (PKCE).	None.
Interactive `dpsctl`	Public	Device authorization grant.	None.
Applications and scripts	Confidential	OAuth 2.0 client-credentials grant.	Required for the `dpsctl` client-credentials flow.

The following overlay uses scope-based access. It explicitly disables the additional group membership check.

OIDC Authentication Overlay

```
dps:
  ldap:
    enabled: false
  auth:
    auth_required_groups: ""
    oidc:
      enabled: true
      issuer: "https://idp.example.com/realms/dps"
      jwksUrl: "https://idp.example.com/realms/dps/protocol/openid-connect/
↪certs"
      audience: dps-api
      requiredScopes:
        - dps:api
      allowedClientIds:
        - dps-ui
        - dpsctl
        - dps-automation
ui:
  auth:
    provider: oidc
    clientId: dps-ui
    oidc:
      scope: "openid dps:api"
```

Use your provider's issuer and JWKS URLs. The server requires a signed JSON Web Token (JWT) with the expected issuer and audience, a nonempty subject, and a valid expiration. Every configured `requiredScopes` entry must be present. If you set `allowedClientIds`, the token's `azp` claim must match an entry.

To restrict access by groups as well, set `dps.auth.auth_required_groups` and `dps.auth.oidc.groupsClaim`. The group claim defaults to `groups` and supports a dotted path for nested claims. Each identity must match exactly one required group. Group membership does not replace the required API scopes.

For a private certificate authority, mount its CA bundle and set `dps.auth.oidc.caCertFile`. Keep

issuer and JWKS URLs on HTTPS, and keep `dps.auth.oidc.allowHTTP` disabled outside local development.

2.5.1.1.3.4 Configure the UI Client

Configure the UI public OIDC client for the authorization code flow with PKCE. Register `https://ui.dps.example.com/login/oauth` as the redirect URI and `https://ui.dps.example.com/login` as the post-logout URI. Use your actual UI origin. Override these defaults with `ui.auth.redirectUrl` and `ui.auth.oidc.postLogoutRedirectUrl` when needed.

The UI discovers the provider endpoints from `dps.auth.oidc.issuer` and sends the access token with API requests. For UI sign-in, refer to *Sign In to the UI*.

2.5.1.1.3.5 Configure the CLI Clients

Enable the device authorization grant for the public interactive CLI client. For client login and endpoint overrides, refer to *dpsctl login*.

For scripts and services, enable the OAuth 2.0 client-credentials grant on a separate confidential client. For token acquisition and secret handling, refer to *Obtain a Service Account Token*.

Some deployments accept an ID token for API access. Use `ui.auth.oidc.useIdTokenForApiRequests: true` or `dpsctl login --id-token` only when your API or proxy accepts that token's audience and other required claims. Access tokens are the default for both clients.

For protected Prometheus metrics, configure the separate `dps:metrics:read` scope and scrape credentials in *Protect the Metrics Endpoint*.

2.5.1.1.3.6 Configure Service-Specific mTLS

You can allow machine clients to authenticate with mutual TLS (mTLS) to specific gRPC services. Configure `dps.grpcServiceMTLSCN` with each service's fully qualified name and permitted client certificate common names (CNs).

The following overlay allows a machine client to query the server version and manage topologies. Provision its certificate and private key outside the chart.

```yaml
dps:
  grpcServiceMTLSCN:
    nvidia.dcpower.v1.VersionService:
      - dps-operator
    nvidia.dcpower.v1.TopologyManagementService:
      - dps-operator
```

With server transport `tls`, clients can present either a permitted certificate or a valid bearer token. A verified certificate whose CN matches the requested service permits access without a token. Other requests continue through token validation. A certificate listed for one service does not grant access to unlisted services.

Server transport `mtls` requires every client to present a trusted certificate during the TLS handshake, including clients that also use tokens. You still need service CN allowlists for clients that authenticate without a token.

The client connection must reach the server without an ingress replacing its certificate identity. For NGINX or Traefik, set `dps.ingress.tlsPassthrough: true` and configure the controller for TLS passthrough. The server certificate must cover the public API hostname.

For client certificate flags and a connection example, refer to *Mutual TLS*.

For agents, configure the `DPSAgentService` allowlist and agent certificate Secret as described in *Deployment and Configuration Guide*. For human UI access, retain LDAP or OIDC sign-in. A deployment using only machine certificates does not provide DPS username/password login when local token issuance is disabled.

2.5.1.1.3.7 Configure External Authentication and Authorization

Use this mode to delegate caller authentication and API access decisions to an external proxy service. The proxy applies your access requirements and forwards accepted gRPC requests to DPS. It can perform authentication, authorization, or both, using its own logic or services it calls. The external integration must enforce all required checks because built-in gRPC authentication and authorization are disabled in this mode.

Deploy a custom or platform-provided sidecar in the DPS server Pod. Install and configure the sidecar independently of the DPS chart.

2.5.1.1.3.8 Configure Proxy Access Controls

Configure your external integration to meet the following requirements.

1. Apply your authentication and authorization policy to every protected gRPC method, including streaming calls and calls originating from the HTTP gateway.
2. Strip caller-supplied identity headers and set verified identity metadata. Use the header selected by `dps.auth.externalSubjectHeader`, default `x-user-id`, for caller attribution in DPS logs.
3. Configure TLS, identity-provider trust, and access policies in the proxy or surrounding platform. Configure internal clients, including the agent, to supply credentials accepted by that policy.
4. Route all client API traffic through the external checks. Keep the backend listener inaccessible to clients that could bypass those checks.

The verified subject header provides log attribution; it does not trigger built-in group or scope checks. Those access decisions belong to the external integration. DPS server token issuance is also disabled in this mode.

2.5.1.1.3.9 Configure Proxy Routing

The following overlay enables delegation and configures OIDC sign-in for the UI. The issuer supplies UI discovery settings; the proxy validates API tokens.

External Authentication and Authorization Overlay

```yaml
dps:
  auth:
    external:
      enabled: true
      proxyPort: 8081
    externalSubjectHeader: x-user-id
    oidc:
      enabled: false
      issuer: "https://idp.example.com/realms/dps"
  ldap:
    enabled: false
```

(continues on next page)

(continued from previous page)

```
ui:
  auth:
    provider: oidc
    clientId: dps-ui
    oidc:
      scope: "openid dps:api"
```

The chart sends HTTP gateway calls through the sidecar so the UI's API requests pass through the external access checks. Configure the following routing in your deployment.

1. Configure a local plaintext HTTP/2 listener on the port selected by `dps.auth.external.proxyPort`, default `8081`. The HTTP gateway uses this listener on loopback.
2. Forward permitted requests to the DPS loopback listener, default `127.0.0.1:50051`, using plaintext HTTP/2.
3. Redirect the gRPC Service's `targetPort` to the proxy's client-facing listener. Keep the Service port unchanged, default `8080`. If the proxy terminates TLS on a separate listener, route the Service to that TLS port and retain the local plaintext listener for HTTP gateway calls.

With a local plaintext proxy listener and the default ports, the path is `Service :8080 proxy :8081 DPS :50051`.

You can use a custom sidecar or one provided by your platform, as long as it meets these requirements. Configure the container and Service routing through your deployment or admission injector.

Enabling `dps.auth.external.enabled` configures the server integration only. It does not install a proxy or policies, or redirect the Service. Without the proxy integration and routing, clients cannot reach the backend.

Configure UI OIDC client registration as described in *Configure OIDC Mode*, using scopes and token settings accepted by the proxy. Keep `dps.auth.oidc.enabled: false` in this mode; the issuer setting supplies UI discovery only. For client access, refer to *Sign In to DPS* and *Service Accounts*. DPS username/password login requires a compatible external token endpoint; the server itself does not issue tokens in this mode.

2.5.1.1.3.10 Configure Permissive Authentication for Evaluation

The Helm chart defaults to `alwaysAllow` when LDAP is disabled and `dps.auth.authenticatorType` is empty. This accepts arbitrary username/password credentials and issues DPS tokens. It is suitable only for isolated evaluation and testing. Do not use this setting in production.

Explicitly select permissive authentication in an evaluation overlay:

```
dps:
  auth:
    authenticatorType: alwaysAllow
  ldap:
    enabled: false
ui:
  auth:
    provider: dps
```

Keep the chart's default access group `dcpower-users` if you retain group checks; evaluation tokens include that membership.

When you run `dps-server` directly, an unset authenticator requires LDAP to be enabled. To select permissive authentication, explicitly set `--authenticator-type=alwaysAllow`. Configuring TLS does not make this authenticator suitable for production.

2.5.1.1.3.11 Verify the Configuration

Before installation, render your chosen overlay together with the deployment values as described in *Deployment and Configuration Guide*. Check the authentication environment variables, certificate mounts, and UI provider in the rendered resources. For an external proxy, also check the proxy deployment and Service routing in your deployment manifests.

After deployment, complete the following checks:

1. Confirm the server and any required authentication sidecars are ready. For external authentication and authorization, confirm the sidecar's listener ports and Service routing match your configuration.
2. Authenticate an allowed identity by following *Sign In to DPS* or *Service Accounts*.
3. Request the server version to verify authenticated API access. For a token-based client, use the following command with your endpoint and CA.

   ```
   dpsctl --host api.dps.example.com --port 443 \
     --ca-cert-path /path/to/api-ca.crt server-version
   ```

   For certificate clients, use the command in *Mutual TLS*.
4. Open the UI and confirm that an allowed API request succeeds after sign-in.
5. Verify that an identity outside the configured access policy cannot call a protected API. Test authentication and authorization separately; successful sign-in does not prove that access rules are correct.

Use the following checks when connection or access fails:

Symptom	Check
TLS connection fails	Verify the hostname, trusted CA, endpoint port, and any global client-certificate requirement.
LDAP login fails	Verify the directory credentials, bind Secret, search base/filter, and LDAP certificates.
OIDC login fails	Verify the registered client, requested scopes, UI callback URI, or CLI device-grant configuration.
Login succeeds but an API request is denied	Check required groups, scopes, audience, allowed client IDs, or the external proxy policy.
Certificate access fails	Verify the certificate chain, requested service CN allowlist, and TLS passthrough.
External proxy requests cannot connect	Check proxy availability, loopback connectivity, listener ports, and Service routing.

Refer to *Cannot Connect or Log In* for diagnostic commands and error codes. Share redacted errors and configuration; keep passwords, tokens, cookies, and private keys out of logs and support materials.

2.5.1.1.4 Air-Gapped Deployment

2.5.1.1.4.1 Overview

DPS is a standard Kubernetes application. Deploying it in an air-gapped environment requires that two artifacts are available inside the environment:

1. **Container images** – all DPS images must be available in a container registry that the cluster can pull from.
2. **Helm chart** – the DPS Helm chart package (`.tgz`) must be accessible to `helm install`.

This page does not cover general Kubernetes air-gap procedures such as setting up a private registry or mirroring images. Refer to your Kubernetes distribution's documentation for those topics.

2.5.1.1.4.2 Standalone Kubernetes

Use standard Helm workflows to prepare the chart and images on a connected machine, then transfer them to the air-gapped environment:

```
# On a connected machine
helm repo add ngc https://helm.ngc.nvidia.com/nvidia
helm repo update ngc
helm pull ngc/dps                # downloads dps-<version>.tgz
```

Transfer the `.tgz` file and all required container images to the air-gapped environment.

Refer to the *Deployment and Configuration Guide* for the full set of configuration options.

2.5.1.1.4.3 NVIDIA Mission Control / Base Command Manager

If you are deploying DPS through **NVIDIA Mission Control (NMC)** or **Base Command Manager (BCM)**, follow the NMC/BCM air-gap packaging documentation to prepare your environment.

Place the DPS Helm chart archive in the following directory on the head node if it is not already there:

```
/cm/local/apps/dps/var/charts/
```

The `cm-setup` installer will automatically detect the local chart and use it instead of pulling from the NGC repository. No additional configuration is required.

2.5.1.1.5 Installing dpsctl

2.5.1.1.5.1 Overview

This guide provides a step-by-step process for installing the DPS command-line client.

Choose the `dpsctl` release that matches the DPS server and Helm chart selected in *Deployment and Configuration Guide*. Do not rely on a floating container tag or an unpinned package update. There are three install paths:

1. ***Install via Container*** (recommended) - Run `dpsctl` as a container, optionally fronted by a small shell wrapper so it behaves like a native `dpsctl` binary.
2. ***Install Pre-built Binary*** - Download a platform-specific `dpsctl` binary from the NVIDIA Partner Portal.
3. ***Install from Package Repositories*** - Install the `dps-utils` package through the system package manager.

2.5.1.1.5.2 Prerequisites

- Linux or macOS operating system
- amd64 or arm64 architecture
- Docker (or a compatible OCI runtime such as Podman) for the container install path

2.5.1.1.5.3 Verification

After installation, verify that `dpsctl` is working correctly:

```
dpsctl --version

dpsctl --help
```

After the server is installed, authenticate through the method configured for your deployment. Follow *dpsctl login* for LDAP or OIDC user sign-in, or *Service Accounts* for service credentials. Then compare both versions:

```
dpsctl server-version
dpsctl --version
```

If the versions differ, install the client artifact from the server release before using mutating commands.

2.5.1.1.5.4 Troubleshooting

2.5.1.1.5.5 Permission Issues

```
chmod +x dpsctl

sudo chown $USER:$USER dpsctl
```

2.5.1.1.5.6 Connection Issues

Use *Cannot Connect or Log In* for endpoint, TLS, authentication, and authorization failures.

2.5.1.1.5.7 Next Steps

After installing `dpsctl`, refer to the following guides:

- *dpsctl Usage Guide* - Complete command reference
- *Deploy DPS* - Set up DPS server
- *Configure Server Authentication* - Configure server authentication and client access

2.5.1.1.5.8 Install via Container

`dpsctl` is published as a container image at `nvcr.io/nvidia/dpsctl:<release-version>`. Use the tag that matches the DPS server release. This path does not require a separate binary download or platform selection.

2.5.1.1.5.9 Prerequisites

- Linux or macOS operating system
- amd64 or arm64 architecture
- Docker (or a compatible OCI runtime such as Podman)

2.5.1.1.5.10 One-shot `docker run`

Run any `dpsctl` subcommand directly:

```
export DPSCTL_VERSION="<release-version>"

docker run --rm -it --network host \
  --mount type=bind,source=/path/to/api-ca.crt,target=/certs/api-ca.crt,
↪readonly \
  "nvcr.io/nvidia/dpsctl:${DPSCTL_VERSION}" \
  login --host api.dps.example.com --port 443 \
  --ca-cert-path /certs/api-ca.crt
```

Notes:

- `--network host` lets the container reach `localhost` and any cluster DNS names that resolve on your host.

2.5.1.1.5.11 Install as a `dpsctl` Shim

For a smoother experience, install a small wrapper script as `~/.local/bin/dpsctl` so every invocation transparently runs the container with sensible defaults (HOME mount, working-directory mount, TTY detection, env-var pass-through):

```
mkdir -p ~/.local/bin
cat > ~/.local/bin/dpsctl <<'EOF'
#!/bin/bash
# dpsctl wrapper — proxies all calls to the Docker image.
# Set DPSCTL_VERSION to the release that matches the server.
IMAGE="${DPSCTL_IMAGE:-nvcr.io/nvidia/dpsctl}"
VERSION="${DPSCTL_VERSION:?Set DPSCTL_VERSION to the server release}"
mkdir -p "${HOME}/.dpsctl"
# Resolve real path so symlinks inside PWD work inside the container
WORKSPACE=$(realpath "${PWD}" 2>/dev/null || echo "${PWD}")

# Mount only the client state directory and current workspace.
MOUNTS=(-v "${HOME}/.dpsctl:${HOME}/.dpsctl" -v "${WORKSPACE}:${WORKSPACE}")

# Always keep stdin attached; only allocate a TTY when running interactively,
# so redirected/captured output is not corrupted by terminal control
↪sequences.
TTY_FLAGS=(-i)
if [ -t 0 ] && [ -t 1 ]; then
  TTY_FLAGS+=(-t)
fi

exec docker run --rm \
```

(continues on next page)

(continued from previous page)

```
  --network host \
  "${TTY_FLAGS[@]}" \
  --user "$(id -u):$(id -g)" \
  -e HOME="${HOME}" \
  "${MOUNTS[@]}" \
  -w "${WORKSPACE}" \
  -e DPSCTL_USERNAME \
  -e DPSCTL_HOST \
  -e DPSCTL_PORT \
  -e DPSCTL_CA_CERT_PATH \
  -e DPSCTL_CLIENT_CERT_PATH \
  -e DPSCTL_CLIENT_KEY_PATH \
  -e DPSCTL_AUTH_MODE \
  -e DPSCTL_INSECURE_TLS_SKIP_VERIFY \
  -e DPSCTL_INSECURE \
  -e DPSCTL_GRPC_TIMEOUT \
  -e DPSCTL_OUTPUT \
  "${IMAGE}:${VERSION}" \
  "$@"
EOF
chmod +x ~/.local/bin/dpsctl
```

Then make sure `~/.local/bin` is on your PATH:

```
export PATH="${HOME}/.local/bin:${PATH}"
```

What the wrapper does:

- Mounts only `~/.dpsctl` from the home directory so client credentials persist without exposing unrelated home-directory content to the container.
- Mounts the current working directory at the same path inside the container so commands like `dpsctl topology import ./datacenter.json` Just Work.
- Requires files referenced by `dpsctl` to be under the current working directory or `~/.dpsctl`, the only mounted paths.
- Uses `--network host` so the container can use host routes and names that the host resolves. Host-network support varies by container runtime.
- Passes through only the endpoint, TLS, authentication-mode, timeout, username, and output-format variables listed in the script. Add `-e` entries for any other *global `dpsctl` environment variables*, including OIDC client credentials.
- Does not pass `DPSCTL_PASSWORD` into the container. Use `--password-stdin` for non-interactive login. Refer to *dpsctl login*.
- Allocates a TTY only when stdin/stdout are interactive, so piped output stays clean.

2.5.1.1.5.12 Pinning a Different Version

`DPSCTL_IMAGE` and `DPSCTL_VERSION` can be overridden independently — the wrapper joins them as `${IMAGE}:${VERSION}`:

```
# Use a different image tag for one invocation
DPSCTL_VERSION="<release-version>" dpsctl --version
```

(continues on next page)

(continued from previous page)

```
# Override both repo and tag (e.g. internal mirror)
DPSCTL_IMAGE=nvcr.io/nvidia/dpsctl \
  DPSCTL_VERSION="<release-version>" dpsctl --version
```

2.5.1.1.5.13 Examples Using the Wrapper

After the shim is on your PATH, use the client login procedure for your server's setup in *Configure Server Authentication*. The following example uses LDAP login:

```
cp /path/to/api-ca.crt .
dpsctl login --host api.dps.example.com --port 443 \
  --ca-cert-path "${PWD}/api-ca.crt"
dpsctl topology import ./datacenter.json
dpsctl resource-group list
```

2.5.1.1.5.14 Install from Package Repositories

This guide covers installing the `dps-utils` package through the system package manager on supported Linux distributions.

Install the `dps-utils` package version that matches the DPS server version. By default, the package manager installs the latest available version. If a matching package is unavailable, use the corresponding pre-built binary or container image.

2.5.1.1.5.15 Pre-Requisites

1. The system package manager has been configured to use an NVIDIA package registry for the system's Linux distribution. For those using a CUDA local package registry on disk, it is recommended to update to the latest version available.

   Please refer to the CUDA installation guide for detailed steps.

2. A user needs sufficient permissions to install packages through the system package manager. This could be either a root user or a user with appropriate sudo privileges.

2.5.1.1.5.16 Supported Linux Distributions

Linux Distribution	x86_64	aarch64
Amazon Linux 2023	✓	✓
Azure Linux 3.0	✓	✓
Debian 12	✓	✓
RHEL 8.y/Rocky Linux 8.y	✓	✓
RHEL 9.y/Rocky Linux 9.y	✓	✓
RHEL 10.y/Rocky Linux 10.y	✓	✓
SLES/OpenSUSE 15.y	✓	✓
Ubuntu 22.04 LTS	✓	✓
Ubuntu 24.04 LTS	✓	✓

2.5.1.1.5.17 Installation

2.5.1.1.5.18 Amazon Linux 2023

1. Install the `dps-utils` package.

```
sudo dnf install --assumeyes dps-utils
```

2.5.1.1.5.19 Azure Linux 3.0

1. Install the `dps-utils` package.

```
sudo tdnf install --assumeyes dps-utils
```

2.5.1.1.5.20 Ubuntu LTS and Debian

1. Remove any previous installations of the `dps-utils` package.

```
sudo dpkg --list dps-utils &> /dev/null && \
  sudo apt purge --yes dps-utils
```

2. Update the package registry cache.

```
sudo apt-get update
```

3. Install the `dps-utils` package.

```
sudo apt-get install --yes dps-utils
```

2.5.1.1.5.21 RHEL / CentOS / Rocky Linux

1. Remove any previous installations of the `dps-utils` package.

```
sudo dnf list --installed dps-utils &> /dev/null && \
  sudo dnf remove --assumeyes dps-utils
```

2. Update the package registry cache.

```
sudo dnf clean expire-cache
```

3. Install the `dps-utils` package.

```
sudo dnf install --assumeyes dps-utils
```

2.5.1.1.5.22 SUSE SLES / OpenSUSE

1. Remove any previous installations of the `dps-utils` package.

```
sudo zypper search --installed-only --match-exact dps-utils &> /dev/null &
↪& \
  sudo zypper --non-interactive remove dps-utils
```

2. Update the package registry cache.

```
sudo zypper refresh
```

3. Install the `dps-utils` package.

```
sudo zypper install --no-confirm dps-utils
```

2.5.1.1.5.23 Post-Install

Verify the installation by checking the `dpsctl` version:

```
dpsctl --version
```

After the server is reachable, compare this output with `dpsctl server-version`. Keep the package pinned according to your operating system policy so an unrelated system update does not move the client ahead of the server.

2.5.1.1.5.24 Install Pre-built Binary

DPS provides pre-built binaries for the following platforms:

- **Linux**
- **macOS**

2.5.1.1.5.25 Download

Download the `dpsctl` archive for the same release as the server from the NVIDIA NVOnline Portal. Select the archive for your operating system and architecture, and verify it by using the checksum or signature supplied with that release.

2.5.1.1.5.26 Installation

```
VERSION="<release-version>"  # Use the server release without a leading v.
OS="linux"             # Use linux or darwin.
ARCH="amd64"           # Use amd64 or arm64.
ARCH_VARIANT="v1"      # Use v1 for amd64 or v8.0 for arm64.

tar -xzf "dpsctl_${OS}_${ARCH}_${ARCH_VARIANT}-${VERSION}.tgz"

# Make it executable
chmod +x dpsctl

# Move to system PATH (optional)
sudo mv dpsctl /usr/local/bin/
```

2.5.1.1.5.27 Verification

```
dpsctl --version
```

2.5.1.1.6 Prometheus Monitoring

2.5.1.1.6.1 Overview

`dps-server` exports Prometheus metrics through a dedicated HTTP metrics service. Use the chart-managed `ServiceMonitor` for an in-cluster Prometheus deployment. For an external Prometheus deployment, expose the metrics service through a customer-managed network and security layer.

This page covers DPS metrics discovery and scraping. It does not cover Prometheus installation, alerting, or dashboard configuration.

2.5.1.1.6.2 Metrics Endpoint

The default chart creates the following endpoint:

Property	Value
Service labels	`app: dps-server` and `app.kubernetes.io/component: metrics`
Service name	The chart-generated metrics Service, such as `dps-server-metrics` for a release named `dps` with default naming
Service port	`dps-metrics`, port `9090`, configurable through `dps.service.metrics.port`
Container port	`9090`, configurable through `dps.containerPorts.metrics.port`
Path	`/metrics`
Scheme	`http`. DPS does not provide TLS on the metrics listener.
Authentication	Disabled by default. Protected mode requires an OpenID Connect (OIDC) Bearer token with the `dps:metrics:read` scope.

2.5.1.1.6.3 Use the Chart-Managed ServiceMonitor

The Prometheus Operator and the `monitoring.coreos.com/v1` `ServiceMonitor` custom resource definition must exist in the cluster. Prometheus must discover the namespace that contains the `ServiceMonitor`, and the `ServiceMonitor` must select the DPS release namespace.

Enable the resource in your values overlay:

```
dps:
  serviceMonitor:
    enabled: true
    interval: 30s
    scrapeTimeout: 10s
    scheme: http
    additionalLabels:
      release: kube-prometheus-stack
```

Many Prometheus installations select `ServiceMonitor` resources by label. Check your Prometheus instance's `serviceMonitorSelector` to determine which labels it requires, and set them through `dps.serviceMonitor.additionalLabels`. Replace the example `release` label with the labels required by your Prometheus instance, then apply the chart:

```
helm upgrade --install dps ngc/dps \
  --namespace dps \
  --values values.yaml
```

To create the `ServiceMonitor` in another namespace, set its namespace and the namespace that contains the metrics Service:

```
dps:
  serviceMonitor:
    enabled: true
    namespace: monitoring
    namespaceSelector:
      matchNames:
        - dps
```

The selected chart's `values.yaml` is authoritative for the available `dps.serviceMonitor` settings. Use *Inspect the Versioned Configuration* to inspect the version you are deploying.

2.5.1.1.6.4 Protect the Metrics Endpoint

Set `dps.metrics.authentication.enabled: true` to require an OIDC Bearer token with the `dps:metrics:read` scope. Protected metrics require OIDC, cannot be combined with LDAP, and require an encrypted scrape path. When the chart-managed `ServiceMonitor` is enabled, configure exactly one credential profile. OAuth 2.0 client credentials are preferred because Prometheus refreshes the token.

Enabling `dps.auth.oidc.enabled` switches DPS API authentication to external OIDC and disables DPS `/Token` issuance. Treat this as an authentication mode change, not only a metrics setting. Configure user and service clients by following *Configure OIDC Mode*. The following example uses scope-based API access without an additional group check. The `ServiceMonitor` OAuth 2.0 field requires Prometheus 2.27.0 or later.

Protected metrics with OAuth 2.0 client credentials

```
dps:
  auth:
    auth_required_groups: ""
    oidc:
      enabled: true
      issuer: "https://idp.example.com"
      jwksUrl: "https://idp.example.com/.well-known/jwks.json"
      audience: "dps-api"
  ldap:
    enabled: false
  metrics:
    authentication:
      enabled: true
  serviceMonitor:
    enabled: true
    oauth2:
      clientId:
        secret:
          name: "prometheus-oidc"
          key: "client-id"
      clientSecret:
        name: "prometheus-oidc"
        key: "client-secret"
      tokenUrl: "https://idp.example.com/oauth2/token"
```

Create `prometheus-oidc` through your approved Secret workflow. Create each referenced Secret in the namespace that contains the `ServiceMonitor`, and ensure that the Prometheus Operator can read it. The chart adds the `dps:metrics:read` scope to the OAuth 2.0 request.

As an alternative, replace the preceding `oauth2` block with a reference to a customer-managed access token:

```
dps:
  serviceMonitor:
    authorization:
      type: Bearer
      credentials:
        name: "prometheus-token"
        key: "access-token"
```

Do not configure both credential profiles. Replace a customer-managed token before it expires.

The direct metrics listener remains HTTP in protected mode. Carry Bearer tokens only over an encrypted path supplied by a transparent service mesh, proxy, or other layer that fronts the selected Service. Set `dps.serviceMonitor.scheme` to `https` and configure `tlsConfig` only when that layer presents HTTPS at the endpoint selected by the `ServiceMonitor`.

2.5.1.1.6.5 Verify the ServiceMonitor

Confirm that the Service and `ServiceMonitor` exist. Adjust the namespaces when the `ServiceMonitor` is deployed separately:

```
kubectl get service --namespace dps \
  --selector app=dps-server,app.kubernetes.io/component=metrics
kubectl get servicemonitor --namespace dps
kubectl describe servicemonitor --namespace dps dps-server-metrics
```

In the Prometheus UI, open **Status > Targets** and confirm that the `dps-server-metrics` target is **UP**.

When metrics authentication is disabled, you can also test the direct endpoint from the cluster. The following service name assumes that the Helm release is named `dps`:

```
kubectl run --namespace dps --rm -it --restart=Never curl \
  --image=curlimages/curl -- \
  curl -sS http://dps-server-metrics:9090/metrics
```

For protected metrics, verify the scrape through Prometheus. Do not place an access token on a command line.

2.5.1.1.6.6 Use a Standalone ServiceMonitor

If another team manages `ServiceMonitor` objects, apply a resource that selects the chart-managed metrics Service. Adjust the chart name, release instance, and namespaces for your deployment.

Standalone ServiceMonitor example

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: dps-server-metrics
  namespace: monitoring
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app: dps-server
      app.kubernetes.io/name: dps
      app.kubernetes.io/instance: dps
      app.kubernetes.io/component: metrics
  namespaceSelector:
    matchNames:
      - dps
  endpoints:
    - port: dps-metrics
      path: /metrics
      interval: 30s
      scrapeTimeout: 10s
      scheme: http
```

When metrics authentication is enabled, add exactly one credential profile under the endpoint. A standalone `ServiceMonitor` must request the metrics scope explicitly.

OAuth 2.0 endpoint configuration

```
oauth2:
  clientId:
    secret:
      name: prometheus-oidc
      key: client-id
  clientSecret:
    name: prometheus-oidc
    key: client-secret
  tokenUrl: https://idp.example.com/oauth2/token
  scopes:
    - dps:metrics:read
```

As an alternative, add a customer-managed Bearer token profile:

```
authorization:
  type: Bearer
  credentials:
    name: prometheus-token
    key: access-token
```

Create referenced Secrets in the `ServiceMonitor` namespace, and ensure that the Prometheus Operator can read them. Configure the DPS authentication settings described in *Protect the Metrics Endpoint* before you enable either profile.

2.5.1.1.6.7 Scrape Metrics From Outside the Cluster

The chart's `dps.ingress` and `dps.gateway` resources route to the gRPC Service. They do not expose `/metrics`. The dedicated metrics Service is a `ClusterIP` Service by default and listens on HTTP port `9090`.

Expose the metrics Service through a path appropriate for your environment:

- When metrics authentication is disabled, a customer-managed `NodePort` or `LoadBalancer` Service can select the `dps-server` pods and set `targetPort` to `dps-metrics`. Restrict access to Prometheus with firewall rules, `NetworkPolicy`, load balancer source ranges, or equivalent controls.
- Across routed or shared networks, use an Ingress, proxy, or service mesh that terminates TLS and restricts access. An encrypted path is required when metrics authentication is enabled.

The following Ingress example terminates TLS and limits source addresses:

Metrics Ingress example

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dps-server-metrics
  namespace: dps
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "<prometheus-source-
↪cidr>"
spec:
```

(continues on next page)

(continued from previous page)

```
  ingressClassName: nginx
  tls:
    - hosts:
        - dps-metrics.example.com
      secretName: dps-metrics-tls
  rules:
    - host: dps-metrics.example.com
      http:
        paths:
          - path: /metrics
            pathType: Prefix
            backend:
              service:
                name: dps-server-metrics
                port:
                  name: dps-metrics
```

Configure external Prometheus with the exposed endpoint and the trust material for the customer-managed TLS layer. The following example assumes that the Ingress certificate chains to a certificate authority that Prometheus already trusts. For a private certificate authority, configure `tls_config.ca_file` as described in the Prometheus configuration reference.

External Prometheus scrape example

```
scrape_configs:
  - job_name: dps-server
    metrics_path: /metrics
    scheme: http
    scrape_interval: 30s
    scrape_timeout: 10s
    static_configs:
      - targets:
          - dps-metrics.example.com
        labels:
          service: dps-server
          cluster: my-dps-cluster
```

When metrics authentication is enabled, configure the external Prometheus job to obtain or supply the required OIDC Bearer token. Prefer OAuth 2.0 client credentials so Prometheus refreshes the token automatically. Add the following configuration to the scrape job and mount the client secret file through your approved Secret workflow:

```
oauth2:
  client_id: "<prometheus-client-id>"
  client_secret_file: /etc/prometheus/secrets/dps/client-secret
  token_url: https://idp.example.com/oauth2/token
  scopes:
    - dps:metrics:read
```

Verify the unprotected Ingress endpoint from the Prometheus host:

```
curl -sS https://dps-metrics.example.com/metrics
```

For protected metrics, verify the scrape through Prometheus instead of placing an access token on a command line.

2.5.1.1.6.8 Troubleshooting

Symptom	Likely cause	Action
`ServiceMonitor` exists but Prometheus shows no target	Prometheus does not select its labels or namespace.	Match `dps.serviceMonitor.additionalLabels` and namespace settings to the Prometheus selectors.
Target is `DOWN` with a TLS error	The customer-managed proxy, Ingress, or mesh certificate cannot be validated.	Configure `scheme` and `tlsConfig` for that layer. `dps.serverTLS` does not configure metrics TLS.
Target is `DOWN` or returns `404`	The scraper uses the UI Service, port `8000`, or the wrong scheme.	Select the dedicated metrics Service and use `dps-metrics` over HTTP unless a customer TLS layer is present.
Scrape returns `401 Unauthorized`	The Bearer token is missing, malformed, expired, or invalid.	Verify the configured credential profile, referenced Secret, and OIDC settings.
Scrape returns `403 Forbidden`	The token does not include `dps:metrics:read`.	Issue a token with the required scope. The chart-managed OAuth 2.0 profile requests it automatically.
External Prometheus cannot connect	The network path, firewall, or external endpoint is unavailable.	Test the exact exposed endpoint and inspect the customer-managed network resource.

2.5.1.1.6.9 Related Pages

- *Deployment and Configuration Guide* – chart installation and external endpoint configuration.
- *Telemetry and Observability* – how DPS ingests live power data, which is separate from Prometheus metrics.

2.5.1.1.7 Redfish Proxy Configuration

2.5.1.1.7.1 Overview

Use this guide when DPS sends BMC Redfish requests through a Redfish proxy instead of connecting directly to each BMC. The proxy receives DPS Redfish requests and routes each request to the correct backend BMC.

This page describes the DPS-side configuration for proxy endpoint selection, routing headers, proxy authentication, TLS, topology routing data, and verification.

2.5.1.1.7.2 Prerequisites

Before configuring DPS, make sure you have:

- A Redfish proxy endpoint that the DPS agent pods can reach.
- The authentication and TLS material required by the proxy.

- Topology entities that include the field used by the routing header mapping, such as `Redfish.TargetIP`, `Redfish.URL`, `Properties.MacAddress`, or `Properties.SerialNumber`.

2.5.1.1.7.3 Configure the Proxy Endpoint

Set `dps.redfish.proxy.url` to the proxy endpoint for your deployment:

```
dps:
  redfish:
    proxy:
      # Replace with the Redfish proxy URL for your deployment.
      url: "https://redfish-proxy.example.svc.cluster.local:1079"
```

When `dps.redfish.proxy.url` is set, DPS sends Redfish traffic through that proxy endpoint.

2.5.1.1.7.4 Configure Proxy Transport

DPS supports proxy-specific transport settings under `dps.redfish.proxy.transport`. Use `protocol: auto` for TLS-enabled proxy services so DPS can use HTTP/2 when the proxy supports it and fall back to HTTP/1.1 when it does not. Use `protocol: http1` only when the proxy requires HTTP/1.1 or has protocol negotiation issues.

The proxy transport also supports connection limits for the DPS-to-proxy connection path. These settings apply only when `dps.redfish.proxy.url` is set:

```
dps:
  redfish:
    proxy:
      # Replace with the Redfish proxy URL for your deployment.
      url: "https://redfish-proxy.example.svc.cluster.local:1079"
      transport:
        protocol: auto
        # Tune these values for the proxy capacity and expected DPS
↪concurrency.
        maxConnsPerHost: 64
        maxIdleConns: 128
        maxIdleConnsPerHost: 64
```

Omit connection limit fields that you do not want to set explicitly to keep DPS defaults. When proxy mode is enabled, DPS uses one shared DPS-to-proxy connection path for read and write traffic.

2.5.1.1.7.5 Configure Routing Headers

DPS can add Redfish request headers that carry routing metadata for a proxy. A proxy can use those header values to route each request to the correct backend BMC.

In each header entry, `entityFieldValue` tells DPS how to derive the header value from the target topology entity. It can be a direct dotted path such as `Properties.SerialNumber`, or a constrained template such as `host={{ .Redfish.TargetIP }}`.

Topology entities can include `Redfish.URL` even when `dps.redfish.proxy.url` is configured. In that mode, DPS uses `Redfish.URL` as the original Redfish target and sends the outbound request through the proxy. Header mappings can derive routing metadata from `Redfish.URL`:

```
dps:
  redfish:
    headers:
      additionalHeaders:
        - name: Forwarded
          entityFieldValue: "host={{ (url .Redfish.URL).Hostname }}"
```

2.5.1.1.7.6 Default Headers

Use `dps.redfish.headers.additionalHeaders` to define headers that apply when the topology entity does not define the same header.

```
dps:
  redfish:
    headers:
      additionalHeaders:
        - name: Forwarded
          entityFieldValue: "host={{ .Redfish.TargetIP }}"
```

2.5.1.1.7.7 Per-Entity Topology Headers

Use `Redfish.AdditionalHeaders` in the topology file when a specific entity needs its own header mapping:

Header mappings resolve values against the target topology entity. The topology configuration must include the routing field referenced by the header mapping. Common routing fields include:

- `Redfish.TargetIP`
- `Properties.MacAddress`
- `Properties.SerialNumber`

```
{
  "Type": "ComputerSystem",
  "Model": "DGX_GB300",
  "Name": "node001",
  "Properties": {
    "SerialNumber": "SN-GB300-01",
    "MacAddress": "00:11:22:33:44:55"
  },
  "Redfish": {
    "TargetIP": "10.10.20.9",
    "AdditionalHeaders": [
      {
        "Name": "Forwarded",
        "EntityFieldValue": "host={{ .Redfish.TargetIP }}"
      }
    ]
  }
}
```

2.5.1.1.7.8 Override Headers

Use `additionalHeadersOverride` when Helm values must take precedence over per-entity `Redfish.AdditionalHeaders` entries with the same header name. Override headers use the same `entityFieldValue` format as default headers; DPS still resolves the value from each target topology entity:

```
dps:
  redfish:
    headers:
      additionalHeadersOverride:
        - name: Forwarded
          entityFieldValue: "host={{ .Redfish.TargetIP }}"
```

Header precedence is, from highest to lowest:

1. `additionalHeadersOverride` from DPS Helm values, for deployment-level forced header mappings.
2. `Redfish.AdditionalHeaders` from each topology entity, for per-entity headers in the topology file.
3. `additionalHeaders` from DPS Helm values, for deployment-level default headers.

   Note: If a referenced entity field is missing, DPS skips that header entry for that entity. `Authorization` and `X-Auth-Token` are reserved and cannot be configured as additional headers.

2.5.1.1.7.9 Configure Authentication

For proxy deployments, the most common Redfish auth methods are `none` and `bearerToken`.

Use `none` when DPS should not send a Redfish authentication header:

```
dps:
  redfish:
    auth:
      method: none
```

Use `bearerToken` when DPS should send `Authorization: Bearer ...` to the proxy. For a shared proxy token, set `defaultSecretName` to the mounted Secret name so DPS can resolve the token when the topology entity does not define `Redfish.SecretName`:

```
dps:
  redfish:
    auth:
      method: bearerToken
      defaultSecretName: redfish-proxy-bearer-token
      bearerToken:
        existingSecret: redfish-proxy-bearer-token
        existingSecretKey: token
```

Use `session` for direct BMC connections or for a proxy path that uses DPS-managed Redfish session authentication.

2.5.1.1.7.10 Configure TLS

DPS supports `insecure`, `tls`, and `mtls` Redfish transport settings. For production proxy deployments, use `tls` or `mtls` with certificate verification enabled. The following example configures mTLS for a proxy that requires DPS to present a client certificate:

```
dps:
  redfish:
    tls:
      transportSecurity: mtls
      insecureSkipVerify: false
      certs:
        certFile: /secrets/redfish-proxy/tls.crt
        certKeyFile: /secrets/redfish-proxy/tls.key
        caFile: /secrets/redfish-proxy/ca.crt

  secrets:
    - name: dps-redfish-proxy-client
      secretKey: tls.crt
      mountPath: /secrets/redfish-proxy/tls.crt
    - name: dps-redfish-proxy-client
      secretKey: tls.key
      mountPath: /secrets/redfish-proxy/tls.key
    - name: dps-redfish-proxy-client
      secretKey: ca.crt
      mountPath: /secrets/redfish-proxy/ca.crt
```

2.5.1.1.7.11 Verify the Configuration

After deployment, validate and import the topology, and then activate it. Activation routes each managed node to its assigned DPS Agent and applies the node's power policy through that Agent's Redfish configuration, including the proxy. The `--at-least-percent-hosts` flag, which requires `--ping-hosts`, sets the minimum percentage of managed nodes that must activate successfully. The default is `50`. A value of `0`, or `--ping-hosts=false`, requires every node to succeed. If fewer nodes succeed than the threshold requires, activation fails, DPS unsets the policy on the nodes that had already applied it, and the topology stays inactive. Follow *Managing Topologies* for activation prerequisites and effects:

```
dpsctl topology validate topology.json
dpsctl topology import topology.json
dpsctl --grpc-timeout 20m topology activate \
  --topology <topology-name> \
  --ping-hosts \
  --at-least-percent-hosts <minimum-success-percent>
dpsctl topology list --active
```

Do not rely on the `dpsctl` exit status, which can be `0` even when activation fails. Confirm that the activation output reports `status.ok: true` and that the topology appears in the active list. Activation can also succeed while some nodes fail, so review each node's result in `node_statuses`. A successful node reports `status.ok: true`, and a failed node reports only `status.diag_msg` with the error.

Activation waits for a final result from every managed node and retries unreachable nodes for up to about 15 minutes. The example sets the global `--grpc-timeout` option to `20m` because the default is 1 minute. An Ingress or gateway between `dpsctl` and DPS can also end the request sooner if its read timeout is shorter. If the request times out, DPS rolls back the activation after it finishes contacting

the nodes, so the topology does not stay active. Correct the unreachable nodes or the timeouts, and then activate the topology again.

Import does not route a topology's nodes to an Agent, so the connection and BMC health checks cannot validate a newly imported topology until you activate it. After activation succeeds, run both checks against the managed nodes in the topology:

```
dpsctl check connection --topology <topology-name>
dpsctl verify bmc-health start --topology <topology-name> --skip-writes --
↪wait --summary-only
```

Start with `--skip-writes` so the first health check is read-only.

2.5.1.1.7.12 Example: NVIDIA Infra Controller (NICo)

This example shows DPS configured with NVIDIA Infra Controller (NICo) `nico-bmc-proxy`. DPS authenticates to the proxy with mTLS, sends routing metadata in the `Forwarded` header, and uses `redfish.auth.method: none`.

```yaml
dps:
  redfish:
    proxy:
      # Replace with the NICo BMC proxy service URL for your deployment.
      url: "https://nico-bmc-proxy.nico-system.svc.cluster.local:1079"
    auth:
      method: none
    tls:
      transportSecurity: mtls
      insecureSkipVerify: false
      certs:
        certFile: /secrets/nico-bmc-proxy/tls.crt
        certKeyFile: /secrets/nico-bmc-proxy/tls.key
        caFile: /secrets/nico-bmc-proxy/ca.crt
    headers:
      additionalHeaders:
        - name: Forwarded
          entityFieldValue: "host={{ .Redfish.TargetIP }}"

  secrets:
    - name: dps-nico-bmc-proxy-client
      secretKey: tls.crt
      mountPath: /secrets/nico-bmc-proxy/tls.crt
    - name: dps-nico-bmc-proxy-client
      secretKey: tls.key
      mountPath: /secrets/nico-bmc-proxy/tls.key
    - name: dps-nico-bmc-proxy-client
      secretKey: ca.crt
      mountPath: /secrets/nico-bmc-proxy/ca.crt
```

For this NICo proxy path, do not create or mount DPS-owned BMC credential secrets. The topology does not need `Redfish.URL` or `Redfish.SecretName` because the proxy routes each Redfish request to the backend BMC and owns BMC credential management.

2.5.2. Concepts

2.5.2.1 Overview

Dynamic Power Software (DPS) manages datacenter power through several key abstractions. Understanding these abstractions is essential for effectively using DPS.

2.5.2.2 Key Abstractions

2.5.2.2.1 Command-line Tools

- *dpsctl*

2.5.2.2.2 Physical Infrastructure

- *Device Specifications*
- *Entities*
- *Topologies*

2.5.2.2.3 Power Management

- *Power Compliance*
- *Power Policies*
- *DPS Power Steering Controller*
- *Resource Groups*
- *Resource Group Priority*
- *Workload Power Profiles Settings*
- *MaxLPS Power Management*
- *Excursion Mitigation*
- *NvGrid*
- *Provisioned and Observed Power*

2.5.2.2.4 User Management

- *User Accounts*
- *Service Accounts*

2.5.2.2.5 Infrastructure

- *RedFish API*
- *Base Command Manager*
- *Telemetry and Observability*

2.5.2.3 Next Steps

- **New to DPS?** Start with *Topologies* to understand how DPS models datacenters
- **Managing workloads?** Learn about *Resource Groups* for dynamic power allocation, *Resource Group Priority* for steal and giveback ordering, *DPS Power Steering Controller* for automatic power redistribution, and *Workload Power Profiles Settings* for GPU optimization
- **Sizing a deployment?** Read *Provisioned and Observed Power* for why DPS lets you provision closer to your power envelope than static sizing allows
- **Setting up hardware?** Review *Device Specifications* and *Entities*
- **Configuring access?** Refer to *User Accounts* and *Credentials Configuration*

2.5.2.3.1 Entities

2.5.2.3.1.1 Overview

Entities represent physical equipment instances in your data center, such as servers, power distribution units, GPUs, switches, and other hardware components. Each entity instantiates a device specification and provides deployment details such as BMC endpoints and site-specific operating limits.

2.5.2.3.1.2 Entity Structure

Each entity defines:

- **Name** - Unique identifier (e.g., “node001”, “rack-pdu-a1”)
- **Type** - Device category from device specification
- **Model** - Specific device model from device specification
- **Redfish** - BMC endpoint configuration for management
- **Policy** - Optional legacy entity policy name retained for compatibility. It does not select a policy bundle.
- **OperatingLimit** - Optional site-specific power cap for `PowerDomain`, `PowerDistribution`, and `PowerSupply` entities
- **StaticLoad** - Fixed power consumption (optional)
- **TelemetryId** - Sensor ID for obtaining telemetry data (optional)

2.5.2.3.1.3 Entity Types and Examples

2.5.2.3.1.4 Compute Entities

Compute Nodes (ComputerSystem):

```
{
  "Type": "ComputerSystem",
  "Model": "DGX_H100",
  "Name": "node001",
  "Redfish": {
    "URL": "https://node001-bmc.example.com",
    "SecretName": "node001"
  },
```

(continues on next page)

(continued from previous page)

```
  "TelemetryId": "sensor-node001"
}
```

2.5.2.3.1.5 Power Distribution Entities

Power Domains (top-level power sources):

```
{
  "Type": "PowerDomain",
  "Name": "PD-A",
  "OperatingLimit": {
    "PowerValue": {"Value": 1150000, "Type": "W"},
    "PowerFactor": 0.9
  }
}
```

Rack PDUs (power distribution units):

```
{
  "Type": "PowerDistribution",
  "Model": "RackPDU95_57500W",
  "Name": "rPDU-A1-001",
  "StaticLoad": {"Value": 3400, "Type": "W"}
}
```

2.5.2.3.1.6 Redfish Integration

Compute entities include Redfish configuration for BMC communication:

```
"Redfish": {
  "URL": "https://node001-bmc.example.com",
  "SecretName": "node001"
}
```

- **URL** - BMC endpoint address (hostname or IP)
- **SecretName** - Reference to stored authentication credentials

2.5.2.3.1.7 Power Configuration

2.5.2.3.1.8 Operating Limit

Specify an operating cap on a `PowerDomain`, `PowerDistribution`, or `PowerSupply` entity:

```
"OperatingLimit": {
  "PowerValue": {"Value": 1150000, "Type": "W"},
  "PowerFactor": 0.9
}
```

2.5.2.3.1.9 Static Load

Add fixed power consumption from unmanaged devices:

```
"StaticLoad": {"Value": 3400, "Type": "W"}
```

2.5.2.3.1.10 Example: Complete Entity Definition

```
{
  "Entities": [
    {
      "Type": "PowerDomain",
      "Name": "PD-A",
      "OperatingLimit": {
        "PowerValue": {"Value": 1150000, "Type": "W"},
        "PowerFactor": 0.9
      }
    },
    {
      "Type": "PowerDistribution",
      "Model": "RackPDU95_57500W",
      "Name": "rPDU-A1-001",
      "StaticLoad": {"Value": 3400, "Type": "W"}
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_H100",
      "Name": "node001",
      "Redfish": {
        "URL": "https://node001-bmc.example.com",
        "SecretName": "node001"
      },
      "TelemetryId": "sensor-node001"
    }
  ]
}
```

2.5.2.3.1.11 BMC Credentials

Entity BMC access requires secure credential management through *secrets*:

```
# Store BMC credentials
kubectl create secret generic node001 \
  --namespace=dps \
  --type=Opaque \
  --from-literal='bmc={"username":"admin","password":"secret123"}'

# Add Label to BMC credential
kubectl label secret node001 \
  --namespace=dps \
  app=bmc-secret
```

(continues on next page)

(continued from previous page)

```
# Entity references the secret
"Redfish": {
  "URL": "https://node001-bmc.example.com",
  "SecretName": "node001"
}
```

2.5.2.3.1.12 Entity Generation

There are multiple ways that entities can be imported into DPS.

2.5.2.3.1.13 Manual Definition

Define entities explicitly in topology JSON files. Refer to *Import a Topology*.

```
# Import entities from topology file
dpsctl topology import datacenter.json
```

2.5.2.3.1.14 BCM Integration

For Base Command Manager (BCM) environments, DPS can auto-generate entities using an asynchronous import workflow:

```
# Step 1: Start entity generation from BCM (returns a request ID)
dpsctl import entity --source bcm \
  --username admin \
  --password secret123

# Sample Output:
{
  "request_id": "4826e1b3-cb36-42eb-97d5-0eaa9e9ee346"
}
```

```
# Step 2: Retrieve generated entities using the request ID
dpsctl import entity-status --request-id <request-id>

# Sample Output:
[
  {
    "Type": "ComputerSystem",
    "Model": "DGX_H100",
    "Name": "node002",
    "Redfish": {
      "SecretName": "node002",
      "URL": "https://node002"
    },
    "UpdatedAt": "0001-01-01T00:00:00Z"
  },
  {
    "Type": "ComputerSystem",
    "Model": "DGX_H100",
    "Name": "node003",
```

(continues on next page)

(continued from previous page)

```
    "Redfish": {
      "SecretName": "node003",
      "URL": "https://node003"
    },
    "UpdatedAt": "0001-01-01T00:00:00Z"
  },
  {
    "Type": "ComputerSystem",
    "Model": "DGX_H100",
    "Name": "node004",
    "Redfish": {
      "SecretName": "node004",
      "URL": "https://node004"
    },
    "UpdatedAt": "0001-01-01T00:00:00Z"
  }
]
```

2.5.2.3.1.15 Nautobot Integration

Similar to BCM Environments, DPS can auto-generate entities from Nautobot's DCIM Inventory using an asynchronous import workflow:

```
# Step 1: Start entity generation from Nautobot (returns a request ID)
dpsctl import entity --source nautobot \
  --locations "DC-1" \
  --racks "Rack-A1,Rack-A2" \
  --device-types "DGX_H100"

# Sample Output:
{
  "request_id": "b12d520d-f3e6-4603-b2bf-74dd55617888"
}
```

```
# Step 2: Retrieve generated entities using the request ID
dpsctl import entity-status --request-id <request-id>

# Sample Output:
[
  {
    "Type": "ComputerSystem",
    "Model": "DGX_B200",
    "Name": "computerSystem001",
    "UpdatedAt": "0001-01-01T00:00:00Z"
  },
  {
    "Type": "ComputerSystem",
    "Model": "DGX_B200",
    "Name": "computerSystem002",
    "UpdatedAt": "0001-01-01T00:00:00Z"
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "Type": "ComputerSystem",
    "Model": "DGX_B200",
    "Name": "computerSystem003",
    "UpdatedAt": "0001-01-01T00:00:00Z"
  }
]
```

See the *import entity* command reference for full details.

2.5.2.3.1.16 Usage

Entities are imported as part of topology files and can be queried through dpsctl:

```
# Import entities and topology
dpsctl topology import datacenter.json

# List entities
dpsctl topology list-entities

# Show entity details
dpsctl topology list-entities node001
```

2.5.2.3.1.17 Further Reading

- *Import a Topology* - Validate and import entities as part of a topology file
- *Device Specifications* - Templates that define entity capabilities
- *Topologies* - Connecting entities together
- *Power Policies* - Managing entity power behavior
- *Telemetry* - Understanding telemetry data collection
- *Secrets* - Storing BMC credentials securely
- *Nautobot Import* - Importing Entities from Nautobot

2.5.2.3.2 Topologies

2.5.2.3.2.1 Overview

Topologies model the physical power-distribution structure of a data center. They define the equipment in a managed area and the parent-child relationships that describe power flow.

A topology import file can contain:

- **Entities**: Physical equipment instances.
- **Topology**: Parent-child connections between those instances.

Multiple topologies can describe different data center sections or configurations, but DPS can manage only one active topology at a time.

A topology also has one persisted topology-wide policy-bundle selection. The selection is not stored in topology JSON and is not assigned per entity. Refer to *Power Policies* for selection, activation, and fallback behavior.

2.5.2.3.2.2 Topology Deactivation and Loss of DPS Control

Topology deactivation and a DPS service outage have different effects on the policies that a Baseboard Management Controller (BMC) enforces. The following table summarizes the operator-visible behavior:

Table 2.1: Policy Behavior When DPS Stops Applying Policies

Event	DPS behavior	BMC behavior
Explicit topology deactivation	DPS marks the topology inactive and sends an `UnsetPolicy` request to each managed node. Active resource groups prevent topology deactivation.	The device plugin applies the configured idle policy. If no idle policy is configured, the plugin restores the BMC defaults or removes the DPS-managed policy, depending on the platform.
Planned DPS shutdown, accidental DPS server or agent pod deletion, or temporary DPS outage	DPS sends no deactivation or policy-change request while it is unavailable.	If the BMC remains operational, it continues enforcing the last successfully applied power limits.
Node restart, BMC restart, or firmware maintenance	DPS cannot guarantee that a policy survives a reset of the system that enforces it.	The BMC can return to its default limits. Reintroduce the node through a topology or resource-group policy operation to put it back under DPS control.

A topology-deactivation response confirms that the topology is inactive. Device cleanup is best effort, so the response does not guarantee that every BMC accepted the `UnsetPolicy` request. Verify the power limits when maintenance or another operational requirement depends on a specific hardware state.

For the maintenance workflow, refer to *Prepare Nodes for Maintenance*.

2.5.2.3.2.3 Topology Definition

Topology import files use JSON and follow DMTF Redfish terminology where possible.

2.5.2.3.2.4 Import Structure

Each topology import file defines one topology with at least one topology entity, plus zero or more data center inventory entities:

```
{
  "Entities": [
    // Entity definitions
  ],
  "Topology": {
    "Name": "topologyName",
    "Entities": [
      // Topology relationships
    ]
  }
}
```

A topology-only file can be imported when every referenced entity already exists. Include missing entities in the same JSON file, or create them through the `CreateEntities` API stream before you create the topology.

2.5.2.3.2.5 Topology Policy Selection

The topology import format does not contain a policy-bundle field. Consequently, `dpsctl topology import` sends an empty `CreateUpdateTopologyRequest.CreateRequest.policy_bundle`, and the server resolves and persists the current `global_policy_bundle` value. API clients that create or update a topology can provide an explicit `policy_bundle` value in the corresponding nested request instead.

Importing or creating a topology does not change hardware. The selected bundle takes effect when the topology activates. For the full lifecycle, refer to *Policy Lifecycle and Fallback*.

2.5.2.3.2.6 Validation

The file format is backed by the topology JSON schema, which is based on a reduced Redfish schema with DPS additions. `dpsctl topology validate` performs local schema validation and then calls the server's non-mutating `ValidateTopology` RPC. During import, the server performs semantic validation as part of the mutating `CreateUpdateTopology` RPC.

Refer to *Topology Schema Reference* for the detailed format.

2.5.2.3.2.7 Topology Entities

The topology's `Entities` array represents entity instances and their one-to-many child connections:

```
{
  "Name": "PD-A",
  "Children": [
    "PDU-A1",
    "PDU-A2",
    "PDU-A3"
  ]
}
```

This relationship means that `PD-A` powers the three child PDUs. Each `Name` refers to an entity in the data center inventory and must be unique within the topology array. Server validation rejects invalid connectivity such as cycles and disconnected entities.

A topology can include a subset of the defined inventory entities.

A topology entity can override supported properties of the inventory entity. For example, this topology entity overrides `OperatingLimit` without changing the other inventory properties:

```
{
  "Name": "PD-A",
  "OperatingLimit": {
    "PowerValue": {
      "Value": 1150000,
      "Type": "W"
    },
    "PowerFactor": 0.9
  },
```

(continues on next page)

(continued from previous page)

```
  "Children": ["node001"]
}
```

2.5.2.3.2.8 Topology and Device Efficiency Factors

Two independent factors account for different power losses:

- `OperatingLimit.PowerFactor` in topology JSON represents site-specific losses between connected entities. It acts as an efficiency modifier: DPS multiplies the entity's operating limit by this value.
- `spec.efficiencyFactor` in the device specification represents losses within the distribution device. DPS divides the combined child load by this value to calculate the device's input load.

Both factors apply; the topology value does not override the device specification. Consider a single distribution entity without static load or overprovisioning. With a 240 kW operating limit, a `PowerFactor` of `0.9`, and an `efficiencyFactor` of `0.95`, it allows up to 205.2 kW for its children (`240 × 0.9 × 0.95 = 205.2`).

2.5.2.3.2.9 Shared GPU Topology Constraints

Shared-GPU resource groups still have to satisfy these topology relationships after a GPU policy update. DPS rejects a shared-GPU update whose post-clamp aggregate load would violate a non-compute ancestor such as a rack or PDN device, even when the resource-group GPU budget still has headroom. Refer to *Shared GPU Budget*.

2.5.2.3.2.10 Example

A complete topology definition with one power domain, one PDU, one rack PDU, and two computer systems.

Complete topology definition

```
{
    "Entities": [
        {
            "Type": "PowerDomain",
            "Name": "PD-A",
            "OperatingLimit": {
                "PowerValue": {
                    "Value": 1150000,
                    "Type": "W"
                },
                "PowerFactor": 0.9
            }
        },
        {
            "Type": "PowerDistribution",
            "Model": "FloorPDU95",
            "Name": "PDU-A1"
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "Type": "PowerDistribution",
            "Model": "RackPDU95_57500W",
            "Name": "rPDU-A1-001",
            "StaticLoad": {
                "Value": 3400,
                "Type": "W"
            }
        },
        {
            "Type": "ComputerSystem",
            "Model": "DGX_H100",
            "Name": "viking001",
            "Redfish": {
                "URL": "https://viking001",
                "SecretName": "viking001"
            }
        },
        {
            "Type": "ComputerSystem",
            "Model": "DGX_H100",
            "Name": "viking002",
            "Redfish": {
                "URL": "https://viking002",
                "SecretName": "viking002"
            }
        }
    ],
    "Topology": {
       "Name": "pdn",
       "Entities": [
            {
                "Name": "PD-A",
                "Children": [
                    "PDU-A1"
                ]
            },
            {
                "Name": "PDU-A1",
                "Children": [
                    "rPDU-A1-001"
                ]
            },
            {
                "Name": "rPDU-A1-001",
                "Children": [
                    "viking001",
                    "viking002"
                ]
            },
            {
                "Name": "viking001"
            },
```

(continues on next page)

(continued from previous page)

```
            {
                "Name": "viking002"
            }
        ]
    }
}
```

2.5.2.3.2.11 Usage

Use `dpsctl` to validate, import, list, and activate topologies:

```
# Validate without mutating the database
dpsctl topology validate datacenter.json

# Import topology from JSON file (created topology is inactive)
dpsctl topology import datacenter.json
dpsctl topology list
dpsctl topology activate --topology pdn
```

An active topology must be deactivated before a different topology can be activated.

2.5.2.3.2.12 Further Reading

For import procedures and related concepts, refer to:

- *Import a Topology*
- *Managing Topologies*
- *Entities*
- *Power Policies*
- *Device Specifications*

2.5.2.3.3 Resource Groups

2.5.2.3.3.1 Overview

Resource groups dynamically associate active-topology resources with a workload. Their policy bundle has higher precedence than the topology bundle, allowing workload-specific behavior without changing the topology selection.

2.5.2.3.3.2 Resource Group Structure

A resource group contains:

- Workload identification and metadata.
- Member compute resources.
- A persisted policy-bundle selection.
- Lifecycle and activation state.
- Optional workload-profile and power-management settings.

- Relative priority for power stealing and giveback. Lower numbers are higher priority, and the default is `0`.

2.5.2.3.3.3 Lifecycle

Resource groups support these operations:

1. **Create** an inactive group.
2. **Add or remove** member resources.
3. **Activate** the group and apply its effective policies.
4. **Update** membership or supported workload settings.
5. **Delete** the group and release its resources.

Membership changes on an inactive group are stored without applying hardware policy. Changes on an active group apply or release policy immediately. Membership changes are rejected while activation or deactivation is in progress.

2.5.2.3.3.4 Policy Selection and Precedence

A topology binding remains the lower-precedence policy source for its managed devices. When a resource group becomes active, DPS composes its higher-precedence binding with the topology binding by policy type. A resource-group policy can supersede or combine with topology policy according to the rules of that type.

An empty resource-group bundle selection resolves to the current `global_policy_bundle` and the resolved name is persisted on the group. It does not categorically mean "inherit the topology bundle." If no global default is configured and the resource group contributes no policy of a given type, the topology policy of that type can remain effective.

Create a group with an explicit bundle by using `--policy-bundle`:

```
dpsctl resource-group create \
  --resource-group ml-training-job \
  --external-id 12345 \
  --policy-bundle site-balanced
```

Current resource-group policy-bundle selection is supported at creation time. The server does not currently support changing that selection with `dpsctl resource-group update`.

When a resource leaves an active group, DPS restores any remaining topology-effective limit. If no effective limit remains, release uses the device idle policy or the hardware-minimum path as described in *Policy Lifecycle and Fallback*.

2.5.2.3.3.5 Per-GPU Power Limits

The standalone `dpsctl gpu-policy` command sets explicit per-GPU watt values for nodes in active resource groups:

```
dpsctl gpu-policy \
  --node "node001=500,550,600,700,650,700,550,600"
```

Specify one value for every GPU on the node, in GPU index order. DPS validates the GPU count, device ranges, and available node power budget before applying the update. Refer to *dpsctl gpu-policy* for command details.

2.5.2.3.3.6 Shared GPU Budget

Shared GPU mode pools GPU watts across the members of a resource group. DPS enforces the group sum rather than each node's original compute-policy GPU cap. One node can take more GPU power while another takes less, as long as the group stays inside the resource-group GPU budget.

Omitted `shared_gpu_enable` values default to on. Pass `--shared-gpu-enable=false` at create time to opt out. Shared GPU mode can run together with Power Reservation Steering (PRS) when dynamic power management is enabled.

The resource-group GPU budget is a policy-derived ceiling, not the sum of currently allocated GPU caps:

- When the effective policy sets an explicit GPU limit, that GPU target is the ceiling for the member, and the group ceiling is the sum of those targets.
- When it does not, each member contributes the greater of its currently allocated GPU watts and the node policy after reserving CPU and memory.

You can raise allocated GPU watts up to that ceiling. The live allocated sum is not a freeze.

When you update GPU policies on a shared-GPU resource group, DPS compares the post-update group total to that ceiling:

- Nodes in the request contribute their requested per-GPU watts.
- Nodes that are not in the request contribute their currently allocated per-GPU watts.
- If the post-update total exceeds the ceiling, DPS clamps feasible increases and can apply lower limits than you requested.
- If no valid clamp exists, for example when the sum of hardware GPU minima exceeds the remaining budget, DPS rejects the request before it sends device updates.

A request that fits the resource-group GPU budget can still be rejected if the post-clamp aggregate load would violate an ancestor topology constraint. Supported aggregate checks apply to non-compute topology entities such as racks and power distribution network (PDN) devices. Shared-GPU nodes can exceed a per-node GPU policy cap. They still cannot exceed per-GPU hardware min and max limits, and the device or BMC still enforces those limits.

If a topology ancestor is already overloaded, DPS accepts a shared-GPU update when the new load is lower, or when the load is unchanged and the power setpoint is not reduced. An update that increases the overload, or that keeps the same load while lowering the power setpoint, is rejected.

Rejected GPU policy results include a diagnostic that names the violating ancestor, for example `shared-GPU update violates topology constraint at device:rack-a`. A controller response that would commit an illegal load is also rejected, with a diagnostic such as `response violates topology constraints`, and is not written to entity state.

2.5.2.3.3.7 Priority

Resource group `--priority` orders groups when DPS steals power, and when giveback cannot admit a complete restore of every eligible member. It does not request a policy or reserve watts. Omit the flag to store `0`, which is currently the highest precedence. Assign a larger number to groups that can give up power first.

For how to set, inspect, and update priority, and for steal versus giveback order, refer to *Resource Group Priority*.

2.5.2.3.3.8 Further Reading

- *Power Policies*
- *Entities*
- *Topologies*
- *Resource Group Priority*
- *Managing Resource Groups*

2.5.2.3.4 Resource Group Priority

Resource group priority tells DPS which workloads to reduce first when power must be taken back, and which restore tier to prefer if a complete giveback cannot be admitted. Use it to express relative importance across resource groups. Do not use it as a reserved watt budget, a policy, or a guarantee that a group keeps its requested allocation.

2.5.2.3.4.1 Operator Contract

The following table is the public contract for `--priority`.

Item	Operator meaning
Numeric direction	A lower number is higher priority.
Default	If you omit `--priority`, the value is `0`. That is currently the highest possible precedence.
Valid range	An integer from `0` through `2147483647`.
Giveback	DPS first tries one complete restore of all eligible throttled members. Priority tiers are used only when that complete plan cannot be admitted.
Stealing and reduction	Eligible groups are reduced in reverse-priority bands: higher numbers first. Groups that share a number are reduced together.
Duplicate values	Duplicate priorities are valid. Product order among equal-priority groups is unspecified.
What priority does not do	Priority does not request watts, set a policy, define a floor, or start a background rebalance loop.

Warning

Assigning `--priority 1` (or any positive value) ranks that group **below** every group that still uses the default of `0`. If production workloads omit the flag and a batch group sets `10`, the batch group is reduced first. If you intend the opposite, set an explicit higher number on the less important groups, or set `0` only on the groups that must be served first.

2.5.2.3.4.2 When to Set Priority

Set priority when two or more resource groups can compete for the same constrained power, and you care which one is reduced last. Typical cases include:

- A production inference tier that must keep power longer than a research or batch tier during an NvGrid load target.
- Several static resource groups that share a feed, where importance is independent of the policy each group requested.
- Scheduler-created groups that map quality-of-service classes onto relative power importance.

Prefer a different control when the actual need is not ordering:

Need	Control to use
A higher or lower requested allocation	Resource group or entity *power policy*, not a smaller priority number.
Refuse a reduced allocation on *this* group	`--strict-policy` on activate or add.
Prevent taking power from other groups during this activate or add	`--allow-reprovision=false` on activate or add.
Stay off the grid or excursion victim list	Do not rely on the `dpm_enable` list field. Steal eligibility is an effective DPM policy with a specified strategy.

Priority is independent of policy-bundle selection. The bundle supplies the effective power controls; priority orders steal victims, and giveback tiers only after a complete restore cannot be admitted.

2.5.2.3.4.3 Three-Group Example

The following example uses three names so default `0`, an explicit production value, and a lower-importance value are not conflated.

`Node-High`, `Node-Med`, and `Node-Low` are operator-created bundles, used here only to keep the three groups distinguishable. DPS ships `MaxP`, `MaxQ`, and `MaxLPS`, which select power modes rather than priority tiers. Create the bundles with `dpsctl policy bundle upsert` before you run this example, or substitute names that already exist on your server.

```
# production-serving uses the default. Omitting --priority stores 0,
# which is the highest precedence.
dpsctl resource-group create \
  --resource-group production-serving \
  --external-id 1 \
  --policy-bundle MaxP

# interactive-dev is important, but ranks below any group left at 0.
dpsctl resource-group create \
  --resource-group interactive-dev \
  --external-id 2 \
  --policy-bundle MaxP \
  --priority 10
```

(continues on next page)

(continued from previous page)

```
# batch-research is reduced first when power is scarce.
dpsctl resource-group create \
  --resource-group batch-research \
  --external-id 3 \
  --policy-bundle MaxP \
  --priority 20
```

Add entities, then activate. Setting priority at create time only persists the value. Redistribution runs later, when a trigger such as activation with reprovisioning, an NvGrid load target, or excursion mitigation needs to move power.

```
dpsctl resource-group add --resource-group production-serving --entities
↪"node001,node002"
dpsctl resource-group add --resource-group interactive-dev --entities
↪"node003,node004"
dpsctl resource-group add --resource-group batch-research --entities "node005,
↪node006"

dpsctl resource-group activate --resource-group production-serving --sync
dpsctl resource-group activate --resource-group interactive-dev --sync
dpsctl resource-group activate --resource-group batch-research --sync
```

Inspect the stored values with `dpsctl resource-group list`. Confirm that `priority` is `0`, `10`, and `20` respectively, that `policy_bundle` is `MaxP`, and that activation status matches what you expect.

The following table shows steal order. Giveback is not the reverse of this table on the first attempt. DPS first tries to restore every eligible throttled member together. Priority decides restore order only if that complete plan cannot be admitted, and that fallback order is the reverse of steal order: lowest number first.

Event	First	Then	Last
Stealing or grid reduction	`batch-research` (`20`) reduced	`interactive-dev` (`10`)	`production-serving` (`0`)
Fallback giveback	`production-serving` (`0`) restored	`interactive-dev` (`10`)	`batch-research` (`20`)

When a tighter load target or activation steal needs watts, reductions start at the least important eligible band. When spare power returns, eligible throttled groups recover together if the complete plan fits. If it does not, restoration walks the bands in reverse of steal order.

2.5.2.3.4.4 Create, Inspect, and Update

2.5.2.3.4.5 Create

`dpsctl resource-group create --priority` defaults to `0`. The flag is optional.

```
dpsctl resource-group create \
  --resource-group example-low \
  --external-id 10 \
  --priority 100
```

Preconditions: you are authenticated, and the resource group name is unique.

Verify: `dpsctl resource-group list` reports `"priority": 100` for `example-low`. Create does not apply device policies. Add entities and activate before you treat allocation as applied.

2.5.2.3.4.6 List

```
dpsctl resource-group list
```

Use list output to confirm the stored `priority` field, activation status, and `policy_bundle`. The retained `policy_name` field is compatibility output; it is not the selected policy bundle. A successful create or update RPC is not evidence that devices received new limits. The `dpm_enable` field is not the steal-eligibility check.

If the `priority` field is missing from the output of an older `dpsctl` binary, the installed client may predate the flag. Confirm versions before you troubleshoot ordering:

```
dpsctl --version
dpsctl server-version
```

Align the client and server so both report the same version. Refer to the *dpsctl command reference* for `--no-version-check` and related global flags.

2.5.2.3.4.7 Update

`dpsctl resource-group update --priority` sends a priority change only when you pass `--priority`. Omitting the flag leaves the stored value unchanged.

```
dpsctl resource-group update --resource-group interactive-dev --priority 5
dpsctl resource-group list
```

Updating priority does not by itself steal or give back power. The new value is used the next time a giveback or stealing path runs.

2.5.2.3.4.8 Activate and Add Triggers

Activation and adding entities to an *active* group can steal from other groups when there is not enough power and `--allow-reprovision` is true (the default).

```
dpsctl resource-group activate \
  --resource-group production-serving \
  --allow-reprovision \
  --sync
```

Use `--allow-reprovision=false` when the new group must not lower other groups. Use `--strict-policy` when this group's requested allocation must apply exactly; if power is insufficient, the request fails instead of accepting a reduced allocation.

For flag details, refer to *resource-group create*, *update*, *list*, *activate*, and *add*.

2.5.2.3.4.9 Giveback Versus Stealing

Stealing uses reverse-priority bands. Giveback uses the reverse of that band order only after a complete restore plan fails.

2.5.2.3.4.10 Giveback

Giveback runs when power is freed, for example when:

- An NvGrid load target expires or is replaced with a higher limit.
- Excursion mitigation succeeds and headroom returns.
- Resources are removed from a group, or a group is deactivated or deleted.

DPS first tries one complete restore of every eligible throttled member. If that plan fits topology limits, it is dispatched and priority does not pick a winner. If the complete plan cannot be admitted, DPS walks priority tiers from highest priority (lowest number) to lowest and restores one tier at a time. Equal-priority members in a fallback tier are restored together. Priority does not create watts. Giveback can stop with residual unused power when the remaining headroom is smaller than the next valid step.

2.5.2.3.4.11 Stealing and Grid Redistribution

Stealing runs when a request or mitigation needs power that is not free, for example when:

- You activate a group, or add entities to an active group, with `--allow-reprovision` (default true).
- NvGrid applies a tighter load target.
- Excursion mitigation reduces groups that have an effective DPM policy under the excursion.

Eligible victims are grouped by `--priority`. Higher numbers (lower importance) are reduced first. Groups that share a number are reduced together rather than draining one peer completely before touching another. DPS lowers policy or caps within approved floors. Grid and excursion reduction are best effort: partial progress can remain if the target cannot be met without terminating jobs. Resource-group activation stealing is request-scoped: if the activation cannot complete, stolen changes to other groups are rolled back.

Steal eligibility is not the `dpm_enable` field on `dpsctl resource-group list`. A group is a candidate when its **effective** policy assignment includes a DPM policy with a specified strategy. Non-workload groups also need an aggregate power policy.

`--dpm-enable=false` opts the group out of dynamic power management, and as a side effect DPS then selects that group's own policy strictly, the way `--strict-policy` does for a single request. Neither flag keeps the group off a later grid or excursion victim list.

NvGrid load targets still do not terminate jobs. If every eligible group is already at its floor, the target can remain unmet (`compliant=false`). Refer to *NvGrid* and *Excursion Mitigation*.

2.5.2.3.4.12 Duplicate Priorities

You can assign the same number to more than one group. Product behavior requires ordered levels for *different* numbers. It does not promise which equal-priority group is chosen first.

When several groups share a priority during stealing, DPS reduces that band together rather than draining one peer completely before touching another. Giveback fallback tiers restore equal-priority members together after a complete restore plan is rejected. Do not automate against device count, requested watts, external ID, or resource group ID as tie-breaks. Those comparators are not the live steal or giveback path. If you need a deterministic winner across groups, give them different priority numbers.

2.5.2.3.4.13 Floors, Strict Mode, and Best Effort

Priority never defines a floor. The following bounds are independent:

Bound	Meaning
Requested allocation	The effective limits selected from the resource group's policy bundle, plus any standalone per-GPU caps.
Effective-policy floor	The lowest limit allowed by the effective policies while lowering a group during ordinary stealing.
Device or hardware minimum	The hardware lower bound. Grid and excursion paths can escalate to this bound after policy floors are exhausted.
Shared-GPU minimum	Additional lower bound when shared-GPU behavior is enabled on the group.
Strict request	`--strict-policy` applies to the *requesting* group. It fails the request instead of accepting a reduced allocation for that group. It does not stop stealing from other groups when `--allow-reprovision` is true.

When no reducible headroom remains above the applicable floors, that group is skipped as a further victim. Grid mitigation keeps whatever reduction already applied (best effort) and reports the outcome. Activation stealing that cannot satisfy the request rolls victim changes back. Floor-violation exception alerts are not a current operator-facing guarantee.

2.5.2.3.4.14 Observe Outcomes

After a redistribution event, do not stop at the create or update status. Confirm:

1. `dpsctl resource-group list` shows the expected `priority`, `activation_status`, and `policy_bundle`.
2. Activate or add responses (or `dpsctl resource-group status` for async work) show per-node policy apply results.
3. For NvGrid events, `dpsctl nvgrid get-current` and webhook payloads show the active load target and compliance.
4. Device power caps moved in the direction you expect: lower-priority bands dropped first under a tighter target. After the target relaxes, eligible groups may recover together. Higher-priority tiers recover first only when a complete restore cannot be admitted.

2.5.2.3.4.15 Troubleshooting

Important workloads were reduced first. Check list output. Groups that omitted `--priority` are `0` and outrank every positive value. Raise the number on less important groups, or set `0` only on the groups that must win.

Every group is `0`, and order looks arbitrary. That is expected. Duplicate default priority is valid and unordered as a product contract. Assign distinct numbers if you need a defined sequence.

You changed `--priority` and nothing moved. Priority is consulted when giveback or stealing runs. Activate, add with reprovisioning, apply or expire an NvGrid target, or wait for an excursion path. Updating the field alone does not rebalance.

A high-priority group still dropped. Priority orders eligible victims. It does not reserve watts. If the higher-priority group is the only remaining candidate, or every lower-priority group is already at a floor, DPS can still reduce it. Confirm applied limits versus the bundle-derived request, whether the group has an effective DPM strategy, and whether `--strict-policy` was set on the *requesting* operation.

A lower-priority group recovered at the same time as a higher-priority group. That can be correct. Giveback prefers one complete restore of all eligible members. Priority tiers apply only when that complete plan cannot be admitted.

A group with `dpm_enable` false was still reduced. Steal eligibility is the effective DPM policy strategy, not the `dpm_enable` list field. `--dpm-enable=false` opts the group out of dynamic power management and makes its own policy selection strict. It does not remove the group from later grid or excursion victim selection.

Activation failed after stealing from others. Request-scoped stealing rolls back other groups when the activation cannot complete. Inspect the activate response and `dpsctl resource-group status`. Retry with a bundle that requests lower limits, more `--allow-reprovision` headroom, or `--strict-policy` omitted if a reduced allocation is acceptable for the new group.

`--priority` is not in `dpsctl resource-group create --help`. The client is older than the server feature. Upgrade `dpsctl` and confirm `dpsctl --version` matches `dpsctl server-version`.

2.5.2.3.4.16 Further Reading

- *Resource Groups* - Lifecycle and policy hierarchy
- *Managing Resource Groups* - Create, add, activate, and delete workflow
- *NvGrid* - Grid load targets that trigger best-effort reduction and restoration
- *Excursion Mitigation* - Automatic reduction of lower-priority DPM-enabled groups
- *Power Policies* - Requested allocations and policy steps
- *dpsctl resource-group* - Command reference

2.5.2.3.5 Power Compliance

DPS delivers modeled power compliance by construction. Before it changes managed power, it evaluates the *topology*, effective *power policies*, configured limits, and known load. When a change transfers capacity, DPS reduces power first and permits an increase only after it confirms the reduction. If it cannot confirm the reduction, it keeps the existing or conservative reservation.

This reduce-then-increase behavior keeps an in-progress change within the modeled boundary, even when a device response is delayed or incomplete. When you give a workload more power, you also change demand on a shared PDU, busbar, or facility feeder. A qualified meter at that same boundary establishes the physical outcome.

This guide applies the distinction in *Provisioned and Observed Power* to a declared electrical boundary. It explains how DPS keeps the modeled constraint intact while a power change is in flight, and how boundary metering confirms the result.

Important

DPS can establish what it planned and what controllable devices acknowledged. A fresh, qualified meter reading at the declared boundary establishes what the electrical system actually did.

2.5.2.3.5.1 Begin With the Boundary

Start with the facility constraint that you need to protect. The boundary can be a facility feeder, rack PDU input, DC busbar, compute-tray input, or a controllable device. It defines which loads, limits, and measurements belong in the same decision.

Use that boundary consistently. The operating limit, provisioning margin, contributing loads, and meter must describe the same electrical path. A device-level cap cannot, on its own, establish the state of a facility-level boundary. Equipment in one rack can also be supplied by different feeds or busbars, so physical location is not enough.

2.5.2.3.5.2 Build an Honest Capacity Model

With the boundary defined, DPS combines topology, effective policies, configured limits, and static loads to determine modeled capacity. This model turns a facility constraint into a workload decision.

Power policies constrain what controllable devices can request. They do not represent every load behind a tray, rack, or facility boundary. You must model, measure, or reserve every material load at the selected boundary.

Use the `StaticLoad` field on an *entity* for modeled fixed consumption. Treat it as a conservative engineering allowance, not as an observed meter reading. Keep unknown or uncontrolled load reserved until you can measure it at the boundary.

2.5.2.3.5.3 Change Power Without Reusing Capacity

Consider a *resource group* that needs more GPU power while another resource group can give some back. DPS reduces the donating workload before it makes that capacity available to the receiving workload. This order prevents both workloads from using the same modeled capacity during a transition.

If DPS cannot confirm the reduction, it keeps the prior or conservative reservation. The model remains aligned with the state that the control plane can substantiate, even when a device response is delayed or incomplete. *DPS Power Steering Controller* follows the same model when it proposes per-GPU updates, and DPS validates those updates before it applies them to hardware.

2.5.2.3.5.4 Use Metering to See the Physical Outcome

The modeled result is an expectation. Metering tells you what happened at the electrical boundary. For a measurement-compliance claim, your meter must observe the same physical boundary as the configured limit.

Telemetry and observability data help you monitor and optimize devices. They do not automatically establish power at a rack, PDU, or facility boundary. To make a meaningful claim, the meter must identify the boundary, units, time basis, quality, and aggregation. Treat missing, stale, implausible, or unscoped readings as unknown — not as zero power.

In deployments that use Zapp, the Telemetry Provider streams sensor data from datacenter entities into DPS, which uses that real-time power data for excursion detection and mitigation. It complements, rather than replaces, the qualified boundary meter required for a measurement-compliance claim. Refer to *Configure Zapp Telemetry* to configure the integration.

Compare the model and the boundary meter after commissioning and material changes. Differences can reveal unaccounted load, a measurement mismatch, conversion loss, or a changed operating state.

2.5.2.3.5.5 Apply the Right Time Horizon

Your facility can protect the same boundary at more than one timescale. Keep those limits distinct.

- **50 ms** defines a short-duration or peak-power envelope. DPS does not establish a 50 ms measurement-compliance result. If your site needs this claim, use a qualified facility measurement and analysis path at the declared boundary.
- **1 second** is the primary continuous power window. Boundary metering at this timescale establishes whether the continuous operating limit is met.
- When you configure the `simple` *excursion mitigation* strategy, its default telemetry wait after a reduction is **10 seconds**. It gives DPS time to obtain fresh telemetry; it is not an allowable duration above the 50 ms or 1-second limit.

2.5.2.3.5.6 Define Excursions With the Facility

Do not infer an allowable excursion from a DPS policy, allocation result, or mitigation timer. An allowable excursion is a facility-approved short-duration envelope. The facility design defines its boundary, limit, time window, measurement method, and required response.

DPS does not create a permitted excursion duration or count. Without an explicitly approved envelope, an over-limit result is non-compliant at the applicable 50 ms or 1-second timescale. The 10-second mitigation interval does not change that result.

2.5.2.3.6 Power Policies

Power policies control device power and performance behavior. DPS stores current policies in named, typed **policy bundles** and selects one bundle for an entire topology or resource group. A topology selection is a persisted topology-wide default; it is not an assignment embedded on each entity.

This page describes the current policy model, including what DPS does when no effective policy remains.

2.5.2.3.6.1 Policy Bundles

For an operator, a policy bundle is a named, reusable set of controls selected for a topology or resource group. Each entry in `Policies` has a `Type`, an optional `Scope`, and the settings for that policy type.

2.5.2.3.6.2 Policy Types

Type	Operator intent
`ComputerSystemLimits.v1`	Define node and component operating ranges.
`DPM.v1`	Select Dynamic Power Management strategy and power shifting.
`PRS.v1`	Enable or disable Power Reservation Steering.
`APPM.v1`	Select APPM behavior.
`EPD.v1`	Select EPD behavior.
`AggregatePowerLimits.v1`	Define aggregate limits for complete compute nodes.
`WPPS.v1`	Select the complete workload power profile ID set.

2.5.2.3.6.3 Built-In Bundles

DPS ships three policy bundles:

Bundle	Intended behavior
`MaxP`	Use the full operating ceiling, proportional DPM with power shifting enabled, and PRS disabled.
`MaxQ`	Use model-specific efficiency targets, proportional DPM with power shifting enabled, and PRS disabled.
`MaxLPS`	Use model-specific sustained-power targets, proportional DPM with power shifting enabled, and PRS enabled.

The model-specific limits can differ by system. Inspect the installed bundle before selecting it.

2.5.2.3.6.4 Custom Bundle Example

The following bundle sets an 80 percent operating target within a 100 percent ceiling, enables proportional DPM and power shifting, and disables PRS:

```
Name: site-balanced
Policies:
  - Type: ComputerSystemLimits.v1
    OperatingRangePercent:
      Target: 80
      Ceiling: 100
  - Type: DPM.v1
    Strategy: PROPORTIONAL
    PowerShift: ENABLED
  - Type: PRS.v1
    Value: DISABLED
```

Download `site-balanced.yaml`.

2.5.2.3.6.5 Manage and Select a Bundle

List and inspect available bundles, then store one custom bundle:

```
dpsctl policy bundle list
dpsctl policy bundle get MaxLPS
dpsctl policy bundle upsert site-balanced.yaml
```

`upsert` stores the bundle definition; it does not apply the bundle to hardware. After storing it, choose the selection path that matches the object:

- To use the bundle as the default for a later Go CLI topology import, or for a later resource-group create that omits `--policy-bundle`, update `global_policy_bundle` before creating the object.
- To select the bundle only for a new resource group, pass `--policy-bundle` when creating that group, as shown in *Resource-Group Policy Selection*.

These are alternative selection paths; setting the global default is not required when the resource group names its bundle explicitly. Refer to *bundle* for the bundle command summary.

2.5.2.3.6.6 Verify Bundle State

Verify each layer separately. A successful check at one layer does not prove the next layer:

Question	Go `dpsctl` command	Meaning
Is the definition stored?	`dpsctl policy bundle list` or `dpsctl policy bundle get <name>`	Confirms that the server stores the named bundle definition. It does not confirm selection or allocation.
Which bundle is selected?	`dpsctl topology list` or `dpsctl resource-group list`	Inspect `policy_bundle` to confirm the name persisted on that topology or resource group.
Which policy object is allocated to a target node?	`dpsctl check status --nodes <node>`	Inspect `current_state.allocated_policy_name`. The value identifies an allocated policy object or type, such as `ComputerSystemLimits.v1`; it is not the bundle name.
Which power limits are enforced by the device?	`dpsctl check metrics --nodes <node> --effective`	Inspect the power-limit fields read from the device. This verifies hardware state rather than only stored or allocated state.

Note

The YAML example uses the Go `dpsctl` file format. SDK clients construct the corresponding typed protobuf policies rather than sending this YAML as an API payload.

2.5.2.3.6.7 Topology Policy Selection

A **topology policy** is the named policy bundle persisted on a topology. It applies topology-wide when the topology is active. API clients set the selection with `CreateUpdateTopologyRequest.policy_bundle`.

When the topology selection is empty, DPS resolves it to the current `global_policy_bundle` setting and persists the resolved name on the topology. The same empty-value resolution applies to resource-group policy bundles. New deployments set `global_policy_bundle` to `MaxLPS` by default.

`global_policy_bundle` is a default-selection setting, not a global binding. Changing it affects later topology operations that provide an empty bundle value and later resource-group creation that omits the bundle. It does not rewrite selections already persisted on topologies or resource groups, and it does not immediately change running hardware.

Administrators can change the default selection:

```
dpsctl settings update --set global_policy_bundle=site-balanced
```

Released Go `dpsctl settings update` rejects an empty setting value. To remove the global default, use the *Web UI removal workflow* or an authenticated settings API client. A topology created with no

resolved bundle can activate without an effective power-limit policy, in which case the unset behavior described in the following sections applies.

Important

The former topology-file policy model is not an activation interface. A topology import with a nonempty top-level `Policies` array is rejected. A legacy per-entity `Policy` value does not create a policy assignment when the topology activates. Manage bundles separately and select a bundle through the current topology or resource-group interface.

2.5.2.3.6.8 Resource-Group Policy Selection

Select a resource group's bundle when you create the group:

```
dpsctl resource-group create \
  --resource-group ml-training-job \
  --external-id 12345 \
  --policy-bundle site-balanced
```

If `--policy-bundle` is omitted, the server resolves and persists the current `global_policy_bundle`.

2.5.2.3.6.9 Policy Composition

While a topology is active, DPS creates a topology-scoped binding from its selected bundle. An active resource group creates a higher-precedence binding. The composition rule is specific to each policy type: resource-group policies supersede or compose with topology policies by policy type.

The effective policy applies to matching power-managed devices, not to a separate per-entity policy selection stored in the topology definition.

2.5.2.3.6.10 Policy Lifecycle and Fallback

The following table distinguishes persisted selection from hardware action. Creating or importing a topology stores a selection; activation and release events are what apply policies to devices.

Situation	Core Action	Documented Result
Topology is created or imported	No hardware action	Resolve and persist the explicit topology bundle or the current global default.
Topology activates with an effective policy set	`SetPolicy`	Apply the topology-wide effective policy to matching power-managed devices.
Topology activates without an effective policy	`UnsetPolicy`	Apply `idlePolicy` when configured; otherwise use the plugin-specific unset fallback.
Resource group becomes active	Compose the higher-precedence resource-group binding	Resource-group policies supersede or compose with topology policies by policy type.
Resource group is released with remaining topology limits	`SetPolicy`	Restore the topology-effective policy.
Resource group is released without limits but with `idlePolicy`	`UnsetPolicy`	Apply the device idle policy.
Resource group is released without limits or `idlePolicy`	`SetPolicy` at the hardware minimum	Do not interpret this path as a return to BMC defaults.
Topology deactivates	`UnsetPolicy`	Mark the topology inactive and perform best-effort device cleanup through idle or plugin fallback behavior.

2.5.2.3.6.11 Device Idle Policies

An `idlePolicy` is an unnamed fallback in a `ComputerSystem` device specification. It is not a topology policy and is not a policy bundle. DPS passes it to the device plugin when an unset operation should place the device into its idle state.

`ComputerSystemSpec.idlePolicy` supports `Node`, `GPU`, `CPU`, and `Memory` entries. Set exactly one `watts` or `percentage` value in each entry. The shipped GB200 specification uses this idle policy:

```
idlePolicy:
  Node:
    watts: 1000
  GPU:
    watts: 800
  CPU:
    watts: 100
```

Values must be positive and percentages cannot exceed 100. Node, GPU, and CPU watt values are validated against their device-specification ranges. When all entries use watts and a Node entry is present, the component total cannot exceed the Node value. When all entries use percentages, the component percentages must total 100; if a Node entry is present, it must also be 100. Refer to *Device Specifications* for the role of device specifications.

A component-only idle policy does not imply a node cap. For example, GPU and CPU idle values without a `Node` entry configure those components but do not create a node limit that was not specified.

2.5.2.3.6.12 Plugin-Specific Unset Behavior

`UnsetPolicy` asks the device plugin to perform cleanup; it does not guarantee a single hardware state across every platform.

- Generic Redfish applies `idlePolicy` when one is configured. Without one, it removes the DPS-managed Redfish policy.
- Specialized EnvironmentMetrics plugins can restore platform BMC defaults when no idle policy is available. If applying the idle policy fails, a plugin can restore the BMC defaults and report a warning.

Idle-policy failures therefore vary by plugin and can appear as errors or warnings in device status. Topology deactivation remains best effort: DPS marks the topology inactive and reports cleanup failures rather than reverting it to active.

An empty allocated-policy name in status does not prove that the device is uncapped. An idle policy or a plugin-restored BMC default can still be active even though no named bundle is allocated.

2.5.2.3.6.13 Related Interfaces

- Topology API selection: `CreateUpdateTopologyRequest.policy_bundle`
- Default-selection setting: `global_policy_bundle`
- Resource-group selection: `dpsctl resource-group create --policy-bundle <name>`
- Device-spec fallback: `ComputerSystemSpec.idlePolicy`
- CLI bundle management: *bundle*
- CLI default selection: `dpsctl settings update --set global_policy_bundle=<name>`

For operator workflows, refer to *Managing Topologies*, *Resource Groups*, and *Manage Policy Bundles and Set the Global Default*.

2.5.2.3.7 Workload Power Profiles Settings

2.5.2.3.7.1 Overview

Workload Power Profiles Settings (WPPS) are NVIDIA's pretuned GPU power optimization profiles. DPS provides a mechanism to enable and disable these profiles on supported GPUs through resource groups settings.

WPPS profiles are designed for different performance and power optimization goals (e.g. Max-Q vs Max-P) and automatically configure GPU power settings without requiring manual tuning.

2.5.2.3.7.2 Key Concepts

2.5.2.3.7.3 Profile IDs

WPPS profiles are identified by numeric IDs ranging from **3 to 258**. DPS allows you to:

- **Enable specific profile IDs** for your workloads
- **Enable multiple profiles simultaneously** (GPU firmware handles conflicts automatically)
- **Apply profiles to entire resource groups** for consistent optimization

Important: The power profile values configured via DPS use Out-Of-Band (OOB) management (Redfish APIs on the BMC), which uses a different indexing system than the one used in DCGMI and NVSMI. Specifically, the OOB values are offset by 3:

OOB Value = DCGMI/NVSMI Value + 3

For example, if a profile is set to 1 in DCGMI or NVSMI, it should be set to 4 in OOB management.

2.5.2.3.7.4 Example Usage

```
# Enable single profile
--workload-profile-ids 5

# Enable multiple profiles
--workload-profile-ids 3,7,12
```

2.5.2.3.7.5 Profile States

For reference, each GPU maintains three profile state masks that are managed automatically by WPPS:

- **Supported Profile Mask** - Profiles available on the hardware (read-only)
- **Requested Profile Mask** - Profiles requested by DPS
- **Enforced Profile Mask** - Profiles currently active on the GPU

DPS users don't need to manage these states directly - they are handled automatically by the GPU firmware.

2.5.2.3.7.6 DPS Integration and Operations

2.5.2.3.7.7 Resource Group Management

DPS provides simple commands to manage WPPS through resource groups:

```
# Create resource group with WPPS
dpsctl resource-group create \
  --resource-group "ml-training" \
  --external-id 1 \
  --policy-bundle MaxP \
  --workload-profile-ids 5,7

# Update existing resource group profiles
dpsctl resource-group update \
  --resource-group "ml-training" \
  --workload-profile-ids 3,12

# Remove all profiles
dpsctl resource-group update \
  --resource-group "training-job" \
  --remove-all-workload-profiles

# Delete existing resource with WPPS
dpsctl resource-group delete \
  --resource-group "ml-training"
```

DPS applies the profiles to all GPUs in the resource group and handles any conflicts automatically. When a resource group configured with WPPS is deleted, DPS removes the WPPS profiles it configured.

2.5.2.3.7.8 Device Requirements

GPUs must have `wppsSupport: true` in their device specification to use WPPS.

> **Note**: WPPS is not supported on GPU architectures before Blackwell.

2.5.2.3.7.9 Best Practices

- **Choose appropriate profiles** for your workload type (training vs inference)
- **Test profile combinations** to find optimal performance for your specific use case
- **Monitor job performance** to validate profile effectiveness

2.5.2.3.7.10 Troubleshooting

If WPPS aren't working:

1. **Check device support**: Verify GPU specifications include `wppsSupport: true`
2. **Verify profile IDs**: Ensure IDs are in the valid range (3-258)
3. **Check logs**: Monitor DPS server logs for profile application errors

2.5.2.3.7.11 Further Reading

- *Power Policies* - Static power limit management
- *Resource Groups* - Dynamic workload resource management
- *Device Specifications* - Hardware capability definitions
- *Redfish API* - Low-level hardware communication protocol

2.5.2.3.8 Excursion Mitigation

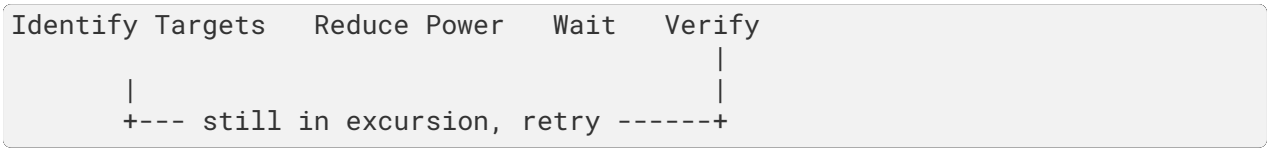
> **Scope:** This document covers DPS-side behavior only. For configuration, refer to the *Deployment Guide*.

2.5.2.3.8.1 Overview

This document describes how DPS receives power telemetry from Zapp, detects power excursions, and automatically mitigates them by reducing power to lower-priority workloads.

The `simple` strategy automatically reduces power to lower-priority resource groups (RGs) when an excursion is detected.

2.5.2.3.8.2 Simple Mitigation Algorithm



2.5.2.3.8.3 Step 1: Identify Targets

DPS determines which resource groups to take power from:

- It identifies the highest-level devices in excursion (e.g., if both a PDU and one of its nodes are in excursion, only the PDU is considered to avoid double-counting).
- It calculates the **power deficit** — how many watts over the limit the system is.
- It sets a **steal target** of `oversteal_factor x deficit` (default: 2x the deficit) to account for the fact that actual power reduction may not exactly match estimates.
- It finds resource groups that have an effective DPM policy with a specified strategy and that have devices under the excursion points. Groups are ordered by `--priority` band. Lower-importance bands (higher numbers) are targeted first. Refer to *Resource Group Priority*.
- If no resource groups can be reduced, the mitigation fails and a critical alert is fired.

2.5.2.3.8.4 Step 2: Reduce Power

DPS reduces power policies on the identified resource groups:

- Starting with the least important `--priority` band, DPS lowers power toward the policy floor.
- Groups that share a priority number are reduced together. A lower-importance band is exhausted before a higher-importance band is touched.
- At each step, DPS estimates the watts saved based on the difference between the old and new limits.
- The process stops when either the steal target is reached or all candidate groups have been lowered as far as possible.
- Power policy changes are sent to the device plugins to take effect.

2.5.2.3.8.5 Step 3: Wait

DPS waits for the configured interval (default 10 seconds) to allow the power changes to take effect and for fresh telemetry readings to arrive.

2.5.2.3.8.6 Step 4: Verify

DPS re-evaluates the excursion using fresh telemetry data:

- If all excursion points are now within their provisioning limits, the mitigation is considered successful. DPS then attempts to **give back** stolen power. Giveback first tries a complete restore of eligible members. Priority tiers apply only if that complete plan cannot be admitted.
- If the excursion persists and the maximum number of iterations has not been reached, the process loops back to Step 1 with updated telemetry.
- If the maximum number of iterations is reached without resolution, the mitigation fails and a `ExcursionMitigationFailed` alert is fired.

2.5.2.3.8.7 Power Give-Back

After a successful mitigation, DPS automatically attempts to restore power to the resource groups that were reduced. This is a best-effort process — whether power can actually be restored depends on the real source of the excursion. If the underlying cause persists (e.g., sustained high workload), there may not be enough headroom to give power back, and the affected RGs will remain at their

reduced policy levels. DPS will do its best to restore as much power as possible given the current conditions.

2.5.2.3.8.8 Alerts

DPS generates two excursion-related alerts via Alertmanager:

2.5.2.3.8.9 PowerExcursion (warning)

Fired for every device where aggregated power exceeds the operational limit.

- **alertname:** `PowerExcursion`
- **severity:** `warning`
- **labels:** `entity` (device ID)
- **summary:** `Power excursion on <entity>: <measured>W measured exceeds <expected>W expected`

2.5.2.3.8.10 ExcursionMitigationFailed (critical)

Fired when the mitigation algorithm has exhausted all options without resolving the excursion.

- **alertname:** `ExcursionMitigationFailed`
- **severity:** `critical`
- **labels:** `entities` (comma-separated device IDs)
- **summary:** `Excursion mitigation failed for <entities> after stealing <watts>W from <count> resource groups`

  Note: Excursion alerts require an Alertmanager instance. Without it, excursions are still logged but no alerts are fired. Refer to the *Deployment Guide* for Alertmanager configuration.

2.5.2.3.8.11 Further Reading

- *Telemetry* - Real-time power data that drives excursion detection
- *Power Policies* - Power configurations adjusted during mitigation
- *Resource Groups* - Workload groupings targeted by excursion mitigation
- *Configure Zapp Telemetry* - Configure Zapp-backed telemetry for excursion workflows
- *Resource Group Priority* - Reduction order across resource groups

2.5.2.3.9 DPS Power Steering Controller

2.5.2.3.9.1 Overview

The DPS Power Steering Controller is the current implementation of Power Reservation Steering (PRS) for DPS. It dynamically redistributes GPU power within the power budget that DPS assigns to an active resource group.

DPS remains the authority for topology, resource groups, power policies, and device safety constraints. The controller proposes per-GPU limits from live telemetry and workload allocation data. DPS validates each proposed limit against the applicable policy and applies only valid limits to the hardware.

2.5.2.3.9.2 Architecture and Control Cycle

Power Steering runs as two services deployed by the `power-steering` Helm subchart:

Service	Port	Responsibility
Config Service	`8000`	Stores Power Steering domains, nodes, devices, and budgets.
Controller Service	`8001`	Runs the control loop that collects data, calculates limits, and requests their application.

When you use the DPS backend, the Controller runs once per configured interval. The Controller redistributes power only within the domain budget and GPU bounds that DPS supplies. It does not replace an effective power policy or create additional budget.

2.5.2.3.9.3 Redistribution Loop

Each control cycle performs the following steps:

1. **Load the current configuration.** The Controller reads the domain, node, and device configuration from the Config Service.
2. **Collect the current state.** It requests device power telemetry and workload allocations from DPS and loads the recent power history.
3. **Estimate demand.** The Controller predicts power demand, identifies under-allocated devices, and detects idle devices that can yield part of their request.
4. **Resolve and distribute target power.** It calculates a target for each GPU, then uses the waterfall allocator to satisfy device minimums before distributing the remaining domain budget across demand.
5. **Verify and apply the result.** Before applying limits, the Controller confirms that the topology did not change during the cycle. It then sends the calculated GPU limits to DPS, which validates and applies them in a compliance-safe order.
6. **Persist and observe.** The Controller records the cycle state and metrics for the next prediction window and for operational monitoring.

If a pipeline step fails, the Controller stops that cycle and tries again at the next scheduled interval. A GPU that has no current power reading retains its last confirmed limit instead of being treated as idle.

2.5.2.3.9.4 Resource Groups and Power Domains

DPS creates one Power Steering domain for each active resource group with resolvable effective policies. The domain contains only that resource group's nodes. When you deactivate the resource group, DPS removes that domain from Power Steering.

An active resource group needs applicable power policies that define node-level power limits for its members. DPS calculates the domain budget from these policies, reserved CPU and memory power, and the configured PRS headroom. If DPS cannot resolve an entity's policy, it excludes that entity from the Controller configuration until the policy becomes available.

The Controller can redistribute only within the resulting budget. It does not override DPS policy or shared-GPU limits.

2.5.2.3.9.5 Deployment and Configuration

Enable both the Power Steering subchart and the DPS integration. The selected DPS chart pins the Power Steering dependency. Use that dependency version when you inspect its database, object storage, and deployment requirements:

```
helm show readme ngc/power-steering --version "<power-steering-version>"
helm show values ngc/power-steering --version "<power-steering-version>"
```

Prepare PostgreSQL and S3-compatible object storage as described by that chart version.

For the DPS backend, the Controller must point to the release-specific DPS gRPC Service.

Refer to *Deployment and Configuration Guide* for the DPS security prerequisites and installation procedure.

```
dps:
  prs:
    enabled: true

power-steering:
  enabled: true
  postgresql:
    enabled: false
  controller:
    cluster:
      dps:
        host: dps-server-grpc
```

The following values control the DPS integration. Refer to the deployment values for the authentication, TLS, storage, and resource settings required by your environment.

Value	Description
`dps.prs.enabled`	Enables the DPS server integration with the Power Steering Config Service.
`dps.prs.hostPort`	Sets the Config Service URL. When empty, DPS derives the in-cluster service URL.
`dps.prs.controllerHostPort`	Sets the Controller Service URL. When empty, DPS derives the in-cluster service URL.
`dps.prs.headroomPercent`	Adjusts the GPU pool that DPS sends to a Power Steering domain. The supported range is `-50` through `100`; the default is `-15`.
`dps.prs.schedulerIntervalSecond`	Changes the Controller schedule. Set `0` to retain the Power Steering default, or use a value of at least 5 seconds.
`dps.prs.historyWindowSize`	Sets the number of per-GPU samples kept for prediction. Set `0` to retain the Power Steering default.

After deployment, the DPS global `prs_enabled` setting controls whether DPS sends Power Steering updates. For example:

```
dpsctl settings update --set prs_enabled=true
```

Note

The legacy `prs.*` Helm values, the `8880` configuration port, the job-scheduler sidecar, and the staged configuration workflow do not apply to the DPS Power Steering Controller. Use the `power-steering.*` subchart values and `dps.prs.*` integration values instead.

2.5.2.3.9.6 Operational Checks

Before you expect Power Steering to change GPU limits, confirm the following:

- `power-steering.enabled` and `dps.prs.enabled` are enabled in the deployed Helm values.
- The DPS global `prs_enabled` setting is enabled.
- The resource group is active and has resolvable effective policies with node-level power limits for its managed entities.
- The Config and Controller Services are ready. For the DPS backend, confirm that the Controller can reach DPS over gRPC.

Power Steering records control-cycle results with `power_steering_` metrics. Use the Controller logs and these metrics to investigate topology, telemetry, allocation, or limit-application failures.

2.5.2.3.9.7 Related Documentation

- *Power Policies*
- *Resource Groups*
- *Managing Resource Groups*
- *`dpsctl settings update`*

2.5.2.3.10 MaxLPS Power Management

MaxLPS asks whether a fixed facility power budget can deliver higher aggregate inference throughput by operating more GPUs inside that budget than static, per-rack sizing would allow. DPS continuously reconciles aggregate draw against the agreed operating envelope.

Inference workloads rarely sustain peak GPU power draw. MaxLPS uses that headroom to operate more compute while keeping aggregate draw within the facility envelope. A common planning rule of thumb is up to 40% more compute than static power sizing would allow. Validate that assumption with your own workload measurements and power-compliance requirements.

Compared with an unmanaged baseline, a successful managed run increases aggregate token throughput or your primary tokens-per-second KPI, maintains an acceptable latency impact, and complies with the operating envelope throughout the steady-state window.

Before you enable DPS power control, complete the *BMC Readiness and Health Guide* for every managed node.

2.5.2.3.10.1 How DPS Enables MaxLPS

DPS combines a power boundary, a managed compute group, and live power control. It does not schedule workloads or add facility capacity. Your workload scheduler continues to place work, while DPS controls the GPU power available to the compute nodes that you place in the pilot resource group.

1. **Topology sets the facility boundary.** The pilot topology models the four racks beneath a `PowerDomain`. Its `OperatingLimit` is the same facility power envelope used for the three-rack baseline. DPS validates power changes against this topology boundary. Refer to *Topologies*.
2. **A resource group identifies the managed compute.** An active resource group contains the pilot's compute nodes and applies their effective power policies. Dynamic power management enforces those policies. Shared GPU pools the policy-derived GPU budget across the group, rather than treating each node as an independent static cap. Refer to *Resource Groups*.
3. **The default `MaxLPS` bundle defines the allowed range.** The pilot selects the `MaxLPS` bundle when it creates the resource group. The bundle supplies model-scoped power limits that DPS uses for dynamic power management and Power Steering. Do not replace it with a pilot-specific bundle. Refer to *Power Policies*.
4. **Power Steering adjusts the split of that budget.** The controller uses live telemetry and workload allocation data to propose per-GPU limits. DPS validates each proposal against effective policy, topology constraints, the shared-GPU budget, and hardware limits before applying it. Refer to *DPS Power Steering Controller*.

The result is dynamic allocation within a fixed budget: GPUs with demand can receive more of the resource group's available power while other GPUs receive less. Neither Power Steering nor a resource group can exceed the topology envelope or the policy and hardware limits that DPS enforces.

2.5.2.3.10.2 Scheduler-Independent Configuration

Workload schedulers, such as Slurm and Kubernetes, own workload placement. Provisioning systems own fleet membership and hardware lifecycle. DPS can integrate with those systems through the *`nvidia.dcpower.v1` resource group API* and matching *`dpsctl resource-group`* commands. Refer to *Managing Resource Groups* for that integrated lifecycle.

The MaxLPS pilot runbook uses a scheduler-independent configuration. You manage the resource group outside the workload scheduler and provisioning system, while inference continues through your existing tooling. This configuration uses:

- A topology that models the four racks and the operating power domain.
- A resource group containing every compute node, with these capabilities:
  - **Dynamic power management**, which enforces per-node power policy.
  - **Power Steering**, which redistributes headroom based on live telemetry. Refer to *DPS Power Steering Controller*.
  - **Shared GPU**, which enforces the GPU budget as a group sum rather than separate per-node limits. Refer to *Shared GPU Budget*.

All three capabilities are required for the pilot configuration. If one is missing, the managed comparison is not reproducible.

2.5.2.3.11 Provisioned and Observed Power

Two different numbers describe the power of the same set of systems, and Dynamic Power Software (DPS) depends on the difference between them:

- **Provisioned power** is what you have committed. It is the sum of the power limits currently in force across the systems in scope — the draw those systems are permitted to reach.
- **Observed power** is what they are actually drawing, measured through telemetry.

Observed power is almost always lower than provisioned power. The gap between them is the headroom that conventional capacity planning has to leave stranded, and closing it safely is what DPS does.

2.5.2.3.11.1 Why the gap exists

Power limits are set for the worst case. A node provisioned at its nameplate TDP is permitted to draw that much, but a real workload rarely holds every component at its ceiling for a sustained period. Inference traffic in particular is bursty: average draw sits well below peak, and the peaks of different nodes do not coincide.

Size a deployment by adding up nameplate limits and you provision for a simultaneous worst case that never arrives. The unused difference is real capacity — power your facility has paid for, that no workload is using.

2.5.2.3.11.2 Why it matters

Three quantities bound a deployment:

Quantity	What it describes
Capacity	The physical limit of the power distribution equipment
Provisioned power	The total of the limits you have committed to systems
Observed power	What those systems actually draw

Static sizing must keep provisioned power below capacity, because nothing prevents systems from reaching their limits at once. That forces provisioned power down, and with it the amount of compute you can install.

DPS enforces limits dynamically and reacts when draw approaches a constraint. That changes which quantity has to stay under capacity: **observed** power does, rather than provisioned power. You can provision closer to the envelope — install more compute in the same facility — because the system will hold actual draw within the constraint rather than relying on a static sum to guarantee it.

This is why a DPS deployment can appear to be "over-provisioned" on paper and still be operating correctly. Provisioned power above what naive sizing would allow is the intended outcome, not a fault, as long as observed power stays within the binding constraint.

2.5.2.3.11.3 Reading the two numbers

Two ratios are worth watching, and they answer different questions.

Provisioned-load coverage — provisioned power as a percentage of physical capacity — tells you how much of the equipment's capability you have committed.

Observed-versus-provisioned pressure — observed power as a percentage of provisioned power — tells you how much of what you committed is being used. A value above 100 means measured draw has exceeded the provisioned load.

A persistently large gap between the two under load is a signal, not a comfort. It can mean the fleet is under-utilized, that limits are binding unevenly across nodes, or that a workload is not exercising the hardware the way you assumed.

2.5.2.3.11.4 Where each number comes from

Number	Metric
Physical capacity	`topology_service_device_power_capacity_watts`
Provisioned load	`topology_service_device_instance_power_load_watts`
Observed power	`device_total_power_watts`

Refer to *Telemetry and Observability* for the queries that build the coverage and pressure ratios, and for the aggregation caveat that matters here: query a `PowerDomain` directly rather than summing parent and child device types, or topology aggregation counts the same power more than once.

DPS Sample Dashboards covers the Grafana dashboards that plot observed-versus-provisioned deviation over time.

2.5.2.3.11.5 Measuring the difference in a pilot

Run the MaxLPS Power Pilot uses these quantities as the basis of its comparison, and defines the derived figures a pilot reports:

- **Capacity efficiency** — the power budget minus provisioned power, the headroom between the envelope and what you committed on paper.
- **Provisioning efficiency** — provisioned power minus observed power, the stranded headroom that does not appear as draw.

A successful pilot shows capacity efficiency shrinking relative to conservative planning — you deliberately provisioned closer to the envelope — while compliance stays clean, meaning observed power never exceeded the binding constraint.

2.5.2.3.12 Device Specifications

2.5.2.3.12.1 Overview

Device specifications define power ranges, component composition, and management capabilities for equipment models. Datacenter entities reference a specification by device type and model, so the required specifications must be available before entities and topologies are created.

DPS ships a standard registry. Administrators can add a model or override a standard model with `dpsctl device upsert`.

2.5.2.3.12.2 Device Types

2.5.2.3.12.3 Compute Devices

A `ComputerSystem` specification describes a complete server or node. The shipped GB200 specification includes its component inventory, hardware range, and idle fallback:

Shipped DGX_GB200 specification

```
- type: ComputerSystem
  description: NVIDIA GB200 Compute Tray (Bianca)
```

(continues on next page)

(continued from previous page)

```
  model: DGX_GB200
  spec:
    devices:
      - type: CPU
        model: Grace
        count: 2
      - type: GPU
        model: GB200
        count: 4
    minLoadWatts: 1000
    maxLoadWatts: 6667
    processorModulesCount: 2
    idlePolicy:
      Node:
        watts: 1000
      GPU:
        watts: 800
      CPU:
        watts: 100
```

Component specifications describe individual device ranges and capabilities:

```
- type: GPU
  description: NVIDIA GB200 GPU
  model: GB200
  spec:
    minLoadWatts: 200
    maxLoadWatts: 1200
    wppsSupport: true
```

```
- type: CPU
  description: NVIDIA Grace CPU
  model: Grace
  spec:
    minLoadWatts: 50
    maxLoadWatts: 420
```

2.5.2.3.12.4 Power Distribution Devices

A `PowerDomain` is a top-level logical power source:

```
- type: PowerDomain
  description: Generic Power Domain 95 Pct Eff
  spec:
    efficiencyFactor: 0.95
```

`PowerDistribution` and `PowerSupply` specifications describe distribution and conversion equipment:

```
- type: PowerDistribution
  description: Generic Rack PDU at 57500W 95 Pct Eff
  model: RackPDU95_57500W
```

(continues on next page)

(continued from previous page)

```
  spec:
    maxLoadWatts: 57500
    efficiencyFactor: 0.95
```

2.5.2.3.12.5 Core Properties

- `minLoadWatts`: The supported lower power bound.
- `maxLoadWatts`: The supported upper power bound.
- `efficiencyFactor`: Power-conversion efficiency, accounting for internal losses. It applies independently of the topology's `OperatingLimit.PowerFactor`. Refer to *Topology and Device Efficiency Factors*.
- `devices`: Components contained in a composite device.
- `wppsSupport`: Workload Power Profile Settings support.
- `processorModulesCount`: Processor-module count for supported systems.

2.5.2.3.12.6 Computer System Idle Policy

`ComputerSystemSpec.idlePolicy` is an unnamed, device-specific fallback used during policy unset behavior. It is not the topology policy and is not a named policy bundle.

An idle policy can contain `Node`, `GPU`, `CPU`, and `Memory` entries. Each entry defines exactly one positive `watts` or `percentage` value; percentages cannot exceed 100. DPS validates the idle policy before converting it for a device plugin.

A specification does not need to provide every component entry. In particular, component-only entries do not imply a node cap. Platform plugins determine the final unset behavior when no idle policy exists or idle application fails. Refer to *Power Policies* for the lifecycle and platform-specific fallbacks.

2.5.2.3.12.7 Usage

Import specifications and list the registry with `dpsctl`:

```
dpsctl device upsert devices.yaml
dpsctl device list
```

Entities then reference a specification by type and model:

```
{
  "Type": "ComputerSystem",
  "Model": "DGX_GB200",
  "Name": "node001"
}
```

2.5.2.3.12.8 Further Reading

- *Entities*
- *Topologies*
- *Power Policies*

2.5.2.3.13 Telemetry and Observability

DPS publishes operational metrics in Prometheus text format for power, topology, allocation, device, and service state. Use the metrics to observe the control plane, investigate power behavior, and build dashboards and alerts.

This page covers the DPS metrics service and its metric reference. The *DPS Power Steering Controller* publishes a separate endpoint when it is deployed. Refer to *DPS Sample Dashboards* for ready-to-import dashboard examples.

2.5.2.3.13.1 Configure and Access Metrics

Use *Prometheus Monitoring* to expose and scrape the DPS metrics service. That guide owns Helm `ServiceMonitor` configuration, external Prometheus access, and metrics authentication.

The chart exposes `/metrics` through the dedicated `<release>-server-metrics` Service on port `9090` (service port name `dps-metrics`). The direct listener uses HTTP and is separate from the HTTP API Service. Inspect a release named `dps` locally:

```
kubectl -n dps port-forward service/dps-server-metrics 9090:9090
curl -s http://127.0.0.1:9090/metrics
```

Use the generated Service name when the Helm release has a different name. If metrics authentication is enabled, use an OIDC Bearer token with the `dps:metrics:read` scope as described in *Prometheus Monitoring*.

Metric labels can include operational identifiers such as hostnames, device IDs, and resource-group names. Restrict endpoint access to the monitoring identity.

2.5.2.3.13.2 Metric Conventions

The reference lists the metric family names exposed by Prometheus. Counter metric families end in `_total`, and histograms use `_bucket`, `_sum`, and `_count`. For example, query `redfish_request_duration_milliseconds_bucket` for a Redfish duration histogram. Dots in OpenTelemetry attribute names become underscores.

A series can be absent until its collector runs or its related operation occurs. Do not treat absent device or node telemetry as zero; it can indicate an inactive topology, a disabled collector, or a missing source reading.

2.5.2.3.13.3 Zapp-Backed Telemetry and Excursion Detection

When Zapp-backed telemetry is enabled and configured, DPS receives real-time power readings from Zapp for excursion detection and automated mitigation. Without Zapp-backed telemetry, DPS continues to operate, but features that require live power readings do not receive them.

Note

For setup details, refer to *Configure Zapp Telemetry*.

Zapp provides a sensor manifest. DPS binds manifest sensor IDs to the configured host set, subscribes to matching sensors, aggregates readings through the topology tree, and compares each device's power to its configured operational limit.

DPS keeps Zapp-backed telemetry current as hosts and streams change. It resubscribes when the configured host set changes.

2.5.2.3.13.4 DPS Metric Reference

Most metric families appear only after DPS records related data.

2.5.2.3.13.5 Exporter Metadata

Metric family	Type	Unit	Attributes	Usage and availability
target_i	Gaug	None	service_name, try_sdk_language, try_sdk_name, try_sdk_version teleme- teleme- teleme-	Shows the service and OpenTelemetry SDK that produced the metrics. The value is 1. A deployment can add more resource labels.

2.5.2.3.13.6 gRPC API

Metric family	Type	Unit	Attributes	Usage and availability
grpc_reque	Coun	None	component, operation, api_family, grpc_service, grpc_method, response_code, grpc_status_code, status, status_class, source_component (conditional), source_direction (conditional), source_confidence (conditional)	Shows the request rate and result for each API and method. Source attributes appear only when DPS can determine the request source.
grpc_durat	His- togra	ms	component, operation, api_family, grpc_service, grpc_method, response_code, grpc_status_code, status, status_class, source_component (conditional), source_direction (conditional), source_confidence (conditional)	Shows request latency for each API, method, and result. Source attributes appear only when DPS can determine the request source.

2.5.2.3.13.7 Topology and Resource Groups

API metrics appear after DPS runs an operation. The periodic `topology_service_*` metrics start after topology activation.

Metric family	Type	Unit	Attributes
topology_api_activation_duration_milliseconds	Histogram	ms	topology_na
topology_api_deactivation_duration_milliseconds	Histogram	ms	topology_na
topology_entity_hostname	Gauge	None	entity_name
topology_coordination_duration_milliseconds	Histogram	ms	operation
topology_deactivation_coordination_duration_milliseconds	Histogram	ms	operation
resource_group_coordination_duration_milliseconds	Histogram	ms	operation, s
resource_group_deactivation_coordination_duration_milliseconds	Histogram	ms	operation
resource_group_update_coordination_duration_milliseconds	Histogram	ms	operation
resource_group_api_create_duration_milliseconds	Histogram	ms	resource_g
resource_group_api_add_resources_duration_milliseconds	Histogram	ms	resource_g
resource_group_api_activate_duration_milliseconds	Histogram	ms	resource_g
resource_group_api_update_duration_milliseconds	Histogram	ms	resource_g
resource_group_api_remove_resources_duration_milliseconds	Histogram	ms	resource_g
resource_group_api_delete_duration_milliseconds	Histogram	ms	resource_g
resource_group_count	Gauge	None	resource_g
resource_group_devices	Gauge	None	resource_g
topology_service_device_count	Gauge	None	device_type
topology_service_active_device_count	Gauge	None	None
topology_service_active_status	Gauge	None	topology_na
topology_service_resource_group_active_count	Gauge	None	None
topology_service_resource_group_active_dpm_enabled_count	Gauge	None	None
topology_service_resource_group_active_prs_enabled_count	Gauge	None	None
topology_service_resource_group_device_count	Gauge	None	None
topology_service_policy_count	Gauge	None	None
topology_service_total_allocated_power_watts	Gauge	W	None
topology_service_total_capacity_watts	Gauge	W	None
topology_service_device_power_capacity_watts	Gauge	W	device_type
topology_service_device_power_load_watts	Gauge	W	device_type
topology_service_device_instance_info	Gauge	None	device_id, d
topology_service_device_instance_power_capacity_watts	Gauge	W	device_id, d
topology_service_device_instance_power_load_watts	Gauge	W	device_id, d
topology_service_active_resource_group_info	Gauge	None	resource_g
topology_service_active_resource_group_priority	Gauge	None	resource_g

Metric family	Type	Unit	Attributes
topology_service_desired_allocation_info	Gauge	None	allocation
topology_service_current_allocation_info	Gauge	None	allocation
topology_service_pending_tasks_count	Gauge	None	None
topology_service_oldest_pending_task_age_seconds	Gauge	s	None

2.5.2.3.13.8 Reconciler and Control-Loop Phases

Metric family	Type	Unit	Attributes	Usage and availability
reconciler_queue_size	Gaug	None	device_type	Records a reconciler's queue size by device type. Reconcilers of the same type share one series.
reconciler_state	Gaug	None	device_type, state	Records a reconciler's state by device type. The current state is 1; other states are 0. Reconcilers of the same type share one series.
reconciler_errors_tot	Coun	None	kind, device_type	Counts errors by type for each device type.
reconciler_request_du	His-togra	s	device_type, request_type	Shows how long each type of device request takes, including response waits and retries.
reconciler_events_tot	Coun	None	event, request_type, device_type (conditional), reason (conditional)	Counts queued requests, sent requests, responses, retries, timeouts, removals, merges, and ignored responses. Ignored GPU policy responses use reason instead of device_type.
devctrl_phase_durati	His-togra	ms	component, operation, phase, request_type (conditional), response_type (conditional), result (conditional), error_class (conditional), device_found (conditional), plugin_found (conditional), source (conditional), has_latest_ping_time (conditional), has_ping_error (conditional)	Shows device-controller request phase duration and result.
power_give_back_	His-togra	ms	component, operation, phase, result, error_class	Shows the total power giveback time and the result. The phase value is total.

2.5.2.3.13.9 Device Power and NvGrid

These metrics appear only when the node or NvGrid collector is enabled and data is available. Component labels depend on the data that each component reports. DPS removes component series for a node that stops returning metrics instead of preserving a stale sample.

Metric family	Type	Unit	Attributes	Usage and availability
de-vice_total_powe	Gaug	W	device_id, device_type, topology_name, power_domain, de-vice_parent	Shows measured power for each device and its location in the topology.
node_component_p	Gaug	W	node, component, id, gpu_uuid (GPU series only)	Shows how much power each node component uses.
node_component_e	Gaug	None	node, component, id, gpu_uuid (GPU series only), unit	Shows the cumulative energy reported by each node component and the reported unit.
node_component_t	Gaug	°C	node, component, id, gpu_uuid (GPU series only), unit (conditional)	Shows the temperature of each component. unit also appears when the same component reports an energy reading in that collection cycle.
node_component_p	Gaug	W	node, component, id, gpu_uuid (GPU series only), unit (conditional), limit_type	Shows the set, minimum, and maximum power limits for each component. unit also appears when the component reports an energy reading in the same collection cycle.
nvgrid_feed_sche	Gaug	W	feed_tag, start_time, end_time, load_unit, sta-tus	Shows load targets from the previous 24 hours through the next seven days.
nvgrid_system_st	Gaug	W	feed_tag, load_unit	Shows the current scheduled or default load target for each feed.
nvgrid_system_st	Gaug	W	feed_tag, load_unit	Shows the calculated load for each feed. Compare it with the current target.
nvgrid_system_st	Gaug	None	feed_tag	Shows whether a power event is in progress for each feed. A value of 1 means in progress, and 0 means not in progress.

2.5.2.3.13.10 Redfish and Policy

Metric family	Type	Unit	Attributes	Usage and availability
red-fish_request_durati	His-togra	ms	component, operation, redfish_method, redfish_route, red-fish_status_class, error_class, red-fish_retry_outcome	Shows request time by normalized Redfish route, HTTP status group, retry result, and error type.
red-fish_open_connectio	Gaug	None	bmc, pool, target_kind	Shows the current number of Redfish connections for each BMC, connection pool, and direct or proxy target.
red-fish_open_connectio	Gaug	None	bmc, pool, target_kind	Shows the highest observed open connection count. The value resets when DPS clears metrics for the target.
environ-ment_metrics_set_po	His-togra	ms	node, device_count, status	Shows how long DPS takes to apply a power limit and whether the request succeeds, grouped by node and device count.
environ-ment_metrics_get_po	His-togra	ms	node, device_count	Shows how long DPS takes to get default power settings, grouped by node and device count.
environ-ment_metrics_get_re	His-togra	ms	node, device_count	Shows how long DPS takes to get environment readings, grouped by node and device count.
environ-ment_metrics_get_ch	His-togra	ms	node	Shows how long DPS takes to get chassis readings from each node.
pol-icy_apply_duration_	His-togra	ms	node, operation, policy_count, do-main_max_capability, domain_min_capability	Shows how long DPS takes to apply policies, grouped by node, policy count, and minimum and maximum domain capability.
pol-icy_delete_duration	His-togra	ms	node, operation, tar-get_domain_name	Shows how long DPS takes to remove policies from each target domain.
pol-icy_domain_create_d	His-togra	ms	node, operation, pol-icy_count	Shows how long DPS takes to create a policy domain, grouped by node and policy count.
pol-icy_domain_update_d	His-togra	ms	node, operation, pol-icy_count	Shows how long DPS takes to update a policy domain, grouped by node and policy count.

2.5.2.3.13.11 Authentication

These metrics never include usernames, tokens, or client addresses.

Metric family	Type	Unit	Attributes	Usage and availability
auth_failed_attempts_to	Counter	None	reason	Counts failed authentication attempts by reason.
auth_rate_limited_reque	Counter	None	limiter	Counts requests rejected by the IP or username rate limiter.
auth_progressive_delay_	Histogram	s	limiter	Shows the delays added by the IP and username rate limiters.

2.5.2.3.13.12 Database Pool

These metrics appear when the database collector is enabled and a database pool is available.

Metric family	Type	Unit	Attributes	Usage and availability
db_pool_total_conns	Gauge	None	None	Shows the total number of open database connections.
db_pool_acquired_conns	Gauge	None	None	Shows the number of database connections currently in use.
db_pool_idle_conns	Gauge	None	None	Shows the number of database connections ready for use.
db_pool_constructing_cc	Gauge	None	None	Shows the number of database connections being opened.
db_pool_max_conns	Gauge	None	None	Shows the maximum number of database connections allowed.
db_pool_empty_acquire_c	Counte	None	None	Counts connection requests that found no idle database connection.

2.5.2.3.13.13 BMC Health

Metric family	Type	Unit	Attributes	Usage and availability
bmc_health_runs_total	Counte	1	status	Counts completed, failed, and canceled health checks.
bmc_health_run_duration_mil	Histogram	ms	status	Shows how long each health check takes, grouped by result.
bmc_health_issues_total	Counte	1	code, severity	Counts health-check issues by code and severity.
bmc_health_node_probe_durat	Histogram	ms	phase	Shows how long each phase of a node probe takes.

2.5.2.3.13.14 Tracing

Metric family	Type	Unit	Attributes	Usage and availability
otel_trace_export_error	Counte	None	error_class	Counts OpenTelemetry trace export failures by error type.

2.5.2.3.13.15 Common PromQL Queries

Adapt label values and time ranges to the deployment. These are investigation queries, not alert thresholds. In the Prometheus expression browser, replace `$__rate_interval` with a fixed range such as `5m`.

2.5.2.3.13.16 BMC Request Latency

The `redfish_request_duration_milliseconds` histogram records the end-to-end duration of DPS requests to BMCs. Its `_sum` and `_count` series calculate the mean latency in milliseconds:

```
sum(rate(redfish_request_duration_milliseconds_sum[$__rate_interval]))
/
sum(rate(redfish_request_duration_milliseconds_count[$__rate_interval]))
```

Use histogram buckets to calculate the 95th-percentile latency across the selected BMC requests:

```
histogram_quantile(
  0.95,
  sum by (le) (
    rate(redfish_request_duration_milliseconds_bucket[$__rate_interval])
  )
)
```

To keep a BMC or request route in the result, filter on labels that are present in the local `/metrics`

output and retain those labels in the aggregation. For example, add the BMC and route labels to `sum by (..., le)` when calculating a per-BMC or per-route percentile.

2.5.2.3.13.17 Datacenter Topology Power Utilization

To chart aggregate observed power in watts by topology and PowerDomain:

```
sum by (topology_name, power_domain) (device_total_power_watts)
```

Calculate observed utilization of a selected PowerDomain as its aggregated power divided by physical capacity. Replace `my-topology` and `PDU-A` with the topology and PowerDomain to investigate.

```
100 *
sum(device_total_power_watts{
  topology_name="my-topology",
  device_type="PowerDomain",
  device_id="PDU-A"
})
/
sum(topology_service_device_power_capacity_watts{
  device_type="PowerDomain",
  device_id="PDU-A"
})
```

The result is a percentage of physical capacity. Query the PowerDomain rather than summing its parent and child device types; topology aggregation would otherwise count the same power more than once.

2.5.2.3.13.18 Provisioned-Load Coverage and Power Pressure

Provisioned-load coverage shows how much of the PowerDomain's physical capacity is currently provisioned:

```
100 *
sum(topology_service_device_instance_power_load_watts{
  device_type="PowerDomain",
  device_id="PDU-A"
})
/
sum(topology_service_device_power_capacity_watts{
  device_type="PowerDomain",
  device_id="PDU-A"
})
```

Observed-versus-provisioned pressure shows whether current measured power is above or below the provisioned load. A value above `100` means observed power exceeds the provisioned load.

```
100 *
sum(device_total_power_watts{
  topology_name="my-topology",
  device_type="PowerDomain",
  device_id="PDU-A"
})
/
```

(continues on next page)

(continued from previous page)

```
sum(topology_service_device_instance_power_load_watts{
  device_type="PowerDomain",
  device_id="PDU-A"
})
```

2.5.2.3.13.19 Reconciler Errors

Count reconciler errors over the last 15 minutes by error kind:

```
sum by (kind) (increase(reconciler_errors_total[15m]))
```

For a point-in-time BMC view, use `dpsctl check metrics --nodes <node-name>`. Its `--effective` option reads enforced limits directly from the BMC and adds one Redfish GET per device.

2.5.2.3.13.20 Related Information

- *DPS Power Steering Controller*
- *DPS Sample Dashboards*
- *Prometheus Monitoring*

2.5.2.3.13.21 DPS Power Steering Controller

The DPS Power Steering Controller continuously evaluates device telemetry and allocates GPU power within configured power domains. It is separate from the DPS metrics service and has its own Prometheus endpoint.

Use this page with *Telemetry and Observability*, and use *Prometheus Monitoring* for the scrape infrastructure.

2.5.2.3.13.22 Access the Endpoint

The in-cluster controller service listens on port `8001`. For a release named `dps`, inspect it locally:

```
kubectl -n dps port-forward service/dps-power-controller 8001:8001
curl -sk https://127.0.0.1:8001/metrics
```

Confirm the generated Service name before port-forwarding a release with a different name or override. Scrape this endpoint separately from the `<release>-server-metrics` Service; its controller series do not appear on the DPS endpoint.

2.5.2.3.13.23 Signals

Signal	Purpose
Prometheus metrics	Controller progress, telemetry coverage, budget behavior, and confirmed limits.
Kubernetes logs	Failed controller actions and the affected domain or device.
OpenTelemetry traces	Slow or failed stages in a control-loop iteration.
Telemetry dumps	Offline incident investigation when controller object-store recording is configured.
CPU profiles	Controller CPU diagnosis when profiling is configured.

Set `global.tempo.url` to export traces and `global.pyroscope.url` to export profiles. Both are optional. Limit access to all collected data because it can identify topology and workload activity.

2.5.2.3.13.24 First Signals to Watch

Metric	Why it matters
`power_steering_scheduler_executions_`	Its rate should remain positive while the controller is enabled.
`power_steering_scheduler_errors_tota`	A sustained increase means controller iterations are failing.
`power_steering_cluster_telemetry_res`	A low ratio means too few devices are reporting usable telemetry.
`power_steering_device_under_allocati`	A sustained nonzero sum means the selected devices are budget-bound.
`power_steering_domain_power_budget_w`	An unexpected value change means the domain budget changed.

2.5.2.3.13.25 Controller Metric Catalog

2.5.2.3.13.26 Device Metrics

These gauges are labeled by `domain`, `node`, `device_type`, and `device_index`. The under-allocation metric also has a `node_active` label.

Metric	Meaning
power_steering_device_power_min_watts	Lowest configured device limit.
power_steering_device_power_max_watts	Highest configured device limit.
power_steering_device_power_watts	Measured device draw for the current iteration.
power_steering_device_power_limit_watts	Limit computed and requested by the controller.
power_steering_device_actual_power_limit_wa	Limit confirmed by DPS after the device update.
power_steering_device_predicted_power_watts	Predicted near-term device demand.
power_steering_device_under_allocation	1 for a constrained device; otherwise 0.
power_steering_device_prediction_deviation_	Measured draw minus the prior prediction.

A device series can be absent when the controller receives no source reading. Alert on sustained loss of coverage instead of converting absent values to zero.

2.5.2.3.13.27 Domain, Pipeline, and Scheduler Metrics

Area	Metrics
Domain	power_steering_domain_power_budget_watts, power_steering_domain_static_power_watts, power_steering_domain_nodes_active
Cluster	power_steering_cluster_telemetry_responding_ratio
Pipeline	power_steering_control_operation_duration_seconds, power_steering_control_operation_errors_total, power_steering_control_operation_ru power_steering_control_history_write_errors_total
Sched-uler	power_steering_scheduler_duration_seconds, power_steering_scheduler_executions_tota power_steering_scheduler_errors_total, power_steering_scheduler_skipped_total

Pipeline metrics use the step label. Scheduler skip metrics use the reason label. The steerable budget is the domain budget minus static power.

2.5.2.3.13.28 Investigation Workflow

1. Confirm that power_steering_scheduler_executions_total is increasing.
2. Check power_steering_cluster_telemetry_responding_ratio and the count of device-power series before changing allocation settings.
3. Compare power_steering_device_power_limit_watts and power_steering_device_actual_power_li A persistent difference means a requested update was not confirmed.
4. Sum power_steering_device_under_allocation by domain. A sustained value above zero indicates a budget-bound workload, not a controller stall.

5. Use pipeline and scheduler histograms to find the slow step, then inspect the matching trace and Kubernetes log window.

```
sum by (domain) (power_steering_device_under_allocation)
```

```
histogram_quantile(
  0.95,
  sum by (le) (
    rate(power_steering_scheduler_duration_seconds_bucket[$__rate_interval])
  )
)
```

```
sum by (step) (increase(power_steering_control_operation_errors_total[15m]))
```

When a telemetry dump is configured, capture the incident window before its retention period expires. Preserve the matching metrics and logs with it. A dump supports investigation, but it does not replace device response evidence for an applied limit.

Use the controller panels in *DPS Sample Dashboards* as the first routine drill-down.

The following example shows server availability, API latency, and controller latency and convergence signals together.



2.5.2.3.13.29 DPS Sample Dashboards

DPS provides portable Grafana dashboard JSON files. They do not install Prometheus, Grafana, or a scrape configuration.

First configure collection through *Prometheus Monitoring*. Then import a JSON file in Grafana and select a Prometheus datasource for `${DS_PROMETHEUS}`. The dashboards avoid hard-coded namespaces and cluster addresses.

Note

`dps-sdk-dashboard.json` is a scaffold with a placeholder panel and no metric query. It is not an operational sample dashboard.

2.5.2.3.13.30 Import the Samples

In Grafana, select **Dashboards** > **New** > **Import**, upload a JSON file, and choose the Prometheus datasource for `${DS_PROMETHEUS}`. Preserve the dashboard variables; they scope queries to a topology, domain, resource group, or node. The DPS Server Health Dashboard also requires configured log and trace data sources.

The Datacenter Dashboard uses `marcusolsson-treemap-panel` and `ae3e-plotly-panel` for treemap and grid views. Install these plugins when those panels are required. Basic Prometheus panels do not require the optional alerting datasource.

2.5.2.3.13.31 DPS Server Health Dashboard

`dps-server-trace.json` provides DPS server availability and request-health signals. It includes replica availability, restart occurrence, management API latency and errors, method response and activation timing, and DPS control-loop latency, convergence, terminal outcomes, and tracker resets.

The following example shows the server-health and control-loop view.



2.5.2.3.13.32 Datacenter Dashboard

`dps-sdk-datacenter.json` provides the fleet-level view. It refreshes every 10 seconds and filters by topology and domain.

Area	What it shows	Main metrics
Power states	Capacity, provisioned load, observed power, and active target	`topology_service_device_power_capacity_watts`, `topology_service_device_instance_power_load_wat` `device_total_power_watts`, `nvgrid_system_status_load_target_watts`
Operating KPIs	Target utilization, provisioned-load coverage, pressure, stranded power, active-fleet coverage, and domains above target	Power-state metrics plus `nvgrid_system_status_calculated_load_watts`, `topology_service_device_instance_info`, and `resource_group_devices`
Redfish health	Failed-request count, failure rate, and failures by class and route	`redfish_request_duration_milliseconds_count`
Controller health	DPS and controller latency, convergence, and scheduler errors	DPS control-loop and controller duration and error series
Inventory and grid	Domain ratios, topology entities, resource-group membership, and grid schedules	`nvgrid_feed_schedule_load_target_watts` and topology-service inventory metrics

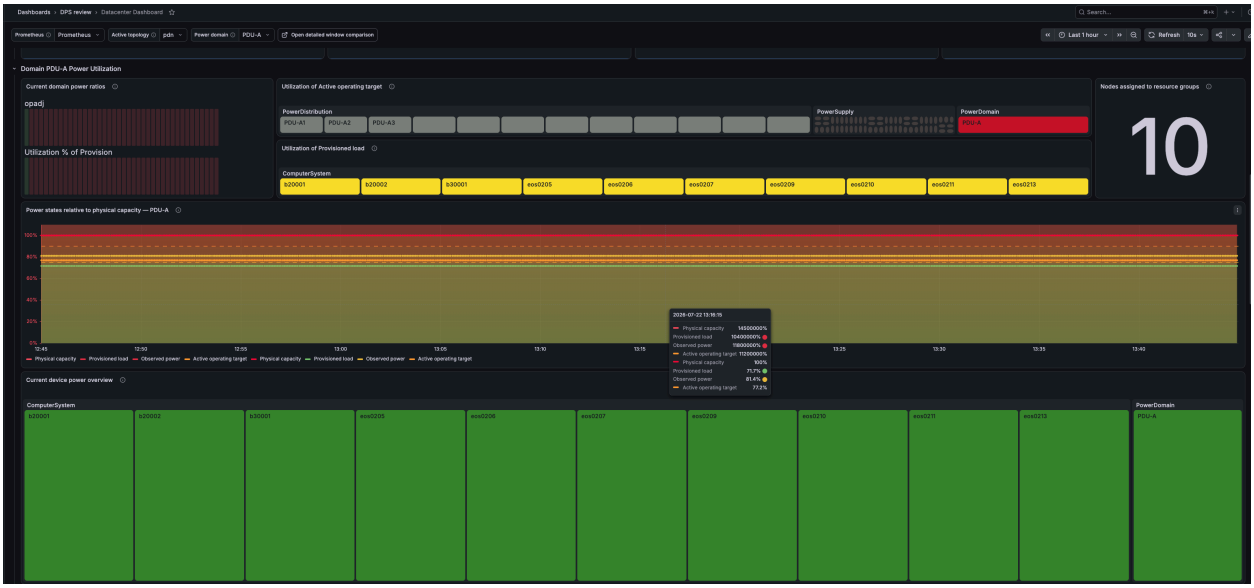
The following example shows the power-state trend, data center KPIs, and controller latency and convergence panels.



The following example focuses on Redfish request health. It shows failed requests, the failure rate, failures grouped by class and route, and the related controller latency and convergence signals.



The following example shows a selected-domain drill-down with power ratios, utilization, resource-group membership, normalized power states, and device power distribution.



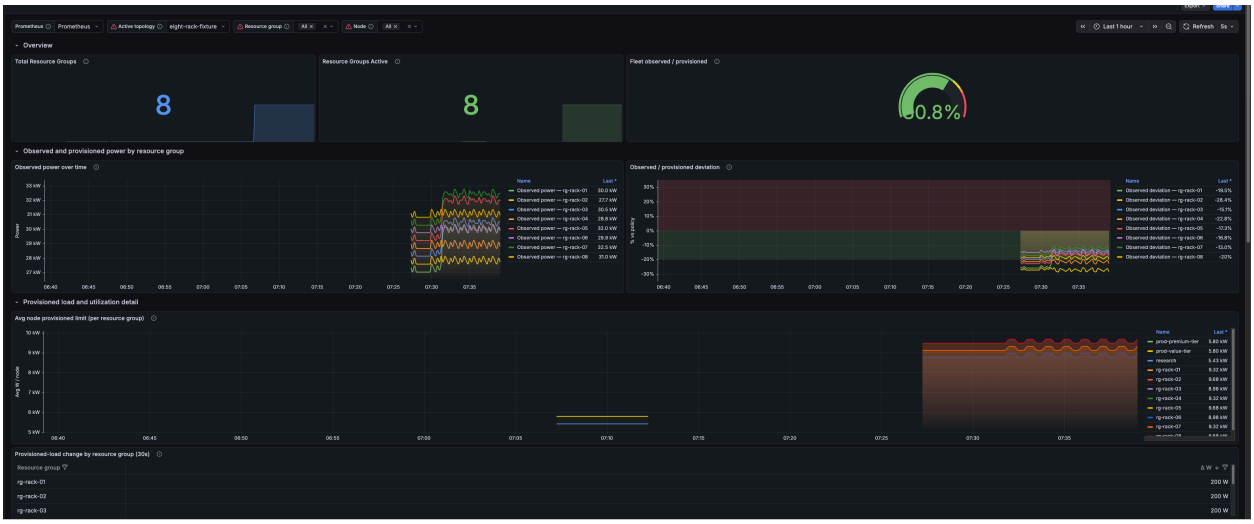
2.5.2.3.13.33 Resource Group Dashboard

`dps-sdk-resource-group.json` is the workload-group view. It refreshes every 10 seconds and filters by topology, resource group, and member node.

Area	What it shows
Summary	Total and active resource groups, membership, and fleet observed-versus-provisioned power.
Power behavior	Observed power, observed-versus-provisioned deviation, average node provisioned limit, and 30-second provisioned-load changes.
Tables	Power and utilization by resource group, group configuration, and nodes above provisioned load.

It combines `resource_group_count`, `topology_service_resource_group_active_count`, `resource_group_devices`, `device_total_power_watts`, and topology-service provisioned-load metrics.

The following example shows the total and active resource-group summary, fleet observed-versus-provisioned power, observed-power and deviation trends, average provisioned limits, and 30-second provisioned-load changes.



2.5.2.3.13.34 MaxP Window Comparison Dashboard

`dps-sdk-maxp-comparison.json` compares an operator-selected Grafana time range with an earlier range. Select a topology, the Grafana time range, `baseline_offset`, and the active-GPU threshold before interpretation.

It compares power envelope, active GPU counts, utilization, headroom, node power, and rack-level changes using `nvgrid_system_status_load_target_watts`, `device_total_power_watts`, `DCGM_FI_DEV_POWER_USAGE`, and `DCGM_FI_DEV_GPU_UTIL`.

GPU panels match GPU hostnames to node metadata from `device_total_power_watts` to identify each GPU's topology and parent rack. DCGM provides the GPU power and utilization measurements. Only GPUs matched to nodes in the selected topology contribute to the comparison.

The envelope uses each root power domain's target: an active grid-schedule target when configured, or the topology's operating limit otherwise. Child rack targets are excluded from the fleet envelope to avoid counting the same budget twice. Utilization and headroom compare GPU power with this target; CPU power and static rack loads are outside the measured GPU power.

For the MaxLPS Pilot sandbox, select **maxlps-pilot** under **Datacenter / topology**. Select **More than 5%** under **Active GPU threshold** to exclude simulated idle activity when comparing the three-rack baseline and four-rack managed workloads.

Active GPU change by rack shows the difference in GPU counts. In the MaxLPS Pilot sandbox, Rack 4 shows an increase of 72 active GPUs when its baseline count is zero. The fleet-level **Active GPU change** shows a percentage.

Warning

This dashboard is an operator-selected diagnostic comparison. It does not create or retain release evidence. Record the time ranges, topology, and workload conditions separately.

The following example shows the diagnostic comparison view with comparison and current power envelopes, active GPU counts, utilization, headroom, and node-power panels.



2.5.2.3.13.35 Optional Alert Rules

`sdk-datacenter-alert-rules.yaml` is an example Grafana alert rule file, not a dashboard-import manifest. It assumes a datasource UID named `prometheus`. Adapt the datasource, labels, routing, and thresholds before provisioning it.

2.5.2.3.13.36 Related Information

- *Telemetry and Observability*
- *DPS Power Steering Controller*
- *Prometheus Monitoring*

2.5.2.3.14 Redfish API

2.5.2.3.14.1 Overview

Redfish is a standard RESTful API specification developed by the Distributed Management Task Force (DMTF) for managing and monitoring server hardware, storage, networking, and converged infrastructure. It provides a standardized way to interact with Baseboard Management Controllers (BMCs) and other hardware management interfaces.

DPS uses Redfish as its primary interface for communicating with NVIDIA DGX compute nodes. Through Redfish, DPS can:

- **Set and monitor power limits** for nodes, GPUs, CPUs, and memory
- **Collect power consumption metrics** and performance telemetry
- **Manage Workload Power Profile Settings (WPPS)** for applicable topology models

2.5.2.3.14.2 Topology Models and Firmware Checks

Use the exact model string from the following table in a topology. These are the Redfish computer-system models implemented by DPS 0.9 and the minimum BMC firmware versions checked by their plugins.

Topology model	Platform family	Power policy interface	Minimum BMC firmware
DGX_H100	DGX H100 and H200	Node Manager (domain-based)	24.0.0
DGX_B200	DGX B200	Node Manager (domain-based)	25.0.0
DGX_B200EM	DGX B200 EM	EnvironmentMetrics (per-component)	25.0.0
DGX_GB200	GB200 NVL	EnvironmentMetrics (per-component)	25.2.0
DGX_B300	DGX B300	Node Manager (domain-based)	46.02.07
DGX_B300EM	DGX B300 EM	EnvironmentMetrics (per-component)	46.02.07
DGX_GB300	GB300 NVL	EnvironmentMetrics (per-component)	25.2.0
DGX_VR_NVL72	Vera Rubin NVL72	EnvironmentMetrics (per-component)	VRB-2601-28.00

Set `dps.auth.firmware_validation_enforce` to `true` to enforce these minima during allocation. Validate the firmware inventory before enabling enforcement. With the default value of `false`, firmware below the minimum is reported as a warning.

2.5.2.3.14.3 Session Management

DPS creates authenticated sessions with BMC for all subsequent API calls. Sessions are maintained as long-lived connections and reused for efficiency.

For detailed session management specifications, see DMTF Redfish Specification DSP0266 v1.23.0 - Section 9.2.4 Session Management.

2.5.2.3.14.4 Create Session

- **Endpoint**: `POST /redfish/v1/SessionService/Sessions`
- **Arguments** (request payload):
  - `UserName` (string) - BMC username from Kubernetes secret
  - `Password` (string) - BMC password from Kubernetes secret
- **Attributes** (response headers):
  - `X-Auth-Token` - Session token for subsequent API requests
  - `Location` - Session URL for deletion

Example:

```
POST /redfish/v1/SessionService/Sessions HTTP/1.1
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "UserName": "admin",
  "Password": "password123"
}
```

```
HTTP/1.1 201 Created
X-Auth-Token: abc123-session-token-xyz789
Location: /redfish/v1/SessionService/Sessions/1
```

Example Error Response (Invalid Credentials):

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{
  "error": {
    "code": "Base.1.13.0.SecurityAccessDenied",
    "message": "While attempting to establish a connection to /redfish/v1/
↪SessionService/Sessions, the service was denied access.",
    "@Message.ExtendedInfo": [
      {
        "@odata.type": "#Message.v1_1_1.Message",
        "MessageId": "Base.1.13.0.SecurityAccessDenied",
        "Message": "While attempting to establish a connection to /redfish/v1/
↪SessionService/Sessions, the service was denied access.",
        "Severity": "Critical",
        "Resolution": "Attempt to ensure that the URI is correct and that the
↪service has the appropriate credentials."
      }
    ]
  }
}
```

2.5.2.3.14.5 Delete Session

- **Endpoint**: `DELETE /redfish/v1/SessionService/Sessions/{session_id}`
- **Path Parameters**:
  - `{session_id}` - Session ID from Location header
- **Request Headers**:
  - `X-Auth-Token` - Session token from create response

Example:

```
DELETE /redfish/v1/SessionService/Sessions/1 HTTP/1.1
X-Auth-Token: abc123-session-token-xyz789
```

```
HTTP/1.1 204 No Content
```

2.5.2.3.14.6 DGX H100 and H200 Systems

Reference: NVIDIA DGX H100/H200 Redfish API Guide

DGX H100 and H200 systems use the Node Manager API for power management through domains and policies. DPS creates a custom domain named `dps-managed-domain` to manage power allocation.

2.5.2.3.14.7 Node Manager Domains

Domains represent power management scopes. DPS manages power through a dedicated domain.

2.5.2.3.14.8 List Domains

- **Endpoint**: `GET /redfish/v1/Managers/BMC/NodeManager/Domains`
- **Attributes** (response):
  - `Members[].@odata.id` - URLs to individual domain resources

Example:

```
GET /redfish/v1/Managers/BMC/NodeManager/Domains HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/Domains",
  "Members": [
    { "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/Domains/0" },
    { "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/Domains/1" }
  ],
  "Members@odata.count": 2
}
```

2.5.2.3.14.9 Get Domain

- **Endpoint**: `GET /redfish/v1/Managers/BMC/NodeManager/Domains/{id}`
- **Attributes** (response):
  - `Id` - Domain identifier for update/delete operations
  - `Name` - Domain name; "`dps-managed-domain`" indicates managed domain
  - `Capabilities.Max` - Maximum power capability in Watts
  - `Capabilities.Min` - Minimum power capability in Watts
  - `Policies.Members[]` - Nested policy objects

Example:

```
GET /redfish/v1/Managers/BMC/NodeManager/Domains/1 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/Domains/1",
  "Id": "1",
  "Name": "dps-managed-domain",
  "Status": { "State": "Enabled" },
  "Capabilities": {
    "Max": 10200,
    "Min": 2500
  },
  "Policies": {
    "Members": [
      {
        "ComponentId": "COMP_GPU",
        "Limit": 5600.0,
        "PercentageOfDomainBudget": 75.0,
        "Status": { "State": "Selected" }
      },
      {
        "ComponentId": "COMP_CPU",
        "Limit": 700.0,
        "PercentageOfDomainBudget": 10.0,
        "Status": { "State": "Selected" }
      }
    ]
  }
}
```

2.5.2.3.14.10 Create Domain

- **Endpoint**: `POST /redfish/v1/Managers/BMC/NodeManager/Domains`
- **Arguments** (request payload):
  - `Id` - Domain identifier (default: `"0"`)
  - `Name` - Always `"dps-managed-domain"` (identifies managed domains)
  - `Status.State` - Always `"Enabled"` (required for active state)
  - `Capabilities.Max` - Node maximum power limit in Watts
  - `Capabilities.Min` - Node minimum power limit in Watts
  - `Policies.Members[]` - Array of policy objects (see *Policy Object Fields*)

Example:

```
POST /redfish/v1/Managers/BMC/NodeManager/Domains HTTP/1.1
Content-Type: application/json

{
  "Id": "1",
  "Name": "dps-managed-domain",
  "Status": { "State": "Enabled" },
  "Capabilities": {
```

(continues on next page)

(continued from previous page)

```
    "Max": 10200,
    "Min": 2500
  },
  "Policies": {
    "Members": [
      {
        "ComponentId": "COMP_GPU",
        "Limit": 5600.0,
        "PercentageOfDomainBudget": 75.0,
        "Status": { "State": "Selected" }
      },
      {
        "ComponentId": "COMP_CPU",
        "Limit": 700.0,
        "PercentageOfDomainBudget": 10.0,
        "Status": { "State": "Selected" }
      },
      {
        "ComponentId": "COMP_MEMORY",
        "Limit": 200.0,
        "PercentageOfDomainBudget": 5.0,
        "Status": { "State": "Selected" }
      }
    ]
  }
}
```

```
HTTP/1.1 204 No Content
```

2.5.2.3.14.11 Update Domain

- **Endpoint**: `PATCH /redfish/v1/Managers/BMC/NodeManager/Domains/{id}`
- **Arguments** (request payload):
  - `Capabilities.Max` - Maximum power limit in Watts
  - `Capabilities.Min` - Minimum power limit in Watts
  - `Policies.Members[]` - Array of policy objects (see *Policy Object Fields*)

Example:

```
PATCH /redfish/v1/Managers/BMC/NodeManager/Domains/1 HTTP/1.1
Content-Type: application/json

{
  "Capabilities": {
    "Max": 9500,
    "Min": 2000
  },
  "Policies": {
    "Members": [
      {
```

(continues on next page)

(continued from previous page)

```
        "ComponentId": "COMP_GPU",
        "Limit": 5000.0,
        "PercentageOfDomainBudget": 70.0,
        "Status": { "State": "Selected" }
      },
      {
        "ComponentId": "COMP_CPU",
        "Limit": 600.0,
        "PercentageOfDomainBudget": 10.0,
        "Status": { "State": "Selected" }
      },
      {
        "ComponentId": "COMP_MEMORY",
        "Limit": 180.0,
        "PercentageOfDomainBudget": 5.0,
        "Status": { "State": "Selected" }
      }
    ]
  }
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed successfully.",
      "MessageId": "Base.1.18.1.Success",
      "Severity": "OK",
      "Resolution": "None."
    }
  ]
}
```

2.5.2.3.14.12 Delete Domain

- **Endpoint**: `DELETE /redfish/v1/Managers/BMC/NodeManager/Domains/{id}`
- **Path Parameters**:
  - `{id}` - Domain identifier

Example:

```
DELETE /redfish/v1/Managers/BMC/NodeManager/Domains/1 HTTP/1.1
```

```
HTTP/1.1 204 No Content
```

2.5.2.3.14.13 Domain Policies

Policies define power limits for specific components within a domain. Each policy object in `Policies.Members[]` contains the following fields:

2.5.2.3.14.14 Policy Object Fields (arguments when creating/updating)

- `ComponentId` (string) - Component type identifier (exact string required):
  - `"COMP_CPU"` - CPU component
  - `"COMP_MEMORY"` - Memory/DRAM component
  - `"COMP_GPU"` - GPU component
- `Limit` (float) - Power limit in Watts
- `PercentageOfDomainBudget` (float) - Percentage of total domain budget (0.0-100.0)
- `Status.State` (string) - Always `"Selected"` (required for active policies)

2.5.2.3.14.15 Get Policy

- **Endpoint**: `GET /redfish/v1/Managers/BMC/NodeManager/Domains/{domain_id}/Policies/{policy_id}`
- **Path Parameters**:
  - `{domain_id}` - Domain identifier
  - `{policy_id}` - Policy identifier
- **Attributes** (response):
  - `ComponentId` - Component type (`"COMP_CPU"`, `"COMP_MEMORY"`, `"COMP_GPU"`)
  - `Limit` - Power limit in Watts
  - `PercentageOfDomainBudget` - Budget percentage allocation

Example:

```
GET /redfish/v1/Managers/BMC/NodeManager/Domains/1/Policies/0 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/Domains/1/Policies/0",
  "Id": "0",
  "ComponentId": "COMP_GPU",
  "Limit": 5600.0,
  "PercentageOfDomainBudget": 75.0,
  "Status": { "State": "Selected" }
}
```

2.5.2.3.14.16 PSU Policies

PSU policies define power limits based on PSU redundancy configuration. DPS validates that requested power caps don't exceed the active PSU policy's `LimitMax`.

2.5.2.3.14.17 List PSU Policies

- **Endpoint**: `GET /redfish/v1/Managers/BMC/NodeManager/PSUPolicies`

Example:

```
GET /redfish/v1/Managers/BMC/NodeManager/PSUPolicies HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/PSUPolicies",
  "Members": [
    { "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/PSUPolicies/0" },
    { "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/PSUPolicies/1" },
    { "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/PSUPolicies/2" }
  ],
  "Members@odata.count": 3
}
```

2.5.2.3.14.18 Get PSU Policy

- **Endpoint**: `GET /redfish/v1/Managers/BMC/NodeManager/PSUPolicies/{id}`
- **Path Parameter Values**:
  - `"0"` - Limp mode (minimal PSU configuration)
  - `"1"` - No Redundancy
  - `"2"` - Full Redundancy
- **Attributes** (response):
  - `LimitMax` - Maximum power limit in Watts; power caps must not exceed this value

Example (Full Redundancy):

```
GET /redfish/v1/Managers/BMC/NodeManager/PSUPolicies/2 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Managers/BMC/NodeManager/PSUPolicies/2",
  "Id": "2",
  "Name": "Full Redundancy",
  "LimitMax": 12000,
  "MaxPSU": 6,
  "MinPSU": 4,
```

(continues on next page)

(continued from previous page)

```
  "Status": { "State": "Selected" }
}
```

2.5.2.3.14.19 Telemetry Service

DGX H100 and H200 use the TelemetryService for real-time power and performance metrics.

2.5.2.3.14.20 Get Metric Reports

- **Endpoint**: `GET /redfish/v1/TelemetryService/MetricReports/NvidiaNMMetrics_0`
- **Attributes** (response):
  - `MetricValues[].MetricId` - Metric identifier (case-sensitive)
  - `MetricValues[].MetricValue` - Metric value as string (e.g., `"8500.0"` not `8500.0`)

Example (partial response):

```
GET /redfish/v1/TelemetryService/MetricReports/NvidiaNMMetrics_0 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/TelemetryService/MetricReports/NvidiaNMMetrics_0",
  "MetricValues": [
    { "MetricId": "dcPlatformPower_avg", "MetricValue": "8500.0" },
    { "MetricId": "AvblNoGPU", "MetricValue": "8" },
    { "MetricId": "AvblNoCPU", "MetricValue": "2" },
    { "MetricId": "gpuPower_avg_0", "MetricValue": "63.0" },
    { "MetricId": "gpuPowerLimit_0", "MetricValue": "700.0" },
    { "MetricId": "gpuPowerCapabilitiesMin_0", "MetricValue": "200.0" },
    { "MetricId": "gpuPowerCapabilitiesMax_0", "MetricValue": "700.0" },
    { "MetricId": "cpuPackagePower_avg_0", "MetricValue": "182.0" },
    { "MetricId": "dramPower_avg_0", "MetricValue": "45.0" }
  ]
}
```

2.5.2.3.14.21 Node-Level Metrics

- `dcPlatformPower_avg` - Total DC platform power in Watts
- `AvblNoGPU` - Available GPU count
- `AvblNoCPU` - Available CPU count

2.5.2.3.14.22 Per-GPU Metrics (for each {id} from 0 to 7)

- `gpuPower_avg_{id}` - GPU average power in Watts
- `gpuPowerLimit_{id}` - GPU power limit in Watts
- `gpuPowerCapabilitiesMin_{id}` - GPU minimum power limit in Watts
- `gpuPowerCapabilitiesMax_{id}` - GPU maximum power limit in Watts

2.5.2.3.14.23 Per-CPU Metrics (for each {id} from 0 to 1)

- `cpuPackagePower_avg_{id}` - CPU average power in Watts
- `cpuPackagePowerLimit1_{id}` - CPU power limit 1 in Watts
- `cpuPackagePowerCapabilitiesMin_{id}` - CPU minimum power in Watts
- `cpuPackagePowerCapabilitiesMax_{id}` - CPU maximum power in Watts
- `cpuEnergy_{id}` - CPU energy in kWh

2.5.2.3.14.24 Per-Memory Metrics (for each {id} from 0 to 1)

- `dramPower_avg_{id}` - DRAM average power in Watts
- `dramPowerLimit_{id}` - DRAM power limit in Watts
- `dramPackagePowerCapabilitiesMin_{id}` - DRAM minimum power in Watts
- `dramPackagePowerCapabilitiesMax_{id}` - DRAM maximum power in Watts
- `dramEnergy_{id}` - DRAM energy in kWh

2.5.2.3.14.25 Firmware Validation

- **Endpoint**: `GET /redfish/v1/UpdateService/FirmwareInventory/HGX_FW_BMC_0`
- **Attributes** (response):
  - `Version` - BMC firmware version; must be >= 24.0.0

Example:

```
GET /redfish/v1/UpdateService/FirmwareInventory/HGX_FW_BMC_0 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/HGX_FW_BMC_0",
  "Id": "HGX_FW_BMC_0",
  "Name": "HGX BMC",
  "Version": "24.08.25.01"
}
```

2.5.2.3.14.26 DGX B200 Systems

Reference: NVIDIA DGX B200 Redfish API Guide

DGX B200 systems expose EnvironmentMetrics for per-component metrics and power limits. The `DGX_B200` topology model applies power policies through Node Manager; `DGX_B200EM` uses EnvironmentMetrics for per-component power limits.

2.5.2.3.14.27 Processors Collection

- **Endpoint**: `GET /redfish/v1/Systems/HGX_Baseboard_0/Processors`
- **Attributes** (response):
  - `Members[].@odata.id` - URLs to individual processors:
    - `GPU_0` through `GPU_7` - GPUs
    - `CPU_0`, `CPU_1` - CPUs

Example:

```
GET /redfish/v1/Systems/HGX_Baseboard_0/Processors HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors",
  "Members": [
    { "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0" },
    { "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_1" },
    { "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_7" },
    { "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/CPU_0" },
    { "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/CPU_1" }
  ],
  "Members@odata.count": 10
}
```

2.5.2.3.14.28 Get Processor

- **Endpoint**: `GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/{processor_id}`
- **Path Parameters**:
  - `{processor_id}` - Processor ID from collection (`GPU_0`-`GPU_7` or `CPU_0`-`CPU_1`)
- **Attributes** (response):
  - `Id` - Processor identifier
  - `Oem.Nvidia.WorkloadPowerProfile.@odata.id` - Presence indicates WPPS support; provides URL to WPPS endpoint

Example (GPU with WPPS support):

```
GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_SXM_1 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0",
  "Id": "GPU_0",
  "Name": "GPU 0",
  "Oem": {
```

(continues on next page)

(continued from previous page)

```
    "Nvidia": {
      "WorkloadPowerProfile": {
        "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/
↪Oem/Nvidia/WorkloadPowerProfile"
      }
    }
  }
}
```

2.5.2.3.14.29 EnvironmentMetrics

2.5.2.3.14.30 Get Environment Metrics

- **Endpoint**: `GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/{processor_id}/EnvironmentMetrics`
- **Path Parameters**:
  - `{processor_id}` - Processor ID (GPU_0-GPU_7 or CPU_0-CPU_1)
- **Attributes** (response):
  - `PowerWatts.Reading` - Current instantaneous power in Watts
  - `PowerLimitWatts.SetPoint` - Current power limit setpoint in Watts
  - `PowerLimitWatts.AllowableMin` - Minimum allowed SetPoint value
  - `PowerLimitWatts.AllowableMax` - Maximum allowed SetPoint value
  - `PowerLimitWatts.DefaultSetPoint` - Factory default setpoint in Watts
  - `EnergyJoules.Reading` - Cumulative energy in Joules
  - `EnergykWh.Reading` - Cumulative energy in kWh
  - `TemperatureCelsius.Reading` - Current temperature in Celsius

Example (GPU):

```
GET /redfish/v1/TelemetryService/MetricReports/HGX_PlatformEnvironmentMetrics_
↪0 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/
↪EnvironmentMetrics",
  "EnergyJoules": { "Reading": 12345.67 },
  "EnergykWh": { "Reading": 1.234 },
  "PowerWatts": { "Reading": 64.342 },
  "TemperatureCelsius": { "Reading": 45.5 },
  "PowerLimitWatts": {
    "Reading": 700.0,
    "SetPoint": 700,
    "AllowableMin": 200,
```

(continues on next page)

(continued from previous page)

```
    "AllowableMax": 1000,
    "DefaultSetPoint": 700
  }
}
```

2.5.2.3.14.31 Set Power Limit

- **Endpoint**: `PATCH /redfish/v1/Systems/HGX_Baseboard_0/Processors/{processor_id}/EnvironmentMetrics`
- **Path Parameters**:
  - `{processor_id}` - Processor ID (GPU_0-GPU_7 or CPU_0-CPU_1)
- **Arguments** (request payload):
  - `PowerLimitWatts.SetPoint` - Power limit in Watts; must be within `AllowableMin` to `AllowableMax` range

Example:

```
PATCH /redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/EnvironmentMetrics
Content-Type: application/json

{
  "PowerLimitWatts": {
    "SetPoint": 700
  }
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed successfully.",
      "MessageId": "Base.1.18.1.Success",
      "Severity": "OK",
      "Resolution": "None."
    }
  ]
}
```

Example Error Response (Out of Range):

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "error": {
    "code": "Base.1.13.0.PropertyValueNotInList",
    "message": "Invalid SetPoint: value 50 is out of range [200, 1000]",
```

(continues on next page)

(continued from previous page)

```
    "@Message.ExtendedInfo": [
      {
        "@odata.type": "#Message.v1_1_1.Message",
        "MessageId": "Base.1.13.0.PropertyValueNotInList",
        "Message": "Invalid SetPoint: value 50 is out of range [200, 1000]",
        "Severity": "Warning",
        "Resolution": "Correct the value for the property in the request body
↪and resubmit the request if the operation failed."
      }
    ]
  }
}
```

2.5.2.3.14.32 Chassis Environment Metrics

For total system power:

- **Endpoint**: `GET /redfish/v1/Chassis/Chassis_0/EnvironmentMetrics`
- **Attributes** (response):
  - `PowerWatts.Reading` - Total chassis power for node-level metrics

Example:

```
GET /redfish/v1/Chassis/Chassis_0/EnvironmentMetrics
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Chassis/Chassis_0/EnvironmentMetrics",
  "PowerWatts": { "Reading": 8500.0 }
}
```

2.5.2.3.14.33 Workload Power Profiles (WPPS)

WPPS allows fine-grained control over GPU power behavior for specific workload types. Profiles are identified by index (0-255) and represented as hex bitmasks.

2.5.2.3.14.34 Get WPPS Configuration

- **Endpoint**: `GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/Oem/Nvidia/WorkloadPowerProfile`
- **Path Parameters**:
  - `{gpu_id}` - GPU ID (GPU_0-GPU_7)
- **Attributes** (response):
  - `SupportedProfileMask` - Hex mask of available profiles on this GPU
  - `RequestedProfileMask` - Hex mask of requested profiles (may differ from enforced during transition)

- `EnforcedProfileMask` - Hex mask of currently active profiles

Example:

```
GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_SXM_1/Oem/Nvidia/
↪WorkloadPowerProfile HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/Oem/
↪Nvidia/WorkloadPowerProfile",
  "SupportedProfileMask": "0x10f98",
  "RequestedProfileMask": "0x0",
  "EnforcedProfileMask": "0x0"
}
```

2.5.2.3.14.35 Enable Workload Profiles

- **Endpoint**: `POST /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/Oem/Nvidia/WorkloadPowerProfile/Actions/NvidiaWorkloadPower.EnableProfiles`
- **Path Parameters**:
  - `{gpu_id}` - GPU ID (GPU_0-GPU_7)
- **Arguments** (request payload):
  - `ProfileMask` - Hex mask of profiles to enable; must use `0x` prefix (e.g., `"0x7"` for profiles 0, 1, 2)

Example:

```
POST /redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_SXM_1/Oem/Nvidia/
↪WorkloadPowerProfile/Actions/NvidiaWorkloadPower.EnableProfiles HTTP/1.1
Content-Type: application/json

{
  "ProfileMask": "0x7"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed successfully.",
      "MessageId": "Base.1.18.1.Success",
      "Severity": "OK",
      "Resolution": "None."
    }
  ]
}
```

2.5.2.3.14.36 Disable Workload Profiles

- **Endpoint**: `POST /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/Oem/Nvidia/WorkloadPowerProfile/Actions/NvidiaWorkloadPower.DisableProfiles`
- **Path Parameters**:
  - `{gpu_id}` - GPU ID (GPU_0-GPU_7)
- **Arguments** (request payload):
  - `ProfileMask` - Hex mask of profiles to disable; must use `0x` prefix (e.g., `"0x7"` to disable profiles 0, 1, 2)

Example:

```
POST /redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_SXM_1/Oem/Nvidia/
↪WorkloadPowerProfile/Actions/NvidiaWorkloadPower.DisableProfiles HTTP/1.1
Content-Type: application/json

{
  "ProfileMask": "0x7"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed successfully.",
      "MessageId": "Base.1.18.1.Success",
      "Severity": "OK",
      "Resolution": "None."
    }
  ]
}
```

2.5.2.3.14.37 Profile Mask Format

The `ProfileMask` is a hex string with `0x` prefix where each bit position represents a profile ID:

- `"0x0"` - No profiles enabled
- `"0x1"` - Profile 0 only (bit 0 set)
- `"0x7"` - Profiles 0, 1, and 2 (bits 0, 1, 2 set)

2.5.2.3.14.38 Firmware Validation

- **Endpoint**: `GET /redfish/v1/UpdateService/FirmwareInventory/HostBMC_0`
- **Attributes** (response):
  - `Version` - BMC firmware version; must be >= 25.0.0

Example:

```
GET /redfish/v1/UpdateService/FirmwareInventory/HostBMC_0 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/HostBMC_0",
  "Id": "HostBMC_0",
  "Name": "Host BMC",
  "Version": "25.04.17.00"
}
```

2.5.2.3.14.39 DGX GB200 NVL Systems

DGX GB200 NVL systems use EnvironmentMetrics API similar to B200, with additional support for multi-GPU processor modules.

2.5.2.3.14.40 Processor EnvironmentMetrics

Same API as DGX B200, with different processor counts:

- **GPU Endpoint**: `GET/PATCH /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/EnvironmentMetrics`
  - `{gpu_id}` - GPU ID (GPU_0-GPU_3, 4 GPUs total)
- **CPU Endpoint**: `GET/PATCH /redfish/v1/Systems/HGX_Baseboard_0/Processors/{cpu_id}/EnvironmentMetrics`
  - `{cpu_id}` - CPU ID (CPU_0-CPU_1, two CPUs total)
- **Arguments** (PATCH request payload):
  - `PowerLimitWatts.SetPoint` - Power limit in Watts

Example:

```
PATCH /redfish/v1/Systems/HGX_Baseboard_0/Processors/GPU_0/EnvironmentMetrics
↪HTTP/1.1
Content-Type: application/json

{
  "PowerLimitWatts": {
    "SetPoint": 700
  }
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@Message.ExtendedInfo": [
    {
```

(continues on next page)

(continued from previous page)

```
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed successfully.",
      "MessageId": "Base.1.18.1.Success",
      "Severity": "OK",
      "Resolution": "None."
    }
  ]
}
```

2.5.2.3.14.41 HGX Platform EnvironmentMetrics

Telemetry uses the same consolidated report as DGX B200 and DGX B300:

- **Endpoint**: `GET /redfish/v1/TelemetryService/MetricReports/HGX_PlatformEnvironmentMetrics_0`
- **Attributes** (response):
  - `MetricValues[].MetricProperty` - Sensor URL for a GPU, CPU, memory, or chassis sensor
  - `MetricValues[].MetricValue` - Sensor value as a string

2.5.2.3.14.42 Processor Module EnvironmentMetrics

For multi-GPU module power management:

2.5.2.3.14.43 Get Module Metrics

- **Endpoint**: `GET /redfish/v1/Chassis/HGX_ProcessorModule_{index}/EnvironmentMetrics`
- **Path Parameters**:
  - `{index}` - Processor module index (`0` or `1`)
- **Attributes** (response): Same as processor EnvironmentMetrics - `PowerWatts.Reading`, `PowerLimitWatts.*` fields

Example:

```
GET /redfish/v1/Chassis/HGX_ProcessorModule_0/EnvironmentMetrics HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/Chassis/HGX_ProcessorModule_0/EnvironmentMetrics",
  "PowerWatts": { "Reading": 2500.0 },
  "PowerLimitWatts": {
    "Reading": 2820.0,
    "SetPoint": 2820,
    "AllowableMin": 1500,
    "AllowableMax": 3000,
    "DefaultSetPoint": 2820
```

(continues on next page)

(continued from previous page)

```
  }
}
```

2.5.2.3.14.44 Set Module Power Limit

- **Endpoint**: `PATCH /redfish/v1/Chassis/HGX_ProcessorModule_{index}/EnvironmentMetrics`
- **Path Parameters**:
  - `{index}` - Processor module index (`0` or `1`)
- **Arguments** (request payload):
  - `PowerLimitWatts.SetPoint` - Module power limit in Watts

Example:

```
PATCH /redfish/v1/Chassis/HGX_ProcessorModule_0/EnvironmentMetrics HTTP/1.1
Content-Type: application/json

{
  "PowerLimitWatts": {
    "SetPoint": 2820
  }
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

2.5.2.3.14.45 Workload Power Profiles (WPPS)

Same as DGX B200 - refer to *Workload Power Profiles (WPPS)*.

2.5.2.3.14.46 Firmware Validation

- **Endpoint**: `GET /redfish/v1/UpdateService/FirmwareInventory/FW_BMC_0`
- **Attributes** (response):
  - `Oem.Nvidia.ActiveFirmwareSlot.Version` - BMC firmware version (NVIDIA OEM format)

Example (NVIDIA OEM format):

```
GET /redfish/v1/UpdateService/FirmwareInventory/FW_BMC_0 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/FW_BMC_0",
  "Id": "FW_BMC_0",
  "Name": "BMC firmware",
```

(continues on next page)

(continued from previous page)

```
  "Oem": {
    "Nvidia": {
      "ActiveFirmwareSlot": {
        "Version": "25.25.2",
        "FirmwareState": "Activated"
      }
    }
  }
}
```

2.5.2.3.14.47 DGX B300 and GB300 NVL Systems

These systems expose EnvironmentMetrics resources similar to DGX B200 and GB200 NVL. The `DGX_B300` topology model applies power policies through Node Manager; `DGX_B300EM` and `DGX_GB300` use EnvironmentMetrics for per-component power limits.

- **WPPS Support**: Yes

For GB300 NVL, processor module endpoints are available similar to GB200 NVL.

2.5.2.3.14.48 Firmware Validation

- **DGX B300**: `GET /redfish/v1/UpdateService/FirmwareInventory/HostBMC_0`
- **DGX GB300 NVL**: `GET /redfish/v1/UpdateService/FirmwareInventory/FW_BMC_0`

2.5.2.3.14.49 Vera Rubin NVL72 Systems

The `DGX_VR_NVL72` topology model uses four GPU processor resources (`GPU_0` through `GPU_3`) and two CPU resources (`CPU_0` and `CPU_1`). DPS reads the consolidated `HGX_PlatformEnvironmentMetrics_0` report and uses each processor's EnvironmentMetrics resource for power-limit operations. GPU actuation uses `Oem.Nvidia.GPUBasePowerWatts.SetPoint`; CPU actuation uses `PowerLimitWatts.SetPoint`.

WPPS uses the endpoints in *Workload Power Profiles (WPPS)* for `GPU_0` through `GPU_3`. Firmware validation reads `GET /redfish/v1/UpdateService/FirmwareInventory/FW_BMC_0` and checks against a minimum version of `VRB-2601-28.00`.

2.5.2.3.14.50 Complete Endpoint Reference

2.5.2.3.14.51 Session Management (All Platforms)

- `POST /redfish/v1/SessionService/Sessions` - Create session
- `DELETE /redfish/v1/SessionService/Sessions/{id}` - Delete session

2.5.2.3.14.52 Firmware Inventory (All Platforms)

- GET /redfish/v1/UpdateService/FirmwareInventory/HGX_FW_BMC_0 - DGX H100 and H200
- GET /redfish/v1/UpdateService/FirmwareInventory/HostBMC_0 - DGX B200 and DGX B300
- GET /redfish/v1/UpdateService/FirmwareInventory/FW_BMC_0 - DGX GB200 NVL, GB300 NVL, and Vera Rubin NVL72

2.5.2.3.14.53 Node Manager

These endpoints apply to the DGX_H100, DGX_B200, and DGX_B300 topology models.

- GET /redfish/v1/Managers/BMC/NodeManager/Domains - List domains
- GET /redfish/v1/Managers/BMC/NodeManager/Domains/{id} - Get domain
- POST /redfish/v1/Managers/BMC/NodeManager/Domains - Create domain
- PATCH /redfish/v1/Managers/BMC/NodeManager/Domains/{id} - Update domain
- DELETE /redfish/v1/Managers/BMC/NodeManager/Domains/{id} - Delete domain
- GET /redfish/v1/Managers/BMC/NodeManager/Domains/{domain_id}/Policies/{policy_id} - Get policy
- GET /redfish/v1/Managers/BMC/NodeManager/PSUPolicies - List PSU policies
- GET /redfish/v1/Managers/BMC/NodeManager/PSUPolicies/{id} - Get PSU policy

2.5.2.3.14.54 Telemetry (DGX H100 and H200)

- GET /redfish/v1/TelemetryService/MetricReports - List metric reports
- GET /redfish/v1/TelemetryService/MetricReports/NvidiaNMMetrics_0 - Get NVIDIA Node Manager metrics

2.5.2.3.14.55 Processors (DGX B200, GB200, B300 Series, and Vera Rubin NVL72)

- GET /redfish/v1/Systems/HGX_Baseboard_0/Processors - List processors
- GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/{id} - Get processor

2.5.2.3.14.56 Environment Metrics (DGX B200, GB200, B300 Series, and Vera Rubin NVL72)

- GET /redfish/v1/TelemetryService/MetricReports/HGX_PlatformEnvironmentMetrics_0 - Get consolidated platform metrics
- GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/{processor_id}/EnvironmentMetrics - Get processor metrics
- PATCH /redfish/v1/Systems/HGX_Baseboard_0/Processors/{processor_id}/EnvironmentMetrics - Set power limit
- GET /redfish/v1/Chassis/Chassis_0/EnvironmentMetrics - Get chassis metrics
- GET /redfish/v1/Chassis/HGX_ProcessorModule_{id}/EnvironmentMetrics - Get processor module metrics (GB200/GB300)

- `PATCH /redfish/v1/Chassis/HGX_ProcessorModule_{id}/EnvironmentMetrics` - Set module power limit (GB200/GB300)

2.5.2.3.14.57 Workload Power Profiles (DGX B200, GB200, B300 Series, and Vera Rubin NVL72)

- `GET /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/Oem/Nvidia/WorkloadPowerProfile` - Get WPPS configuration
- `POST /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/Oem/Nvidia/WorkloadPowerProfile/Actions/NvidiaWorkloadPower.EnableProfiles` - Enable profiles
- `POST /redfish/v1/Systems/HGX_Baseboard_0/Processors/{gpu_id}/Oem/Nvidia/WorkloadPowerProfile/Actions/NvidiaWorkloadPower.DisableProfiles` - Disable profiles

2.5.2.3.14.58 References

2.5.2.3.14.59 NVIDIA Platform Documentation

- NVIDIA DGX H100/H200 Redfish API Guide
- NVIDIA DGX B200 Redfish API Guide
- NVIDIA DGX Documentation

2.5.2.3.14.60 DMTF Redfish Specification

- DMTF Redfish Specification
- DMTF Redfish Specification DSP0266 v1.23.0 (PDF) - Session Management (Section 9.2.4)
- Redfish Schema Index

2.5.2.3.15 NvGrid

2.5.2.3.15.1 Overview

NvGrid provides grid integration capabilities that enable datacenter power management in response to utility grid signals. It allows grid operators, demand response aggregators, and power management systems to schedule datacenter power constraints aligned with grid conditions, pricing signals, and curtailment events.

NvGrid operates at the **power feed level**, providing granular control over individual electrical circuits or datacenter-wide power consumption.

2.5.2.3.15.2 NvGrid Architecture

NvGrid consists of:

- **Load Target Scheduler** - Event-driven scheduler for time-based power constraints
- **Feed Metadata** - Power limits and default constraints per electrical feed
- **System Status** - Current power targets and calculated consumption
- **Webhook Notifications** - Real-time power event callbacks

2.5.2.3.15.3 Power Feeds

Datacenters are powered by one or more electrical feeds, each representing a distinct circuit. NvGrid provides independent control over each feed.

2.5.2.3.15.4 Feed Properties

Each feed has operational boundaries defined by metadata:

- **Feed Tag**: Unique identifier (e.g., “feed-a”)
- **Power Limits**: Minimum and maximum power caps
- **Default Constraint**: Power limit when no targets are active

2.5.2.3.15.5 Load Target Scheduling

Load targets define time-based power constraints for datacenter feeds:

- **Interval** - Start and end times for the constraint
- **Load Constraint** - Target power limit (watts, kilowatts, or megawatts)
- **Strategy** - Algorithm for achieving the target (`best_effort`)
- **Feed Tags** - Specific feeds to constrain (or all if omitted)
- **Correlation ID** - Unique identifier for tracking grid signals

2.5.2.3.15.6 Schedule Resolution

For overlapping time intervals, the **most recently scheduled** load target takes precedence. When constraints expire or are removed, power reverts to the default constraint.

2.5.2.3.15.7 Power Management Strategy

NvGrid uses a `best_effort` strategy:

- Reduces power from DPM-enabled workloads without terminating jobs
- Target may not be achieved if constraint requires job termination
- Automatically restores power when constraints are relaxed
- Integrates with resource group power management

2.5.2.3.15.8 DPM-Enabled Jobs

Dynamic Power Management (DPM) is a resource group feature that allows power limits to be adjusted without terminating workloads. NvGrid can only reduce power from jobs that have DPM enabled.

How DPM Works with NvGrid:

- NvGrid reduces groups that have an effective DPM policy with a specified strategy, not groups selected only by the `dpm_enable` list field
- DPS reduces lower-priority eligible bands first (higher `--priority` number first) and shares the reduction among groups that share a number
- DPS adjusts power limits incrementally
- Jobs continue running at reduced power levels

- Power reduction is non-disruptive to workload execution
- Refer to *Resource Group Priority* for steal bands and giveback restore order

When a group has no effective DPM strategy:

- NvGrid cannot reduce power from that group
- Load targets may not be fully achieved
- System reports `compliant=false` in webhook events

Refer to *Resource Groups* for more information on DPM configuration. The `dpm_enable` field on `dpsctl resource-group list` is not the steal-eligibility check.

2.5.2.3.15.9 Power Restoration

When power constraints expire or are relaxed, DPS automatically restores power to DPM-enabled workloads.

2.5.2.3.15.10 Automatic Recovery

Power restoration occurs when:

- A load target's end time is reached
- A new target with higher power limit is scheduled
- Default constraint is higher than the expiring target

Giveback first tries to restore every eligible throttled member together. Priority tiers (lower number first) apply only when that complete plan cannot be admitted. Setting `--priority` does not start restoration by itself. Refer to *Resource Group Priority*.

2.5.2.3.15.11 Restoration Behavior

Power Distribution:

- DPM-enabled resource groups receive increased power allocations
- Power may be distributed differently than the original reduction
- Restoration respects workload power policies and limits
- Recovery occurs incrementally to maintain system stability

Timeline:

- Restoration begins automatically when constraint expires
- `START_RAMP_UP` webhook event signals restoration start
- `END_RAMP_UP` webhook event confirms new power level reached
- `in_flight` status tracks active power transitions

2.5.2.3.15.12 Restoring Default Schedule

To explicitly return to default constraints before a scheduled target expires, schedule a zero watt target for the time period:

```
dpsctl nvgrid set-load-target \
  --value=0 \
  --unit=watt \
  --start-time="2025-10-24T16:00:00Z" \
  --end-time="2025-10-24T19:00:00Z"
```

This cancels any active targets in that time range, allowing the system to revert to default constraints.

2.5.2.3.15.13 Webhook Events

NvGrid sends HTTP POST webhook notifications for power state changes:

- **START_RAMP_UP** - Power constraint is relaxing (increasing limit)
- **END_RAMP_UP** - Power ramp up complete, new higher limit reached
- **START_RAMP_DOWN** - Power constraint is tightening (decreasing limit)
- **END_RAMP_DOWN** - Power ramp down complete, new lower limit reached

Webhooks include power values, correlation IDs, and compliance status.

2.5.2.3.15.14 Webhook Security

Webhook URLs must be configured in the NvGrid whitelist for security. Only whitelisted URLs can be registered for event notifications.

Whitelist Configuration:

Configure allowed webhook URLs in DPS Helm values:

```
dps:
  nvgrid:
    webhook:
      whitelist:
        - "https://grid-solution.example.com/nvgrid/webhook"
        - "https://monitoring.example.com/api/"
```

Validation:

- RegisterWebhook API validates URLs against the whitelist
- Non-whitelisted URLs are rejected with error: “webhook URL not whitelisted”
- Whitelist supports path prefix matching (trailing `/` matches all sub-paths)

Refer to *Prerequisites - Webhook Security Configuration* for detailed whitelist rules.

2.5.2.3.15.15 Use Cases

Utility Grid Integration:

- Respond to curtailment events from utilities
- Participate in demand response programs
- Adjust power based on dynamic pricing signals
- Report compliance to grid operators

Power Management:

- Schedule maintenance windows with reduced power
- Coordinate with renewable energy availability
- Manage peak demand periods
- Support sustainability initiatives

2.5.2.3.15.16 Further Reading

- *Managing NvGrid* - Step-by-step operations guide
- *NvGrid CLI Commands* - Command reference
- *Resource Groups* - Workload power management integration
- *Resource Group Priority* - How grid redistribution steals and gives back power by priority

2.5.2.3.16 Slurm

Note

This page explains Slurm itself, for readers who have not used it. To connect DPS to an existing Slurm cluster, refer to *Integrating with Slurm*.

2.5.2.3.16.1 Overview

Slurm (Simple Linux Utility for Resource Management) is an open-source job scheduler and workload manager designed for high-performance computing (HPC) environments. It manages compute resources, schedules jobs, and provides a framework for parallel and distributed computing. Slurm is widely used in datacenters, research institutions, and enterprise environments to efficiently allocate computing resources among multiple users and applications.

2.5.2.3.16.2 Key Concepts

2.5.2.3.16.3 Job Scheduling

Slurm manages the allocation of compute resources (nodes, CPUs, GPUs, memory) to user jobs. It provides:

- **Job queuing** - Jobs wait in queues until resources become available
- **Resource allocation** - Automatic assignment of compute nodes and resources
- **Job prioritization** - Fair-share scheduling based on user quotas and priorities
- **Preemption** - Ability to suspend or terminate lower-priority jobs for higher-priority ones

2.5.2.3.16.4 Resource Management

Slurm tracks and manages:

- **Compute nodes** - Individual servers in the cluster
- **Partitions** - Logical groupings of nodes (e.g., “gpu”, “cpu-only”, “debug”)
- **Accounts** - User groups with resource quotas and limits
- **Quality of Service (QoS)** - Service levels that affect job priority and limits

2.5.2.3.16.5 Slurm CLI Tools

2.5.2.3.16.6 Core Commands

Job Submission:

```
# Submit a simple job
sbatch job_script.sh

# Submit with specific requirements
sbatch --nodes=4 --ntasks-per-node=8 --gres=gpu:4 job_script.sh

# Submit an interactive job
srun --pty --nodes=1 --ntasks=1 bash
```

Job Management:

```
# List all jobs
squeue

# List jobs for specific user
squeue -u username

# Cancel a job
scancel job_id

# Hold a job (prevent from starting)
scontrol hold job_id

# Release a held job
scontrol release job_id
```

Resource Information:

```
# Show cluster status
sinfo

# Show detailed node information
scontrol show nodes

# Show partition information
scontrol show partitions

# Show account information
sacctmgr show accounts
```

2.5.2.3.16.7 Job Scripts

Slurm job scripts are shell scripts with special directives:

```
#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
```

(continues on next page)

(continued from previous page)

```
#SBATCH --gres=gpu:4
#SBATCH --time=02:00:00
#SBATCH --partition=gpu
#SBATCH --account=my_account

# Job commands
module load cuda/11.8
mpirun -np 32 ./my_application
```

2.5.2.3.16.8 Prolog and Epilog Functionality

2.5.2.3.16.9 Overview

Slurm's prolog and epilog scripts provide hooks for custom actions before and after job execution. These scripts run on the compute nodes and can perform setup, cleanup, and integration tasks.

2.5.2.3.16.10 Prolog Scripts

Prolog scripts execute **before** a job starts on a compute node. Common uses include:

File System Operations:

```
#!/bin/bash
# Create job-specific directories
mkdir -p /scratch/job_${SLURM_JOB_ID}
ln -s /scratch/job_${SLURM_JOB_ID} $SLURM_SUBMIT_DIR/scratch
```

Environment Setup:

```
#!/bin/bash
# Load required modules
module load cuda/11.8
module load openmpi/4.1.4

# Set environment variables
export CUDA_VISIBLE_DEVICES=0,1,2,3
export OMP_NUM_THREADS=4
```

2.5.2.3.16.11 Epilog Scripts

Epilog scripts execute **after** a job completes on a compute node. Common uses include:

Cleanup Operations:

```
#!/bin/bash
# Remove job-specific files
rm -rf /scratch/job_${SLURM_JOB_ID}
rm -f $SLURM_SUBMIT_DIR/scratch
```

Logging and Monitoring:

```
#!/bin/bash
# Log job completion
echo "$(date): Job ${SLURM_JOB_ID} completed on $(hostname)" >> /var/log/
↪slurm/jobs.log

# Collect performance metrics
sacct -j ${SLURM_JOB_ID} --format=JobID,JobName,Elapsed,MaxRSS,MaxVMSize >> /
↪var/log/slurm/metrics.log
```

2.5.2.3.16.12 Further Reading

- Slurm Documentation - Official Slurm documentation
- *Resource Groups* - DPS resource group management
- *DPS Integration Guide* - Detailed DPS-Slurm integration
- Slurm Prolog/Epilog Guide - Official prolog/epilog documentation

2.5.2.3.17 Base Command Manager

2.5.2.3.17.1 Overview

NVIDIA Base Command Manager (BCM) is a comprehensive cluster management solution designed specifically for AI and HPC workloads. It provides centralized management of NVIDIA DGX systems, GPU clusters, and heterogeneous computing environments. BCM simplifies datacenter operations by offering automated provisioning, monitoring, and management capabilities for large-scale AI infrastructure.

2.5.2.3.17.2 Key Concepts

2.5.2.3.17.3 Cluster Management

BCM provides unified management of:

- **Compute nodes** - DGX systems, GPU servers, and traditional compute nodes
- **Storage systems** - Network-attached storage and distributed file systems
- **Network infrastructure** - High-speed interconnects and network switches
- **User management** - Authentication, authorization, and resource quotas

2.5.2.3.17.4 Resource Orchestration

BCM orchestrates resources through:

- **Automated provisioning** - Bare metal and container-based deployment
- **Configuration management** - Centralized system configuration and updates
- **Monitoring and alerting** - Real-time health monitoring and proactive maintenance
- **Backup and recovery** - Automated backup strategies and disaster recovery

2.5.2.3.17.5 BCM Architecture

2.5.2.3.17.6 Core Components

BCM Head Node:

- Central management server running BCM software
- Web-based management interface
- REST API for programmatic access
- Database for configuration and monitoring data

BCM Compute Nodes:

- Managed compute resources (DGX systems, GPU servers)
- BCM agent software for communication with head node
- Automated configuration and monitoring capabilities

BCM Storage:

- Centralized configuration and user data storage
- Backup and recovery management
- Shared file systems and data management

2.5.2.3.17.7 DPS Integration

DPS (Dynamic Power Software) is integrated into BCM as a plugin to provide power management and optimization capabilities for the infrastructure BCM manages.

2.5.2.3.17.8 Entity Generation

BCM can automatically generate DPS entities from its cluster inventory. Entity generation uses an asynchronous two-step workflow:

```
# Initiate entity generation from BCM cluster (returns a request ID)
dpsctl import entity --source bcm \
  --username admin \
  --password secret123
```

Example response:

```
{
  "request_id": "abc123-def456-ghi789"
}
```

Step 2: Retrieve generated entities

```
# Poll for results using the request ID
dpsctl import entity-status --request-id abc123-def456-ghi789
```

Generated Entity Example:

```
[
  {
    "Name": "dgx001",
    "Type": "ComputerSystem",
    "Model": "DGX_H100",
    "Redfish": {
      "URL": "https://dgx001-bmc.example.com",
      "SecretName": "dgx001"
    }
  }
]
```

2.5.2.3.17.9 Further Reading

- NVIDIA Base Command Manager - Official BCM product page
- Base Command Platform Documentation - Complete platform documentation
- *BCM Integration with DPS* - Entity generation from BCM
- *DPS Import Commands* - DPS CLI entity import from BCM and other sources

2.5.2.3.18 DPS Nautobot Integration - Current Use Cases

2.5.2.3.18.1 Overview

This document describes the Nautobot import flow as it exists today. It is written for developers and architects working against the import API.

If you are performing an import rather than building against it, refer to *Importing External Inventory*.

Current implementation spans:

- Web UI import wizard
- CLI commands for generate/status/cancel/delete
- gRPC/HTTP API surfaces used by the UI and clients

It does not currently include saved default queries or import-specific RBAC.

2.5.2.3.18.2 Current API Surface

The current Nautobot import flow relies on these operations:

- `GenerateEntities`: starts an async fetch and returns a `request_id`
- `GetGenerateEntitiesStatus`: streams generated `Entity` messages for a `request_id`
- `CancelGenerateEntitiesRequest`: cancels a pending or running request
- `DeleteGenerateEntitiesRequest`: deletes a terminal request and its staged results
- `UpsertEntities`: imports selected entities into the main DPS entity store

2.5.2.3.18.3 Interface Scope

The interfaces do not all expose the same workflow:

- Web UI: fetch, review, select, optional edit, and import
- CLI: generate, status, cancel, and delete primitives
- API: the same low-level primitives used by clients

The guided review-and-import experience is currently a Web UI workflow, not a single dedicated first-class CLI or API workflow but can be accomplished in multiple manual steps.

2.5.2.3.18.4 Filter Semantics

Nautobot filter handling in the current implementation works as follows:

- Structured filters are `locations`, `racks`, `device_types`, and `request_string`
- Human-friendly values such as location names, rack names, and device type models may be resolved to Nautobot UUIDs before the device query runs
- Rack-name lookup is location-scoped when locations are available
- UUID values are passed through without lookup
- Some parameters in `request_string` are reserved for structured handling, while other parameters are passed through to the Nautobot devices endpoint

2.5.2.3.18.5 Persistence and Cleanup

Fetched entities are staged before final import:

- Async generate requests and fetched entities are stored in the `generate_entities_*` tables
- This staging data is separate from the main DPS entity store
- `GetGenerateEntitiesStatus` reads from staged request state and staged entities
- `UpsertEntities` writes selected entities into the main DPS entity store
- `DeleteGenerateEntitiesRequest` removes staged request data; it does not remove already imported entities
-

2.5.2.3.18.6 Capability 1: Nautobot-to-DPS Entity Fetch and Import

2.5.2.3.18.7 Use Case 1.1: Operator Queries Nautobot for Candidate Entities

Actor: Operator (via Web UI, CLI, or API)

Precondition: DPS is configured with Nautobot access.

Flow:

1. Operator submits Nautobot filters: `locations`, `racks`, `device_types`, and/or `request_string`.
2. DPS validates the request.
3. DPS creates an async `GenerateEntities` request and returns a `request_id`.

4. DPS resolves human-friendly filter values to Nautobot UUIDs where needed.
5. DPS queries Nautobot and stores generated entities in the `generate_entities_*` staging tables.
6. Client retrieves the generated entities by calling `GetGenerateEntitiesStatus(request_id)`, which streams `Entity` messages until completion.
7. Web UI displays the fetched entities for review.

Postcondition: Candidate entities exist in the generate-entities staging tables and are available for review. They are not yet imported into the main DPS entity store.

Current validation rules:

- At least one filter is required
- Location-only is rejected unless the query string provides an allowed identifier such as `rack=UUID`
- Device-type-only is rejected
- Rack names require a location filter; rack UUIDs do not
- Query-string-only requests are allowed only for specific identifier-style filters

Error paths:

- Invalid filters -> validation error
- Nautobot unreachable or query failure -> request reaches terminal error state
- Missing request ID -> invalid argument
- Zero results -> successful request with no streamed entities

Concurrency note: The current implementation does not prevent multiple in-flight `GenerateEntities` requests per user or session. Each request gets its own `request_id`. The Web UI tracks a single `lastRequestId` and cleans up before re-fetch; CLI and API callers are responsible for managing their own request lifecycle.

2.5.2.3.18.8 Use Case 1.2: Operator Reviews and Selects Entities for Import

Actor: Operator (primarily via Web UI)

Precondition: Candidate entities were fetched via UC 1.1.

Flow:

1. Operator reviews the fetched entities.
2. Operator selects which entities to keep in the working set.
3. Operator optionally edits entities during the review step.
4. Operator confirms import.
5. UI uploads the selected entities through the existing `UpsertEntities` path.

Postcondition: Selected entities are upserted into DPS. They are no longer only staged import results; they are part of the DPS entity inventory and can be used to build a topology afterward.

Notes:

- The current repo does not have a dedicated `ImportEntities(selectedEntityIDs)` RPC.

- Selection and review are currently a Web UI workflow, not a first-class CLI workflow.
- "Optionally edits entities" refers to UI-level field adjustments before upsert; the specific editable fields and their validation are governed by the `UpsertEntities` schema, not this import flow.
- Re-importing the same entities is handled by `UpsertEntities` semantics (create-or-update); there is no special duplicate detection or merge logic in the import path.

2.5.2.3.18.9 Use Case 1.3: Operator Cancels or Clears a Pending Fetch

Actor: Operator (via Web UI, CLI, or API)

Precondition: A generate-entities request exists.

Flow:

1. Operator chooses to cancel a request.
2. DPS calls `CancelGenerateEntitiesRequest(request_id)`.
3. If the request is pending or running, DPS cancels the stored context.
4. DPS marks the request status as `cancelled`.
5. If the operator also wants staged results removed, client calls `DeleteGenerateEntitiesRequest(request_id)` after the request is terminal.

Postcondition:

- After cancel: request is terminal and no more work should continue
- After delete: request row and staged entities are removed from the staging tables

Important distinction:

- `CancelGenerateEntitiesRequest` does not itself delete the staging row
- `DeleteGenerateEntitiesRequest` is a separate operation
- In the current Web UI, "Remove All" performs cancel first and then attempts delete

---

2.5.2.3.18.10 Use Case 1.4: Operator Re-fetches with New Filters

Actor: Operator

Precondition: A previous request may still exist.

Flow in the current Web UI:

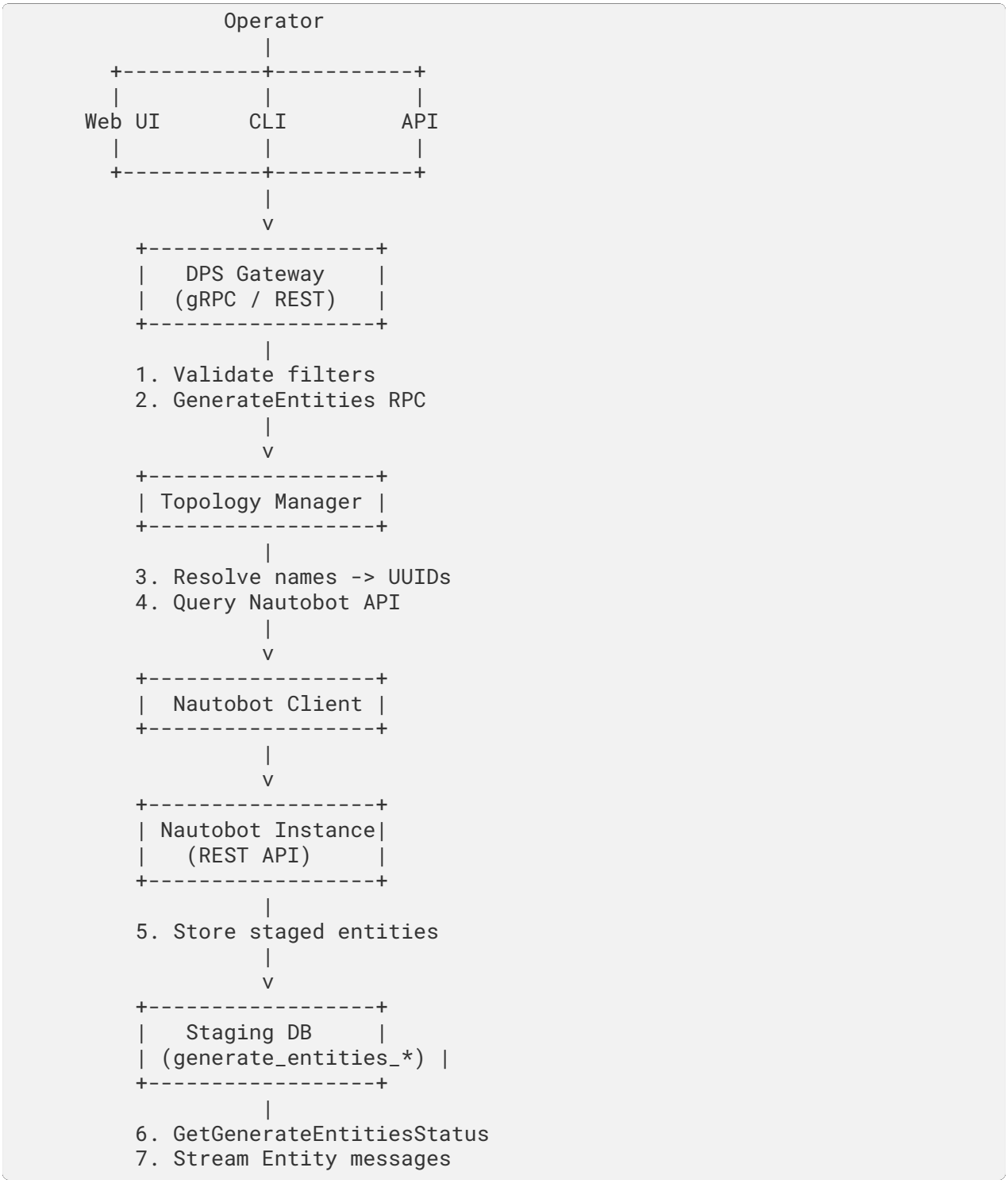
1. Operator changes filters and clicks Fetch again.
2. UI attempts to cancel the previous request.
3. UI then attempts to delete the previous request and its staged entities.
4. UI clears the local working set.
5. UI starts a new `GenerateEntities` request.

Postcondition: The UI presents only the newly fetched candidate entities.

Scope note: Automatic cleanup on re-fetch is currently a Web UI behavior. CLI and direct API clients manage request lifecycle explicitly.

2.5.2.3.18.11 Architecture and Sequences

2.5.2.3.18.12 UC 1.1 - Query Nautobot for Candidate Entities



Sequence:

2.5.2.3.18.13 UC 1.2 - Review and Import Selected Entities

Expanded sequence:

2.5.2.3.18.14 UC 1.3 - Cancel and Optional Delete

2.5.2.3.18.15 UC 1.4 - Re-fetch with New Filters

2.5.2.3.18.16 UC 2.1 - Auto-Import with Continuous Sync

2.5.2.3.18.17 Request State Diagram

The async generate-entities request lifecycle is:

Notes:

- `GenerateEntities` creates the request in `pending`
- Worker execution moves it to `running`
- Successful generation moves it to `complete`
- Backend failures move it to `error`
- `CancelGenerateEntitiesRequest` moves a request to `cancelled`
- `DeleteGenerateEntitiesRequest` removes terminal requests from storage; it is not itself a request state
- In the current implementation, delete is rejected for `pending` or `running` requests

2.5.2.3.19 User Accounts

2.5.2.3.19.1 Overview

User accounts identify individual users who operate Dynamic Power Software (DPS) through `dpsctl`, the UI, or the gRPC and REST APIs. Use this guide to connect to an already configured DPS server. Administrators configuring server authentication should follow *Configure Server Authentication*. For applications, services, and scripts, use *Service Accounts*.

2.5.2.3.19.2 Sign In to DPS

Your administrator provides the API endpoint, trusted certificates, and configured authentication mode. With Lightweight Directory Access Protocol (LDAP), use your directory username and password. With OpenID Connect (OIDC), sign in through your identity provider.

Use the hostname, port, and certificate paths for your environment in the following examples. Omit `--ca-cert-path` when the server certificate uses a certificate authority (CA) your system trusts. `dpsctl` stores tokens in `${HOME}/.dpsctl/credentials.yaml` by default. For full options, refer to *dpsctl login*.

Follow the instructions for *LDAP* or *OIDC*, according to your authentication mode.

2.5.2.3.19.3 LDAP

In LDAP mode, your directory administrator provisions your account and its required group membership. The following examples use the reference OpenLDAP chart. In the UI, enter your directory credentials on the sign-in page.

2.5.2.3.19.4 Groups and Roles

The reference OpenLDAP chart creates users in `ou=users` and the following groups:

- `rogroup`, at `cn=rogroup,dc=cm,dc=cluster`, is a directory group. It does not grant DPS API access by itself.
- `dcpower-users`, in `ou=groups`, identifies DPS users.
- `dcpower-admins`, in `ou=groups`, identifies administrator accounts.
- `automation-users`, in `ou=groups`, identifies service accounts.

The server's `dps.ldap.defaultRole` and `dps.ldap.groupRoleMapping` settings determine the LDAP role mapping. The server separately checks `dps.auth.auth_required_groups` for API access when the setting is nonempty. Successful sign-in does not guarantee that a later API request is authorized. Ask your administrator to check the required membership through *Configure LDAP Mode*.

2.5.2.3.19.5 Default Users

The reference OpenLDAP chart creates the following example users for development and testing:

- `testuser` / `testuser` is a member of `rogroup` and `dcpower-users`.
- `adminuser` / `adminuser` is a member of `dcpower-users` and `dcpower-admins`.
- `non-dcpower-user` / `non-dcpower-user` is a member of `rogroup` only, for testing access restrictions.

These example credentials are for development and testing. Provision your own accounts and configure access for a secure deployment.

2.5.2.3.19.6 Create a User Account

If you administer the directory, use the following LDAP Data Interchange Format (LDIF) example to add `alice`. Replace the user and group IDs and the password hash with values for your directory. Select group membership for the configured access requirements.

Example LDAP User and Group Membership

```
# New user
dn: uid=alice,ou=users,dc=cm,dc=cluster
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: Alice
sn: Example
uid: alice
uidNumber: 15001
gidNumber: 15000
homeDirectory: /home/alice
```

(continues on next page)

(continued from previous page)

```
loginShell: /bin/bash
mail: alice@example.com
userPassword: ${SSHA_PASSWORD_HASH}

# Add to the reference directory group
dn: cn=rogroup,dc=cm,dc=cluster
changetype: modify
add: member
member: uid=alice,ou=users,dc=cm,dc=cluster

# Add to the DPS users group
dn: cn=dcpower-users,ou=groups,dc=cm,dc=cluster
changetype: modify
add: member
member: uid=alice,ou=users,dc=cm,dc=cluster
```

After applying the LDIF, verify the new account with a password prompt:

```
ldapwhoami \
  -x -H ldaps://openldap.dps.svc.cluster.local:636 \
  -D "uid=alice,ou=users,dc=cm,dc=cluster" \
  -W
```

2.5.2.3.19.7 Sign In with dpsctl

In LDAP mode, enter your directory password at the prompt:

```
dpsctl --host api.dps.example.com --port 443 \
  --ca-cert-path /path/to/api-ca.crt login --username alice
```

The server returns an access token and a refresh token. `dpsctl` refreshes the access token when needed.

For a workflow that supplies a password file through secret management, pass its contents through standard input:

```
dpsctl login --username alice --password-stdin < /path/to/password-file
```

You can also supply `DPSCTL_USERNAME` and `DPSCTL_PASSWORD` through your secret-management workflow before running `dpsctl login`.

2.5.2.3.19.8 Programmatic Token Retrieval

In LDAP mode, custom clients can exchange a username and password through `nvidia.dcpower.v1.AuthService/Token`. Send a `passwordCredential` with `username` and `password` fields over TLS. Use `nvidia.dcpower.v1.AuthService/RefreshToken` to refresh the returned tokens. For request and response definitions, refer to *APIs*.

2.5.2.3.19.9 OIDC

In OIDC mode, your identity-provider administrator provisions your user account. The UI and interactive `dpsctl` must use separate public OAuth clients without client secrets. Neither client sends a secret during token exchange. Administrators register these clients through *Configure OIDC Mode*.

2.5.2.3.19.10 Sign In to the UI

Open the UI sign-in page and follow the redirect to your identity provider. Complete sign-in there. The UI uses the authorization code flow with Proof Key for Code Exchange (PKCE) and sends the access token with API requests by default.

2.5.2.3.19.11 Sign In with dpsctl

Use the issuer and public client ID provided by your administrator. The provider must support the device authorization flow. Run the following command without username/password options or a client secret.

```
dpsctl --oidc-issuer https://idp.example.com/realms/dps \
  --oidc-client-id dpsctl \
  --oidc-oauth-scope dps:api login
```

Open the displayed URL in a browser and enter the displayed code if requested. Complete sign-in through your identity provider. Browser login contacts the provider directly. Configure the DPS API endpoint on later commands, as shown in *Check API Access*.

Use `--oidc-ca-cert-file` if the identity provider uses a private CA. For endpoint overrides, refresh behavior, and optional ID token use, refer to *dpsctl login*.

For unattended applications, obtain client credentials for a separate confidential client as described in *Obtain a Service Account Token*.

2.5.2.3.19.12 Mutual TLS

Mutual TLS (mTLS) client certificates can authenticate to specific gRPC services. For the certificate requirements and client command, refer to *Mutual TLS*. UI access uses LDAP or OIDC sign-in.

2.5.2.3.19.13 Check API Access

After sign-in, confirm that your account can make a read-only request:

```
dpsctl --host api.dps.example.com --port 443 \
  --ca-cert-path /path/to/api-ca.crt topology list
```

If the request is denied, ask your administrator to check the configured access requirements. Repeating sign-in with the same account does not change its permissions.

2.5.2.3.19.14 Best Practices

Use the following practices to manage user accounts:

- Assign users only the groups and permissions they require.
- Enforce your organization's password and multifactor authentication requirements through the configured identity system.

- Encrypt LDAP, identity-provider, and DPS connections in secure deployments.
- Deactivate or remove accounts when users leave the organization.

2.5.2.3.19.15 Further Reading

Refer to the following guides for account and server setup:

- *Service Accounts* for non-interactive service accounts.
- *`dpsctl` installation instructions*
- *Configure Server Authentication*

2.5.2.3.20 Service Accounts

2.5.2.3.20.1 Overview

Service accounts identify applications, services, and scripts that call Dynamic Power Software (DPS) without an interactive user sign-in. Use a separate identity for each service, such as Power Reservation Steering (PRS) or a deployment job.

Use this guide to connect services and scripts to an already configured DPS server. Your administrator provides the API endpoint, trusted certificates, and account credentials. Administrators configuring server authentication should follow *Configure Server Authentication*.

DPS supports the following authentication modes for service accounts:

- In Lightweight Directory Access Protocol (LDAP) mode, a directory service account exchanges a username and password for DPS tokens.
- In OpenID Connect (OIDC) mode, a confidential OAuth client obtains an access token from the identity provider through the client credentials grant.
- With mutual TLS (mTLS), a client authenticates with a certificate approved for the requested gRPC service.

Use the mode your administrator configured. When an external service handles authentication and authorization, your administrator specifies the accepted tokens or certificates through *Configure External Authentication and Authorization*.

2.5.2.3.20.2 LDAP

2.5.2.3.20.3 Create a Service Account

Ask your directory administrator to provision a service account with the required group membership. The reference OpenLDAP chart places its PRS service account in `ou=users` and uses the following group:

```
cn=automation-users,ou=groups,dc=cm,dc=cluster
```

If you administer the directory, use the following LDAP Data Interchange Format (LDIF) example to add `myservice`. Replace the user and group IDs and the password hash with values for your directory.

Example LDAP Service Account

```
# New automation user
dn: uid=myservice,ou=users,dc=cm,dc=cluster
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: myservice
sn: service
uid: myservice
uidNumber: 20001
gidNumber: 20000
homeDirectory: /home/myservice
loginShell: /usr/sbin/nologin
mail: myservice@cm.cluster
userPassword: ${SSHA_PASSWORD_HASH}

# Add the user to the automation group
dn: cn=automation-users,ou=groups,dc=cm,dc=cluster
changetype: modify
add: member
member: uid=myservice,ou=users,dc=cm,dc=cluster
```

After applying the LDIF, verify the account with a password prompt:

```
ldapwhoami \
  -x -H ldaps://openldap.dps.svc.cluster.local:636 \
  -D "uid=myservice,ou=users,dc=cm,dc=cluster" \
  -W
```

2.5.2.3.20.4 Account Discovery Requirements

Your server administrator must set `dps.ldap.baseDn` to a directory subtree that contains the account and configure `dps.ldap.userFilter` to match its attributes. The preceding example uses `dc=cm,dc=cluster` as the search base and this user filter:

```
(&(objectClass=inetOrgPerson)(uid={username}))
```

Directory administrators can place service accounts in a dedicated organizational unit (OU) if the search base includes it. An account's location under an OU does not add an `ou` attribute. Match attributes present on the user entry.

Ask your server administrator to check group resolution and required API group membership through *Configure LDAP Mode*. Server administrators apply configuration changes through their deployment workflow and restart the server to load them.

2.5.2.3.20.5 Obtain a DPS Token

In LDAP mode, `nvidia.dcpower.v1.AuthService/Token` exchanges a `passwordCredential` containing `username` and `password` for an access token and a refresh token. Send this request over Transport Layer Security (TLS). The response contains `accessToken` and `refreshToken` fields.

For `dpsctl`, use the endpoint and TLS options for your deployment, and enter the service-account password at the prompt:

```
dpsctl login --username myservice
```

For unattended provisioning, supply the password to `--password-stdin` through your secret-management workflow. `dpsctl` stores the returned tokens in its credentials file and refreshes them when needed.

Other clients call `nvidia.dcpower.v1.AuthService/RefreshToken` with the stored refresh token when the access token expires. Refer to *APIs* for request and response definitions. Verify the account's API access after token issuance through *Verify the Configuration*.

2.5.2.3.20.6 OIDC

In OIDC mode, service accounts use OAuth clients. Ask your identity-provider administrator to register a separate confidential client for each service, enable the OAuth 2.0 client credentials grant, and provide its client ID and secret. The UI and interactive `dpsctl` use separate public clients without client secrets.

2.5.2.3.20.7 Obtain a Service Account Token

`dpsctl` can obtain a short-lived service account token from the OIDC provider using the client credentials grant.

Provide the token endpoint, client ID, and client secret. Prefer the secret-file form:

```
DPSCTL_OIDC_CLIENT_ID=<oauth-client-id> \
DPSCTL_OIDC_CLIENT_SECRET_FILE=/path/to/client-secret \
DPSCTL_OIDC_TOKEN_URL=https://<keycloak-host>/realms/<realm>/protocol/openid-
↪connect/token \
dpsctl --host <dps-api-host> --port <grpc-port> topology list
```

`dpsctl` requests the `dps:api` scope by default and caches valid client-credentials tokens in the credentials file. The token endpoint must use HTTPS unless local development explicitly enables `--oidc-allow-http`.

The available flags, environment variables, secret-file option, and cache behavior are documented in the *`dpsctl` flag reference*.

Client credentials apply to normal `dpsctl` commands. Use *dpsctl login* for interactive sign-in through LDAP or the OIDC device authorization flow.

2.5.2.3.20.8 Mutual TLS

2.5.2.3.20.9 Authenticate with a Client Certificate

Ask your administrator for a trusted client certificate and private key. The certificate common name (CN) must be permitted for each gRPC service you call. For server configuration, refer to *Configure Service-Specific mTLS*.

Use the following command when the certificate permits `VersionService` requests. Replace the hostname, port, and certificate paths with your values. Omit `--ca-cert-path` if the server certificate uses a certificate authority (CA) your system trusts.

```
dpsctl --host api.dps.example.com --port 443 --auth-mode mtls \
  --ca-cert-path /path/to/api-ca.crt \
  --client-cert-path /path/to/client.crt \
  --client-key-path /path/to/client.key server-version
```

No `dpsctl login` step is required. The `mtls` client mode sends no bearer token. A certificate accepted for one service does not grant access to other services. For certificate options and other client modes, refer to *Flags*.

2.5.2.3.20.10 Best Practices

Use the following practices to manage service accounts:

- Create separate identities for each service and grant only the permissions they require.
- Rotate LDAP passwords, OAuth client secrets, or client certificates through the system that manages each identity.
- Encrypt LDAP, identity-provider, and DPS connections in secure deployments.
- Keep passwords, tokens, and client keys in a secret management system such as Kubernetes Secrets or HashiCorp Vault.

2.5.2.3.20.11 Further Reading

Refer to the following guides for client and server setup:

- *DPS CLI (`dpsctl`) installation*
- *nvidia.dcpower.v1 API documentation*
- *Configure Server Authentication*

2.5.2.3.21 dpsctl

2.5.2.3.21.1 Overview

`dpsctl` is the command-line interface for DPS that provides comprehensive management and monitoring capabilities. It serves as the primary tool for administrators, operators, and users to interact with the DPS system from the command-line.

2.5.2.3.21.2 Command-Line Implementations

The DPS 0.9 source contains two clients named `dpsctl`. Their command trees are not interchangeable:

Interface	Policy command shape
Go `dpsctl`	The `policy bundle` subgroup with `upsert`, `get`, `list`, and `delete`
`libdpsapi` C++ `dpsctl`	`policy list` and `policy upsert --filename <file>`

The command categories and command reference in this guide describe the Go implementation unless a page explicitly identifies the `libdpsapi` C++ interface.

2.5.2.3.21.3 Command Categories

The Go `dpsctl` implementation organizes functionality into logical command groups:

2.5.2.3.21.4 Infrastructure Management

- **topology** - Import, activate, and manage datacenter topologies
- **import** - Import entities from external sources (BCM, Nautobot)
- **device** - Manage device specifications and hardware capabilities
- **policy** - Manage named, typed policy bundles through the `bundle` subgroup

2.5.2.3.21.5 Workload Management

- **resource-group** - Create, activate, and manage workload power allocation
- **gpu-policy** - Set per-GPU power limits for fine-grained control

2.5.2.3.21.6 Authentication & Access

- **login** - Authenticate with DPS server and manage user sessions
- **verify** - Check DPS deployment status and connectivity

2.5.2.3.21.7 Monitoring & Diagnostics

- **check** - Diagnostic operations, connectivity tests, and health monitoring
- **server-version** - Get DPS server version and build information
- **task** - Monitor ongoing async tasks (e.g., activation/deactivation operations)

2.5.2.3.21.8 Usage

`dpsctl` can be installed as a native binary, from package repositories, or run as a container image. Refer to *Installing dpsctl* for all options.

Configure authentication for your server's setup through *Configure Server Authentication*. The following example uses LDAP login:

```
# Authenticate with DPS
dpsctl login --username alice

# Import datacenter topology
dpsctl topology import datacenter.json

# Activate the imported topology
dpsctl topology activate --topology datacenter

# Check BMC connectivity in the active topology
dpsctl check connection --topology datacenter

# Create workload resource group
dpsctl resource-group create --resource-group "ml-job" --external-id 1 --
↪policy-bundle MaxLPS

# Check system health
dpsctl check connection
```

2.5.2.3.21.9 Further Reading

- *dpsctl Installation* - Install and configure the CLI
- *User Accounts* - Authentication for interactive users
- *Service Accounts* - Authentication for programmatic access
- *Configure Server Authentication* - Server setup and client login flows
- *dpsctl Command Reference* - Complete command documentation

2.5.3. Operating DPS

Day-to-day procedures for a running DPS deployment, grouped by what you are trying to do. If a command fails, refer to *Troubleshooting*.

For an end-to-end scenario rather than an individual operation, refer to *MaxLPS Power Pilot Runbooks*.

2.5.3.1 Change what DPS manages

The operations you perform most often: adjusting which hardware DPS knows about, and which workloads hold power.

- ***Managing Resource Groups*** — create and manage the groups that hold power for a workload
- ***Resource Group Priority*** — set `--priority` and understand steal versus giveback order
- ***Managing Devices*** — manage the devices in your deployment
- ***Import a Topology*** — validate and import a topology from JSON through the CLI or API
- ***Managing Topologies*** — work with topology configurations
- ***Add nodes to DPS*** — bring new hardware into an existing deployment
- ***DPS UI Operator Guide*** — create, validate, and inspect these workflows in the DPS UI

2.5.3.2 Watch what it is doing

- ***Telemetry and Observability*** — the metrics DPS exposes and the queries that make them useful
- ***Provisioned and Observed Power*** — how to read the gap between committed and actual power, and what a persistent gap tells you
- ***Prometheus monitoring*** — collect and visualize power metrics

2.5.3.3 Diagnose a problem

- ***Run a BMC Health Check*** — run read-only or approved write-enabled BMC health checks and interpret the report
- ***Troubleshooting*** — operator diagnosis steps and escalation criteria

2.5.3.4 Maintain the deployment

Less frequent, but the operations where being wrong is expensive.

- ***Backing up with Velero*** — back up DPS with Velero
- ***Credentials and Secrets Configuration*** — configure authentication and secrets management

- ***Database TLS*** — configure PostgreSQL TLS, including CA verify and mutual TLS client certificates

2.5.3.5 Connect other systems

Covered in full by *Integrating DPS*. The procedures themselves:

- ***Import External Entities*** — import devices from Nautobot or BCM
- ***Integrating with Slurm*** — connect DPS to Slurm
- ***Managing NvGrid*** — schedule power constraints for grid integration
- ***Installing DPS in BCM*** — set up DPS inside a BCM-managed cluster

2.5.3.5.1 Import External Entities

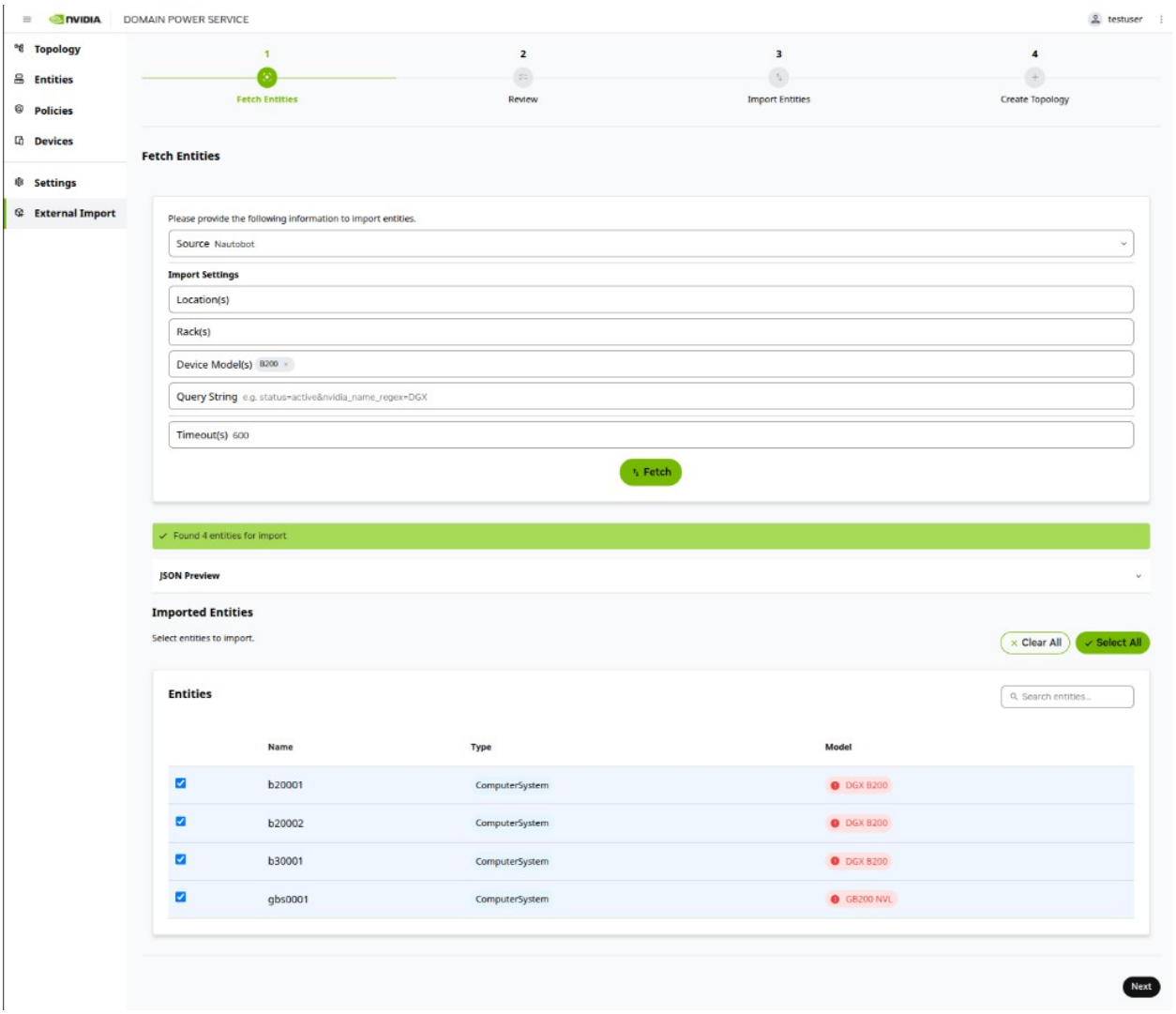
2.5.3.5.1.1 Overview

DPS can import devices directly from external inventory sources such as Nautobot (DCIM) or BCM (Base Command Manager). This allows you to quickly populate your DPS deployment with entities that reflect your existing datacenter infrastructure.

2.5.3.5.1.2 Importing from Nautobot

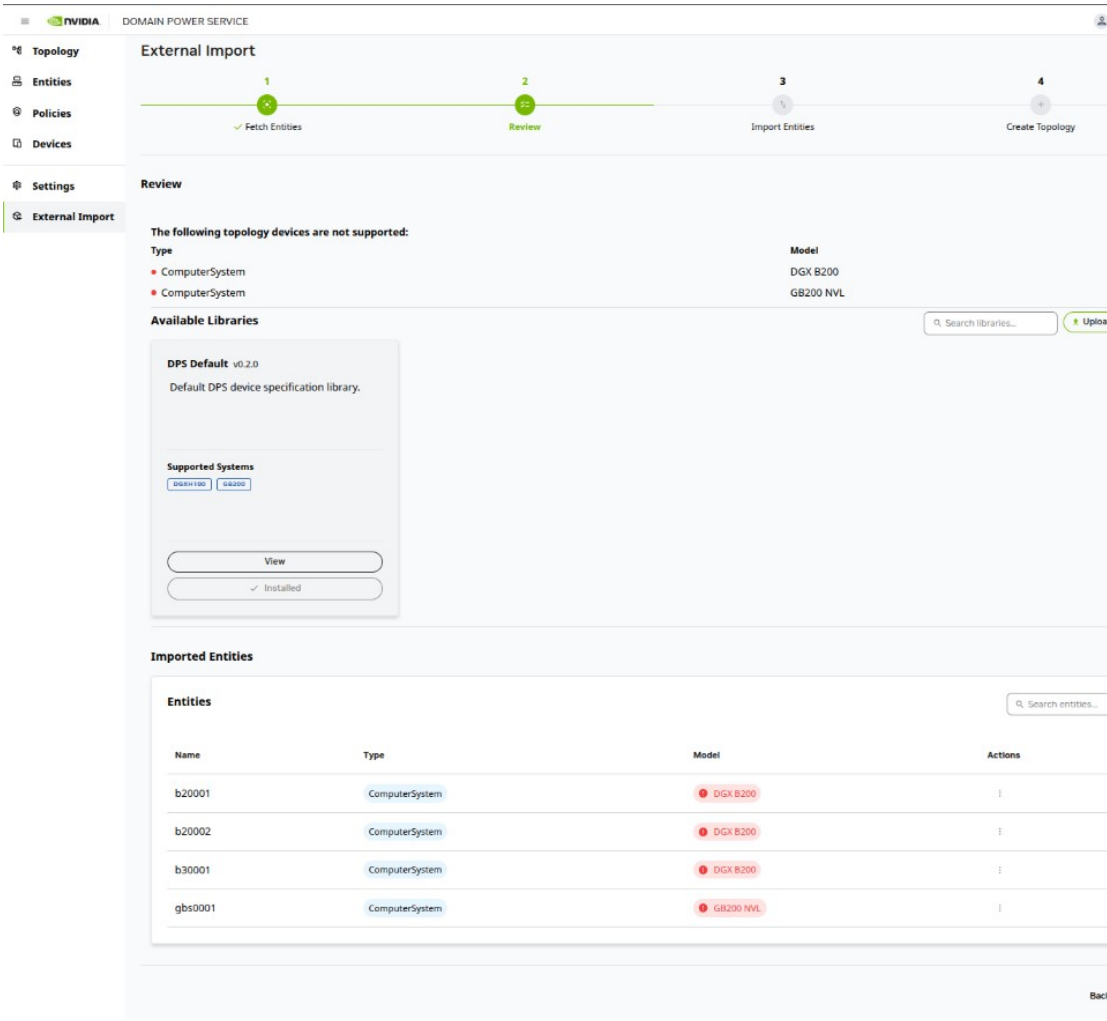
2.5.3.5.1.3 Step 1: Fetch Entities

1. Navigate to **External Import** in the left sidebar.
2. Select **Nautobot** as the source.
3. Add filters to scope the import — locations, racks, device models, or query strings.
4. Click **Fetch**.
5. Select the desired devices from the results.
6. Click **Next**.

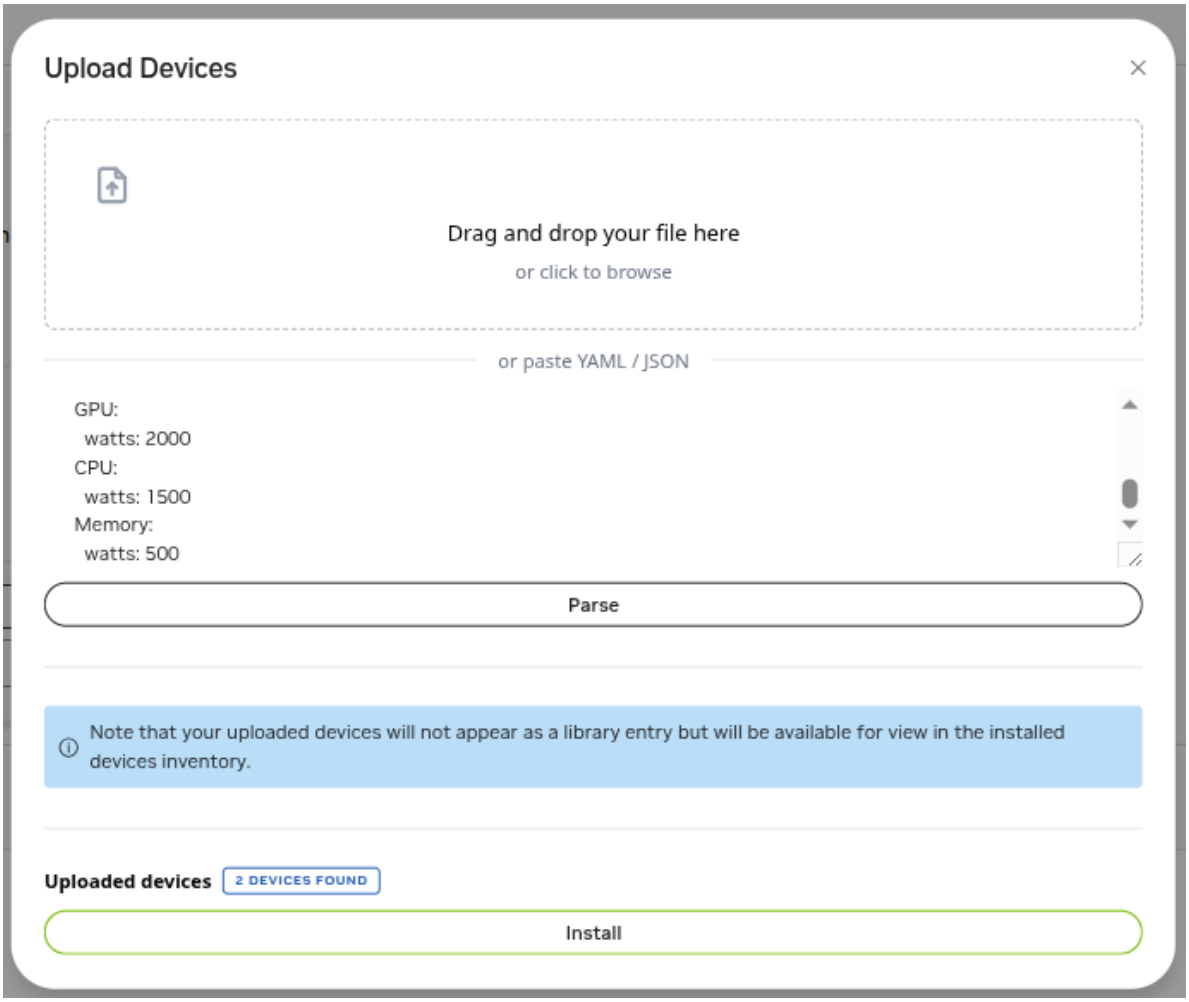


2.5.3.5.1.4 Step 2: Review and Upload Devices

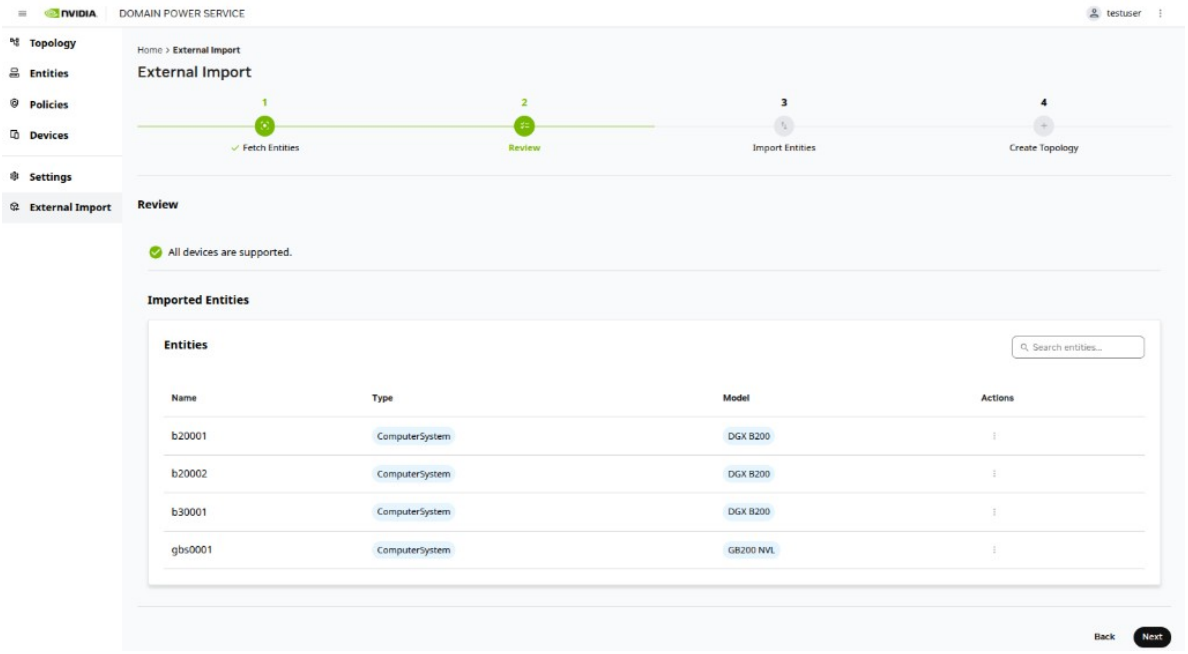
DPS must recognize the device types being imported. Devices with unsupported models appear with their model name highlighted in red. To resolve this, upload the device specifications for any unrecognized models.



1. Click **Upload Devices**.
2. Copy or upload a YAML file containing the device specification.
3. Click **Parse**.
4. Click **Install**. *Displays the upload dialog and resolving device types:*



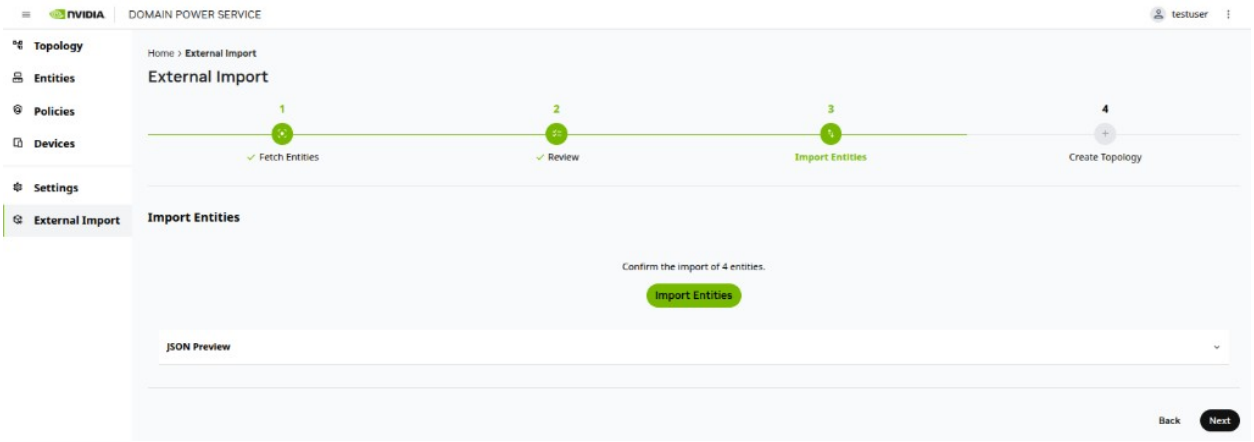
5. Observe devices with previously unsupported models are resolved.



6. Click **Next**.

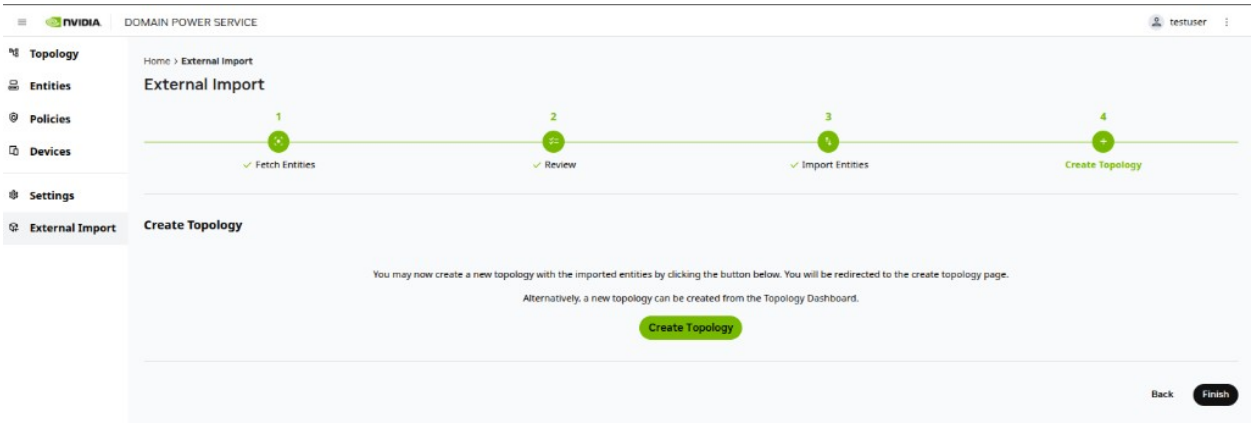
2.5.3.5.1.5 Step 3: Import Entities

1. Click **Import Entities** to confirm the import.
2. Click **Next**.



2.5.3.5.1.6 Step 4: Create Topology

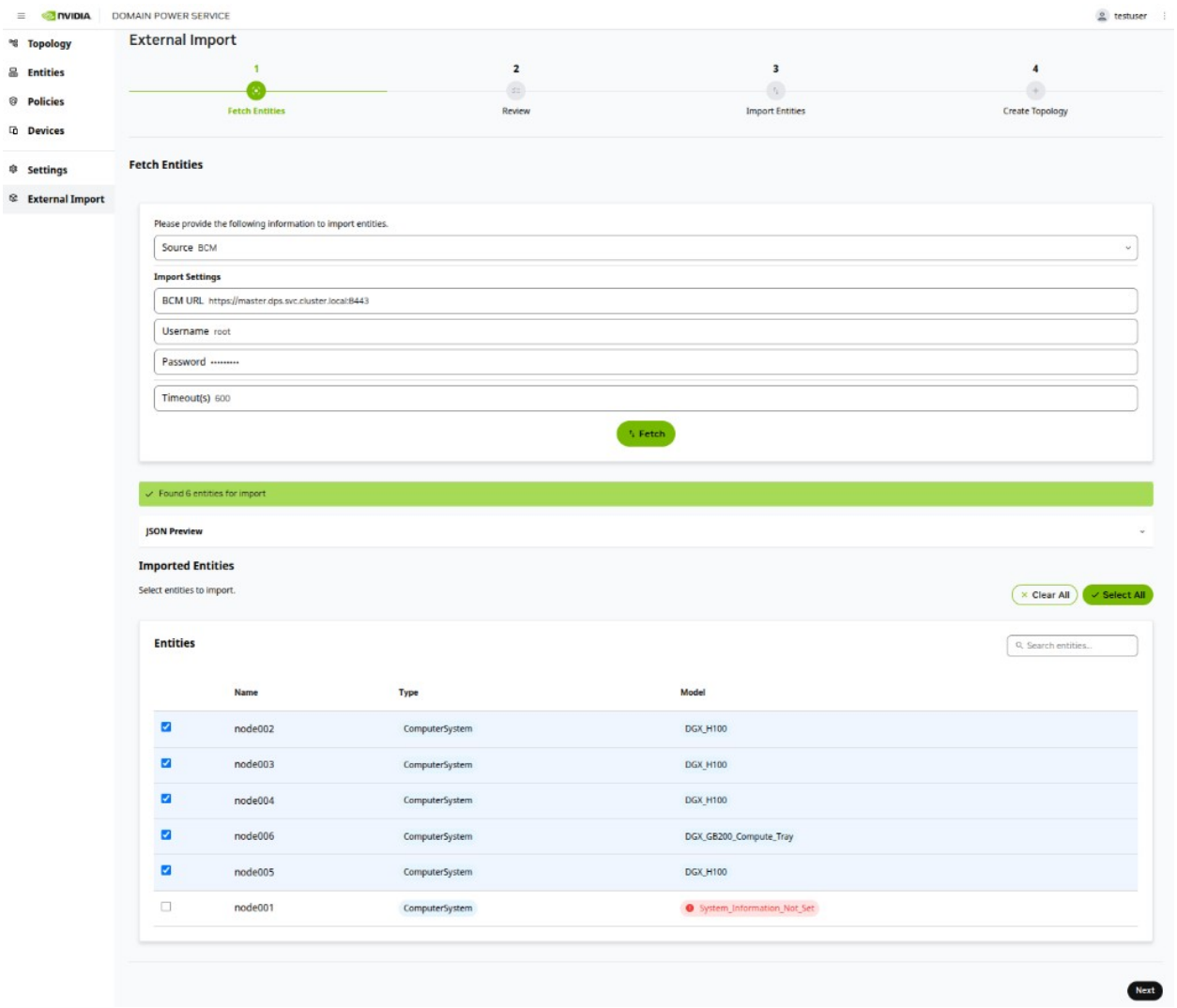
1. Click **Create Topology** to build a topology with the imported entities, or click **Finish** to complete the import without creating a topology.



2.5.3.5.1.7 Importing from BCM

2.5.3.5.1.8 Step 1: Fetch Entities

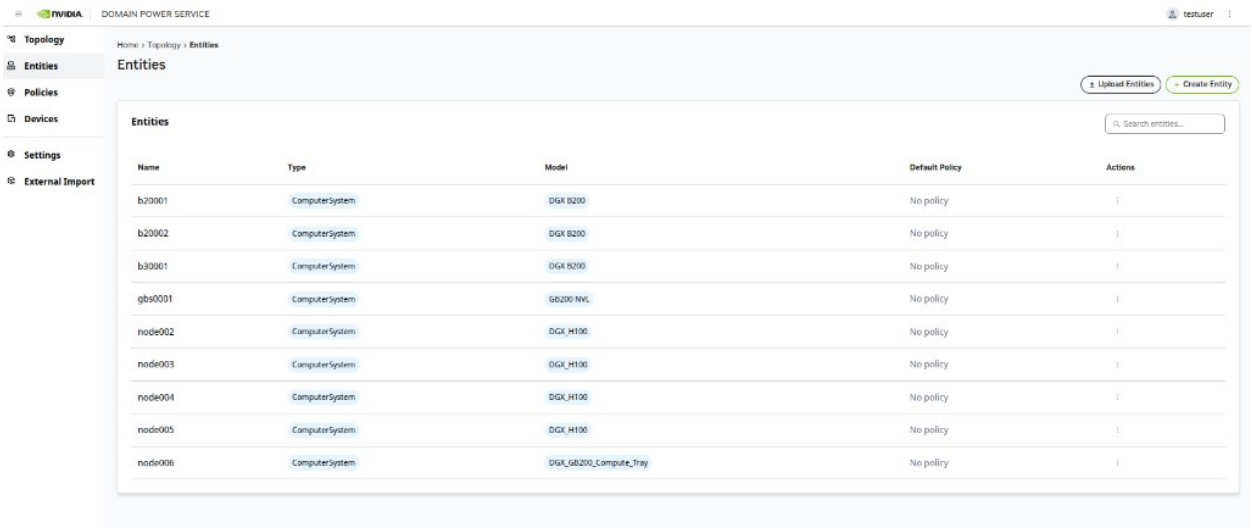
1. Navigate to **External Import** in the left sidebar.
2. Select **BCM** as the source.
3. Enter the BCM URL, username, and password.
4. Click **Fetch**.



The remaining steps (Review and Upload Devices, Import Entities, and Create Topology) are the same as the Nautobot flow described earlier.

2.5.3.5.1.9 Viewing Entities

After importing, you can view all imported entities by clicking the **Entities** icon in the left sidebar.



2.5.3.5.2 Import a Topology

Use this workflow to load a clean topology from a JSON file into DPS. The procedure covers schema-backed validation, CLI import, direct API composition, and how to confirm that the imported inventory matches the file.

Import registers entities and topology relationships. It does not activate the topology or apply power policies to BMCs.

2.5.3.5.2.1 Prerequisites

Confirm the following before you import:

- The DPS server is running and reachable.
- `dpsctl` is installed and you are authenticated. Refer to *Installing dpsctl*.
- Device specifications for every entity `Type` and `Model` already exist. Refer to *Managing Devices*.
- The environment is a clean install, or you have exported any topology you need to keep. Refer to *export*.
- Typed policy bundles already exist if you will bind a bundle at resource-group creation or through the `CreateUpdateTopology` create request. Create bundles with `dpsctl policy bundle upsert`. Do not embed a top-level `Policies` array in the topology file.

2.5.3.5.2.2 Schema

Topology files validate against the JSON schema embedded in the current DPS release. The published schema reference is generated from that artifact. The schema `$id` is `dcpower_0.1`, a schema document identifier rather than a product version. The gRPC API package is `nvidia.dcpower.v1`.

Refer to *Schema Reference* for the schema and downloadable JSON.

Each file is a JSON object with these top-level members:

- `Entities` — optional array of data center entity definitions
- `Topology` — required for import; name plus parent-child power relationships

- `Policies` — retained in the schema for compatibility. A non-empty array is rejected by `dpsctl topology validate` and `dpsctl topology import`

The topology must contain at least one topology entity. Entity `Name` values are unique keys. Every name referenced in `Topology.Entities` must appear in the file or already exist in the inventory.

Power-domain caps use `OperatingLimit`, not `Constraints`. The Go topology model does not load `Constraints` as a typed power cap. The extra field can pass schema validation and remain in the entity's untyped JSON, but it does not configure operating-limit behavior.

The entity `Policy` string is a legacy field. Typed policy activation binds a named bundle on the topology (`CreateUpdateTopologyRequest.CreateRequest.policy_bundle`) or on a resource group (`--policy-bundle`). Upserting a bundle that happens to share an entity `Policy` name does not apply that bundle.

The topology JSON model does not serialize a topology-level policy bundle (`PolicyBundle` is omitted from JSON). `dpsctl topology import` therefore cannot bind a bundle from the file. The server resolves an empty create value to the current `global_policy_bundle` setting and persists that selection. Direct API clients can set the create RPC `policy_bundle` field explicitly. Resource groups select their bundle separately with `--policy-bundle`.

2.5.3.5.2.3 Example File

The following file defines one power domain and one compute node. It omits `Policies`.

```
{
  "Entities": [
    {
      "Type": "PowerDomain",
      "Name": "PD-A",
      "OperatingLimit": {
        "PowerValue": {"Value": 1150000, "Type": "W"},
        "PowerFactor": 0.9
      }
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_H100",
      "Name": "node001",
      "Redfish": {
        "URL": "https://node001-bmc.example.com",
        "SecretName": "node001"
      }
    }
  ],
  "Topology": {
    "Name": "pdn",
    "Entities": [
      {
        "Name": "PD-A",
        "Children": ["node001"]
      },
      {
        "Name": "node001"
      }
```

(continues on next page)

(continued from previous page)

```
    ]
  }
}
```

For the file model and relationship rules, refer to *Topologies*.

2.5.3.5.2.4 Validate Before Import

Validation does not write entities or topologies. Use it to catch errors before you mutate the database.

`dpsctl topology validate` runs two layers:

1. Local schema validation against the embedded JSON schema. Syntax errors report file, line, column, and offset. Schema errors report instance location and a sanitized message.
2. Server semantic validation through the `TopologyManagementService.ValidateTopology` stream. The server combines the uploaded model with existing inventory and checks names, device models, connectivity, cycles, and disconnected entities. Closing the stream starts validation. The call does not modify the database.

```
dpsctl topology validate topology.json
```

Successful server validation returns:

```
{
  "status": {
    "ok": true,
    "diag_msg": "Topology validation passed"
  }
}
```

If semantic validation fails, `status.ok` is `false`, `diag_msg` reports the error count, and `validation_errors` lists each object name, object type, error code, and message. Server error codes are snake_case values such as `device_not_found`, `invalid_model`, and `circular_dependency`.

A non-empty `Policies` array fails before the server call:

```
legacy topology policies are not supported; create policy bundles with dpsctl
↪policy bundle upsert
```

Command reference: *validate*.

2.5.3.5.2.5 Import with dpsctl

Go `dpsctl` is the operator CLI. Import is client-side orchestration, not a single `ImportTopology` RPC.

```
dpsctl topology import topology.json
```

Successful import returns:

```
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

The CLI performs these steps:

1. Reads and schema-validates the JSON file.
2. Rejects a missing topology, a topology with no entities, or a non-empty `Policies` array.
3. Calls `ListTopologies` and fails if the topology name already exists. This check is not a lock; a concurrent create can still collide on the server.
4. Calls `GetEntities` and `CreateEntities` for entity names that are not already present. Existing names are left unchanged.
5. Streams `CreateUpdateTopology` with a create operation. The created topology is inactive.

If entity creation succeeds and topology creation then fails, the new entities remain. Fix the file or inventory, then retry. Import does not roll back earlier writes.

Command reference: *import*.

2.5.3.5.2.6 Import through the API

API consumers compose import from multiple RPCs. Do not send the CLI JSON file as a single RPC payload.

The release-supported service is `nvidia.dcpower.v1.TopologyManagementService`. Refer to *APIs*.

The following table lists the recommended sequence:

Step	RPC	Mutates State	Role
1	`ValidateTopology` (client stream)	No	Schema-valid payload plus semantic checks against existing inventory
2	`ListTopologies`	No	Detect a topology-name collision before creating entities
3	`GetEntities`	No	Detect which entity names already exist
4	`CreateEntities` (client stream)	Yes	Create only missing entities; error if any streamed name already exists
5	`CreateUpdateTopology` (client stream)	Yes	Create the named topology. The first message is a create request. Referenced entities must already exist. Set `CreateUpdateTopologyRequest.CreateRequest.policy_bundle` to bind a typed bundle. The new topology is inactive
6	`ListTopologies` or `GetTopologyModel`	No	Inspect names, `is_active`, leaf nodes, and exported model

`dpsctl topology import` performs local schema validation but does not call `ValidateTopology` before it starts the mutation sequence. The server performs semantic validation when it receives

`CreateUpdateTopology`. Direct API clients should call `ValidateTopology` first to detect semantic errors before creating inventory entities.

`UpsertEntities` is retained for compatibility and returns `UNIMPLEMENTED` without changing state.

The Python `api.topology_import.import_from_json` helper follows the create-missing-entities then create-topology sequence and rejects a nonempty `Policies` array. Unlike the Go CLI, it does not call `ListTopologies` before entity creation, so check for a topology-name collision separately. The helper does not set `CreateUpdateTopologyRequest.CreateRequest.policy_bundle`. The libdpsapi topology import path uses the same policy-array rejection.

Python import composition

```
import json

from dpsapi import DpsApi

api = DpsApi(
    dps_host="api.dps.example.com",
    dps_port=443,
).with_username("alice").with_password("<password>")

with open("bundle.json", encoding="utf-8") as fh:
    api.policies.upsert_bundle(json.load(fh))

with open("topology.json", encoding="utf-8") as fh:
    topology_data = json.load(fh)

topology = topology_data["Topology"]
validation = api.topology.validate_topology(
    topology_name=topology["Name"],
    entities=topology_data.get("Entities"),
    topology_entities=topology.get("Entities"),
)
print(validation)

imported = api.topology_import.import_from_json(topology_data)
print(imported)

listed = api.topology.list_topologies(is_active=False)
print(listed)
```

After authentication, list topologies through `TopologyManagementService`:

```
grpcurl -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
  api.dps.your.domain:443 \
  nvidia.dcpower.v1.TopologyManagementService/ListTopologies
```

2.5.3.5.2.7 Verify the Import

Compare the loaded inventory with the input file. Import success does not mean the topology is active.

```
dpsctl topology list
dpsctl topology list --active=false
```

(continues on next page)

(continued from previous page)

```
dpsctl topology list --active=true
dpsctl topology list-entities
dpsctl topology export --topology pdn
```

Confirm the following:

- `topology list` includes the imported name.
- `dpsctl topology list --active=false` lists the topology.
- `dpsctl topology list --active=true` does not list it. JSON output omits `is_active` when the value is `false`.
- Leaf node names match the compute entities in the file.
- `list-entities` returns every imported `Name`, `Type`, and `Model`.
- `export` hierarchy and entity identities match the input. Export sanitization omits the runtime-only `topologyHash`.

Command references: *list*, *list-entities*, and *export*.

2.5.3.5.2.8 Activation Is Separate

Power policy application and BMC connectivity checks run during activation, not import. Deactivate the currently active topology before you activate another. The `--replace-topology` flag is retained for compatibility and is currently rejected.

```
dpsctl topology activate --topology pdn
```

Refer to *activate* and *Managing Topologies*.

2.5.3.5.2.9 Troubleshoot and Recover

The following table maps common failures to a corrective action:

Symptom	Likely Cause	Action
Invalid JSON syntax at line, column, offset	Malformed file	Fix the reported location and re-run validate
`topology schema validation failed` with instance paths	Field type, required member, or uniqueness	Correct the named instance path against the schema
`legacy topology policies are not supported`	Non-empty top-level `Policies` array	Remove `Policies`. Upsert typed bundles with `dpsctl policy bundle upsert`
`no topology in the input file` or `topology has no entities defined`	Missing `Topology` or empty `Topology.Entities`	Add a named topology with at least one entity
`failed to create topology` during import preflight	Topology name already exists	Choose a new name, or use the API-only `CreateUpdateTopology` update operation. Refer to *update*
`device_not_found` in `validation_errors`	Missing device specification	Register the device, then validate again
`circular_dependency`, `disconnected_graph`, or `referenced_entity_not_found`	Invalid hierarchy	Fix `Children` references and unique `Name` values
Entity create succeeded, topology create failed	Partial write	Inspect `list-entities`, fix the topology, then retry import or remove unused entities
Authentication or connectivity errors	Session, TLS, or endpoint	Re-authenticate and confirm `--host` and `--port`

Validate remains safe to retry. Import can encounter partial prior writes after entity creation.

2.5.3.5.2.10 Next Steps

After the imported topology matches the file and remains inactive:

1. Activate it when you are ready to apply policies. Refer to *Managing Topologies*.
2. Create resource groups for workloads. Refer to *Managing Resource Groups*.

2.5.3.5.3 Managing Devices

2.5.3.5.3.1 Overview

This guide provides step-by-step instructions for creating device specifications in DPS. Device specifications define the power characteristics and management capabilities of datacenter equipment and must be created before defining entities and topologies.

For more information about device specifications, see *Device Specifications*.

2.5.3.5.3.2 Prerequisites

- DPS server running and accessible
- `dpsctl` installed and authenticated through *Configure Server Authentication*
- Understanding of your hardware's power characteristics

2.5.3.5.3.3 Step 1: Create Device Specification YAML File

Device specifications are defined in YAML format. Create a file (e.g., `devices.yaml`) with your hardware specifications.

2.5.3.5.3.4 Basic Structure

```yaml
- type: DeviceType
  model: DeviceModel  # Optional for some types
  spec:
    # Power characteristics
    minLoadWatts: 100
    maxLoadWatts: 1000
    efficiencyFactor: 0.95
    # Management integration
    powerPolicyPlugin: "PluginName"  # e.g., DGX_H100, DGX_B200
```

2.5.3.5.3.5 Example: Complete Device Registry

```yaml
# Power infrastructure devices
- type: PowerDomain
  spec:
    efficiencyFactor: 0.95

- type: PowerDistribution
  model: RackPDU95_57500W
  spec:
    maxLoadWatts: 57500
    efficiencyFactor: 0.95

- type: PowerSupply
  model: PowerSupply95_3300W
  spec:
    maxLoadWatts: 3300
    efficiencyFactor: 0.95

# Compute devices
- type: ComputerSystem
  description: NVIDIA GB200 Compute Tray (Bianca)
  model: DGX_GB200
  spec:
    devices:
      - type: CPU
        model: Grace
        count: 2
```

(continues on next page)

(continued from previous page)

```
      - type: GPU
        model: GB200
        count: 4
    minLoadWatts: 1000
    maxLoadWatts: 6667
    processorModulesCount: 2

- type: GPU
  description: NVIDIA GB200 GPU
  model: GB200
  spec:
    minLoadWatts: 200
    maxLoadWatts: 1200
    wppsSupport: true

- type: CPU
  description: NVIDIA Grace CPU
  model: Grace
  spec:
    minLoadWatts: 50
    maxLoadWatts: 420
```

2.5.3.5.3.6 Step 2: Import Device Specifications

Use `dpsctl device upsert` to import your device specifications:

```
# Import device specifications
dpsctl device upsert devices.yaml
```

Expected Output:

```
{
  "status": {
    "diag_msg": "Success",
    "ok": true
  }
}
```

2.5.3.5.3.7 Step 3: Verify Import

List all registered devices to confirm successful import:

```
# List all device specifications
dpsctl device list
```

Expected Output:

```
[
  {
    "type": "PowerDomain",
    "description": "Generic power domain",
    "Spec": {
```

(continues on next page)

(continued from previous page)

```
      "PowerDomain": {
        "efficiency_factor": 0.95
      }
    }
  },
  {
    "type": "ComputerSystem",
    "model": "DGX_GB200",
    "description": "NVIDIA GB200 Compute Tray (Bianca)",
    "Spec": {
      "ComputerSystem": {
        "devices": [
          {
            "type": "CPU",
            "model": "Grace",
            "count": 2
          },
          {
            "type": "GPU",
            "model": "GB200",
            "count": 4
          }
        ],
        "min_load_watts": 900,
        "max_load_watts": 5640,
        "processor_modules_count": 2
      }
    }
  }
]
```

2.5.3.5.3.8 Step 4: Update Existing Devices

To modify existing device specifications, edit your YAML file and run the same import command:

```
# Update existing devices (upsert operation)
dpsctl device upsert devices.yaml
```

The `upsert` operation will:

- **Insert** new devices that don't exist
- **Update** existing devices with the same type and model

2.5.3.5.3.9 Troubleshooting

2.5.3.5.3.10 Common Errors

Invalid YAML Syntax:

```
Error: yaml: line 5: mapping values are not allowed in this context
```

- Solution: Check YAML indentation and syntax

Missing Required Fields:

```
Error: device specification missing required field 'spec'
```

- Solution: Ensure all devices have a `spec` section

Invalid Power Values:

```
Error: maxLoadWatts must be greater than minLoadWatts
```

- Solution: Verify power value relationships are correct

2.5.3.5.3.11 Validation

Before importing, validate your YAML file:

```
# Check YAML syntax
yaml-validator devices.yaml

# Test import in dry-run mode (if available)
dpsctl device upsert devices.yaml --dry-run
```

2.5.3.5.3.12 Next Steps

After creating device specifications:

1. ***Manage Topologies*** - Connect entities together
2. ***Manage Resource Groups*** - Set up workload power management

2.5.3.5.4 Managing Topologies

2.5.3.5.4.1 Overview

This guide covers the command-line workflow for validating, importing, and activating a topology. For the topology data model, refer to *Topologies*.

2.5.3.5.4.2 Prerequisites

- DPS server running and accessible.
- `dpsctl` installed and authenticated through *Configure Server Authentication*.
- Required device specifications available. Refer to *Managing Devices*.

2.5.3.5.4.3 Step 1: Prepare the Topology JSON File

A topology import file contains entity definitions and topology relationships. Do not put policy definitions or policy assignments in this file.

Example topology import

```
{
  "Entities": [
    {
```

(continues on next page)

(continued from previous page)

```
      "Type": "PowerDomain",
      "Name": "PD-A",
      "Constraints": {
        "PowerValue": {
          "Value": 1150000,
          "Type": "W"
        },
        "PowerFactor": 0.9
      }
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_GB200",
      "Name": "node001",
      "Redfish": {
        "URL": "https://node001-bmc.example.com",
        "SecretName": "node001"
      }
    }
  ],
  "Topology": {
    "Name": "pdn",
    "Entities": [
      {
        "Name": "PD-A",
        "Children": ["node001"]
      },
      {
        "Name": "node001"
      }
    ]
  }
}
```

If you enabled a DPS Agent, assign managed devices to it in `Topology.Entities`. An `AgentId` assigns the entity and is inherited by descendants that do not set their own value. For example, set it on a shared parent:

```
{
  "Name": "PD-A",
  "AgentId": "data-hall-a-agent",
  "Children": ["node001"]
}
```

Set `AgentId` to the `agent.agentID` configured for that data hall.

A nonempty top-level `Policies` array is rejected. A legacy `Policy` value on an entity does not assign a policy during activation.

2.5.3.5.4.4 Step 2: Validate the File

Validate the topology before importing it:

```
dpsctl topology validate topology.json
```

Resolve schema, reference, duplicate-name, cycle, and connectivity errors before continuing.

2.5.3.5.4.5 Step 3: Select the Policy Bundle

The Go CLI topology file has no policy-bundle field. Before importing, store any custom bundle and set the global default that the new topology should use:

```
dpsctl policy bundle upsert site-balanced.yaml
dpsctl policy bundle list
dpsctl settings update --set global_policy_bundle=site-balanced
```

Use an installed bundle name, and omit the `upsert` command when selecting a built-in bundle. An API client can instead provide an explicit `policy_bundle` in the topology create request.

2.5.3.5.4.6 Step 4: Import the Topology

Import the validated file:

```
dpsctl topology import topology.json
```

Import creates missing inventory entities and a new inactive topology. An existing topology with the same name causes the import to fail; use the update command for that topology. Import does not apply a policy to hardware.

The Go import request leaves `CreateUpdateTopologyRequest.policy_bundle` empty, so the server resolves and persists the current `global_policy_bundle` as the topology selection.

For default selection and lifecycle behavior, refer to *Power Policies*.

2.5.3.5.4.7 Step 5: List Topologies

List stored topologies and review the selected policy bundle and active state:

```
dpsctl topology list
```

2.5.3.5.4.8 Step 6: Activate a Topology

Only one topology can be active. If another topology is active, deactivate it before activating the candidate:

```
dpsctl topology activate --topology pdn
```

Activation applies the topology-wide effective policy to matching power-managed devices. Without an effective policy, activation invokes the device unset path, which can apply an idle policy or a plugin-specific fallback.

Host reachability checks are enabled by default. To set the minimum acceptable reachability percentage explicitly:

```
dpsctl topology activate \
  --topology pdn \
  --ping-hosts \
  --at-least-percent-hosts 80
```

2.5.3.5.4.9 Step 7: Deactivate a Topology

Deactivate the topology before removing it or activating a different one:

```
dpsctl topology deactivate --topology pdn
```

Deactivation marks the topology inactive and performs best-effort cleanup on managed devices. A successful response does not confirm that every BMC accepted the cleanup request, so verify the resulting power limits.

For details about idle policies, BMC defaults, and service outages, refer to *Topology Deactivation and Loss of DPS Control*.

2.5.3.5.4.10 Prepare Nodes for Maintenance

Before you power down a managed node, restart its BMC, or update its firmware, remove the node from its active resource group. Removing a node from an active resource group reapplies the topology default policy, so the node remains power-managed by the active topology. After maintenance, add the node back to the resource group to reapply its resource-group policy.

For maintenance on selected nodes in an active resource group, use the following procedure:

1. Remove the affected nodes from the active resource group:

   ```
   dpsctl resource-group remove \
     --resource-group <resource-group> \
     --entities <comma-separated-node-list>
   ```

2. Complete the maintenance and confirm that each BMC is reachable.
3. Add the nodes back to the active resource group. Adding a node to an active resource group applies its policy immediately:

   ```
   dpsctl resource-group add \
     --resource-group <resource-group> \
     --entities <comma-separated-node-list> \
     --strict-policy \
     --allow-reprovision=false
   ```

4. Confirm the resource-group state and the effective node limits:

   ```
   dpsctl resource-group list
   dpsctl check status --nodes <comma-separated-node-list>
   ```

For topology-wide maintenance, record the configuration of every active resource group, delete those groups, and deactivate the topology. After maintenance, activate the topology, recreate the resource groups, and activate them to reapply their policies.

DPS does not provide a standalone `resource-group deactivate` command. Deleting an active resource group deactivates it as part of the delete operation.

2.5.3.5.4.11 Step 8: Update a Topology

After editing the topology JSON, update the stored topology:

```
dpsctl topology update updated-topology.json
```

Use `--force` only to bypass the optimistic topology-hash conflict check. Schema and server validation still run.

This Go command preserves the topology's current policy-bundle selection. API clients can explicitly change the selection only while the topology is inactive.

2.5.3.5.4.12 Step 9: Remove a Topology

A topology must be inactive before removal:

```
dpsctl topology remove --topology pdn
```

2.5.3.5.4.13 Step 10: Manage Inventory Entities

List all inventory entities or request specific names:

```
dpsctl topology list-entities
dpsctl topology list-entities node001
```

Remove entities only when active topology and resource-group state no longer depends on them:

```
dpsctl topology remove-entities node001 node002
```

2.5.3.5.4.14 Next Steps

- *Managing Resource Groups*
- *dpsctl check Usage Guide*
- *Power Policies*

2.5.3.5.5 Run a BMC Health Check

Use `dpsctl verify bmc-health` to test baseboard management controller (BMC) reachability and collect a structured health report for an imported topology. Depending on the device plugin and selected mode, the check can also exercise firmware inventory, power-limit controls, telemetry, Electrical Design Point (EDPp), and Workload Power Profile Settings (WPPS). It compares in-band and out-of-band power-limit values where supported.

For every command and flag, refer to the *`dpsctl verify bmc-health` reference*.

2.5.3.5.5.1 Prerequisites

Before you start, confirm the following conditions:

- *`dpsctl`* is installed, authenticated, and aligned with the DPS server version.
- The topology is validated and imported. Refer to *Managing Topologies*. It does not need to be active.

- Every intended node belongs to the topology and has a configured power plugin and BMC credential Secret. Refer to *Credentials and Secrets Configuration* for credential setup.
- No other BMC health check is running on the server. The server permits one check at a time, even when two checks target different topologies or nodes.
- You know whether the result is for diagnosis, a readiness gate, or an approved write-enabled validation. These uses require different evidence.

Start with a small node scope when you are validating a new device model, credential configuration, or operator procedure.

2.5.3.5.5.2 Choose a Mode

The current server default is read-only when the request does not specify a write mode. Pass a mode flag anyway so saved commands and automation state the intended behavior explicitly.

Selection	Behavior
`--skip-write`	Reads health data without power-limit write cycles or WPPS remediation. Use this mode first and for general diagnosis.
No mode flag	Uses the server default, which is currently read-only. This behavior can change between versions.
`--force-write`	Requests power-limit write and readback cycles and WPPS remediation from a device plugin that supports write-enabled deep checks. Use only after the write-safety preflight.

`--skip-writes` and `--force-writes` are mutually exclusive.

2.5.3.5.5.3 Run a Read-Only Check

Run a blocking summary check when you need the overall result and issue list:

```
dpsctl verify bmc-health start \
  --topology <topology-name> \
  --skip-writes \
  --wait \
  --summary-only
```

The command prints progress to standard error and the report to standard output. Without `--prs-bmc-latency-check`, a completed report with `status.ok: false` is still printed, and the command exits with status 1. With the PRS latency check, the CLI prints the BMC report followed by the PRS artifact, and the final exit status reflects the PRS artifact. Inspect the BMC report's `status.ok` value separately.

Use the asynchronous flow when the check can outlive your terminal session or when you need the full report:

```
dpsctl verify bmc-health start --topology <topology-name> --skip-writes
dpsctl verify bmc-health status <task-id>
dpsctl verify bmc-health report <task-id> --wait --output json > bmc-health-
↪report.json
```

2.5.3.5.5.4 Recover Interrupted Polling

Save the `task_id` from `start`. The task runs on the server after `dpsctl` disconnects. A local interruption, including Ctrl-C during `--wait`, stops only polling. Reattach with `status` or `report --wait`.

The server retains a completed task for the configured `--zombie-lifetime`, which defaults to `10m`. A server restart discards both running and completed task state. There is no `dpsctl verify bmc-health cancel` command.

2.5.3.5.5.5 Run an Approved Write-Enabled Check

`--force-writes` can change BMC state. Use it only with applicable operator approval and an approved procedure that defines the maintenance window, recovery action, and acceptable node scope. The flag does not override a device plugin's safety policy, suspend effective DPS policy, or disable other power controllers. Some plugins remain read-only. If the selected device plugin does not support the deep BMC health check, DPS checks only BMC reachability and firmware.

Complete this preflight before you use write mode:

1. Place the selected nodes in an idle or otherwise approved maintenance state.
2. Review the active and effective policy and its hardware and hierarchical constraints for the selected scope. If enforcement can reapply or redistribute power during the probe, quiesce it through the approved procedure or account for that behavior in the result.
3. Prevent other operators and power-management controllers from changing power limits or WPPS on the selected nodes during the check.
4. Run and retain a full read-only report as the baseline.
5. Confirm that you can restore the captured power limits through the approved operational path if automated restoration fails.
6. Confirm that write-enabled WPPS remediation is acceptable for the selected scope. The probe does not restore WPPS profiles that it disables.

Start with one node and one in-flight node probe:

```
dpsctl verify bmc-health start \
  --topology <topology-name> \
  --nodes <node-name> \
  --force-writes \
  --concurrency 1 \
  --wait
```

For each supported GPU and CPU component, the write-cycle probe performs the following actions:

1. Reads and records the original power-limit set point.
2. Moves the set point in 1 `W` steps, up to three steps away from the original while staying within the reported bounds.
3. Reads after every write until the requested value appears or the `--write-resolution-timeout` expires.
4. Walks the set point back and issues a final best-effort write to the captured original value.
5. Reads the value again and records whether restoration succeeded.

The default readback window is `1s`. Increase `--write-resolution-timeout` only when observed BMC commit latency justifies it. A larger value does not correct an unreachable or stuck BMC.

When a device plugin runs write-enabled deep checks and finds active WPPS profiles, it can disable them and verifies that the enforced mask becomes empty. A `WPPS_PROFILES_ACTIVE` issue can remain in the same run because the profiles were active when observed.

Warning

Treat `POWER_LIMIT_RESTORE_FAILED` as an unresolved state change. Keep the node out of service, compare `original_w` with `restored_value_w`, and restore and verify the limit through the approved recovery procedure before returning the node to use.

2.5.3.5.5.6 Set Sampling and Timeouts

The following defaults affect run duration and BMC load:

Option	Current Behavior
`--samples-per-teleme`	`0` uses the server default of 10 samples per node. The server accepts at most 2,500.
`--telemetry-interval`	The CLI default is `500ms` between samples. This option does not control power-write readback.
`--write-resolution-t`	The default is `1s` per write readback, with a server maximum of `60s`.
`--per-node-timeout`	The default is `5m`. An explicit value must be from `5m` through `48h`.
`--concurrency`	`0` uses the server default of 16 concurrent node probes. The server maximum is 1,024.
`--expected-edpp-pct`	An unset value uses the server floor of `100`. An explicit value must be from 1 through 200.

When you increase the sample count or interval, set `--per-node-timeout` high enough for all requested samples plus firmware, power, EDPp, WPPS, and write phases. A node that reaches this timeout is reported as `BMC_UNREACHABLE`, and a later response does not replace the timeout result.

Lower concurrency when the fleet has tight BMC session or connection limits. Higher concurrency increases simultaneous BMC and agent load; it does not change the server-wide limit of one BMC health task.

2.5.3.5.5.7 Interpret the Report

Read the report from the outside in:

Field	Interpretation
`status`	The overall result. `ok: false` means at least one error-severity issue or failed node. Warnings alone can leave `ok: true`.
`cluster_summary`	Scope, pass/fail counts, unreachable nodes, and cluster latency distributions. Confirm the counts match the intended topology or node subset.
`issues`	Actionable findings. Use `severity`, `code`, `node`, `resource`, `observed`, and `threshold` together. Do not automate against the human-readable message alone.
`node_reports`	Per-node evidence, including reachability, firmware, endpoint statistics, power settings and writes, EDPp, WPPS, drift, and telemetry. This field is omitted by `--summary-only`.
`endpoint_stats`	Attempt, success, failure, and latency counts for each probed endpoint.
`telemetry_samples` and `telemetry_samples_completed`	Requested and attempted telemetry reads. Also inspect `telemetry_completion_reason`.
`power_writes`	Write-cycle evidence. Require `mismatches: 0`, `restored: true`, and `restored_value_w` equal to `original_w`.

The JSON formatter omits fields that have their default values. For example, a clean report can omit an empty `issues` list and zero-valued `failed`, `nodes_unreachable`, and `mismatches` fields. An omitted false-valued WPPS field such as `reset_attempted` also means `false`. Do not apply this rule to an entire evidence section: an omitted `power_writes` list means that the report contains no write-cycle evidence.

The report does not identify the requested write mode. The presence of `power_writes` confirms that power write cycles ran, but their absence does not distinguish `--skip-writes`, a device plugin that remained read-only, and a plugin without write support. Record the exact command options with the report. When those options show `--skip-writes`, `power_writes` must be absent. When they show `--force-writes` for a plugin expected to support write-enabled deep checks, require `power_writes` evidence or treat the requested check as not tested. Older server versions can also omit newer telemetry completion fields; align the client and server versions when your acceptance criteria require those fields.

A device plugin without deep BMC health support falls back to BMC reachability and firmware validation. The final cluster report does not expose the internal deep-probe marker. If expected `node_reports` fields are absent, treat those checks as not tested instead of inferring that they passed.

The probe also treats some data as optional. For example, a missing in-band one-shot power value prevents that drift comparison but does not by itself create an issue. A clean `status` is therefore not a substitute for confirming that the report contains every field required by your readiness criteria.

2.5.3.5.5.8 Apply Safety Criteria

For a full readiness gate, confirm all of the following conditions:

- `status.ok` is `true`.
- `cluster_summary.total_nodes` and `nodes_attempted` match the intended scope, and `failed` and `nodes_unreachable` are `0`.
- The report has no error-severity issues. Review and disposition every warning even when the process exits with status `0`.
- Every required node contains evidence for the checks that its device model is expected to support.
- Telemetry sampling completed. Failed individual samples produce `TELEMETRY_SAMPLES_FAILED`; an incomplete sampling loop produces `TELEMETRY_SAMPLING_INCOMPLETE`.
- Average telemetry, power GET, power SET, and EDPp GET latency do not exceed `5s`, and maximum telemetry latency does not exceed `10s`.
- EDPp is at or above the selected floor, WPPS is accessible with no active profiles, and required in-band and out-of-band values do not drift.
- For a write-enabled gate on a plugin that supports write-enabled deep checks, every write cycle has zero mismatches and verified restoration. A second run confirms that any WPPS remediation left the node in the intended state.

The built-in thresholds are coarse product checks. Apply any stricter release, platform, or site acceptance criteria separately.

2.5.3.5.5.9 Handle Expected Outcomes and Failures

Use the following table to decide the next action:

Outcome	Meaning and Action
`start` returns `ALREADY_EXISTS`	Another server-wide check is active. The public error does not provide its task ID. Use a previously saved ID if you own the run; otherwise, coordinate with the operator who started it or wait for it to finish.
`report` returns `done: false`	The task is still running. The command can exit successfully because no health decision exists yet. Use `status` or `report --wait`.
`status` is `Failed` or the report has `error_message`	The task failed before it produced a usable report. Preserve the task ID and inspect the DPS server logs.
`status` or `report` returns `NOT_FOUND`	The ID is invalid, the completed task expired, or the server restarted. For a write-enabled run, verify node state before starting again.
`BMC_UNREACHABLE`	The BMC, agent, device registration, power plugin, dispatch, or per-node timeout path failed. Start with *BMC or Redfish Calls Fail*.
`POWER_WRITE_READBACK_MISMATCH`	A written value did not converge before the readback window. Keep the node out of the readiness set and investigate the write path.
`POWER_LIMIT_RESTORE_FAILED`	Automated restoration was not verified. Apply the approved recovery procedure before another run or workload.
`WPPS_PROFILES_ACTIVE`	Profiles were active at probe time. Read-only mode leaves them unchanged; a device plugin running write-enabled deep checks attempts to disable them. Review the recorded profile IDs and rerun after remediation.
A completed report has `status.ok: false`	Without `--prs-bmc-latency-check`, `start --wait` and a completed `report` print the report, then exit with status 1. Use the structured report as the decision input.
`--prs-bmc-latency-check` is enabled	The CLI prints the BMC report followed by the PRS artifact. The final exit status reflects the PRS artifact, so inspect the BMC report's `status.ok` value separately.
The report contains warnings only	The command can exit with status `0`. Review warnings such as `IB_OOB_LIMIT_DRIFT` and `TELEMETRY_SAMPLES_FAILED`; do not rely on the exit status alone.
The server returns `NOT_IMPLEMENTED` or an unknown BMC-health method	The server can predate the client command. Align `dpsctl` and the server to the same release before retrying.

2.5.3.5.5.10 Preserve and Share Evidence Safely

Retain the full JSON report in an access-controlled location long enough to review per-node evidence. Before sharing it outside the operational team, remove or replace node names, firmware inventory, topology identifiers, and other site-specific details that the recipient does not need. Never add BMC

credentials, tokens, private keys, or Secret contents to the report.

Record the client and server versions, topology scope, exact options, task ID, start and completion times, overall status, issue dispositions, and the approved recovery result for any state-changing failure. Follow the evidence and publication process established for your release or acceptance program; this guide does not define a separate publication system.

For pilot-specific issue remediation, refer to *BMC Health Check*.

2.5.3.5.6 DPS UI Operator Guide

Use the DPS UI to inspect configuration and operational state, create inactive topology candidates, manage policy bundles, and prepare resource groups for activation. Dynamic Power Software remains authoritative for supported configuration, installed policy definitions, and the outcome of every write.

The examples and images in this guide use a fixed set of locally generated test data. Names such as `east-hall`, `guide-topology`, and `e2e-operator` are synthetic. They are not customer or production data.

2.5.3.5.6.1 Before You Begin

Sign in with the authentication method configured for your deployment. LDAP deployments use the DPS username and password form. OpenID Connect (OIDC) deployments redirect you to the identity provider. Refer to *Configure Server Authentication* for server and UI setup. Do not include credentials, secrets, OAuth callback values, browser-session data, or access tokens in screenshots, logs, diagnostic files, or other materials you share for troubleshooting.

The API authorizes each request through DPS or the configured authorization proxy. Signing in and finding an available control does not guarantee that its request is permitted. Disabled controls can also reflect deployment capabilities or the current DPS state. If a request is denied, resolve the displayed error or ask your administrator to check the configured access requirements.

Use the safety conventions in this table throughout the guide.

Label	Meaning	Operator Action
Read-only	Retrieves or displays DPS data.	Verify the selected topology, resource group, or task.
Write	Changes a stored object or configuration.	Review the object, scope, and confirmation text.
Guarded operation	Can affect lifecycle state, capacity, or another operator's configuration.	Confirm the operational plan and recovery path before acknowledging the action.

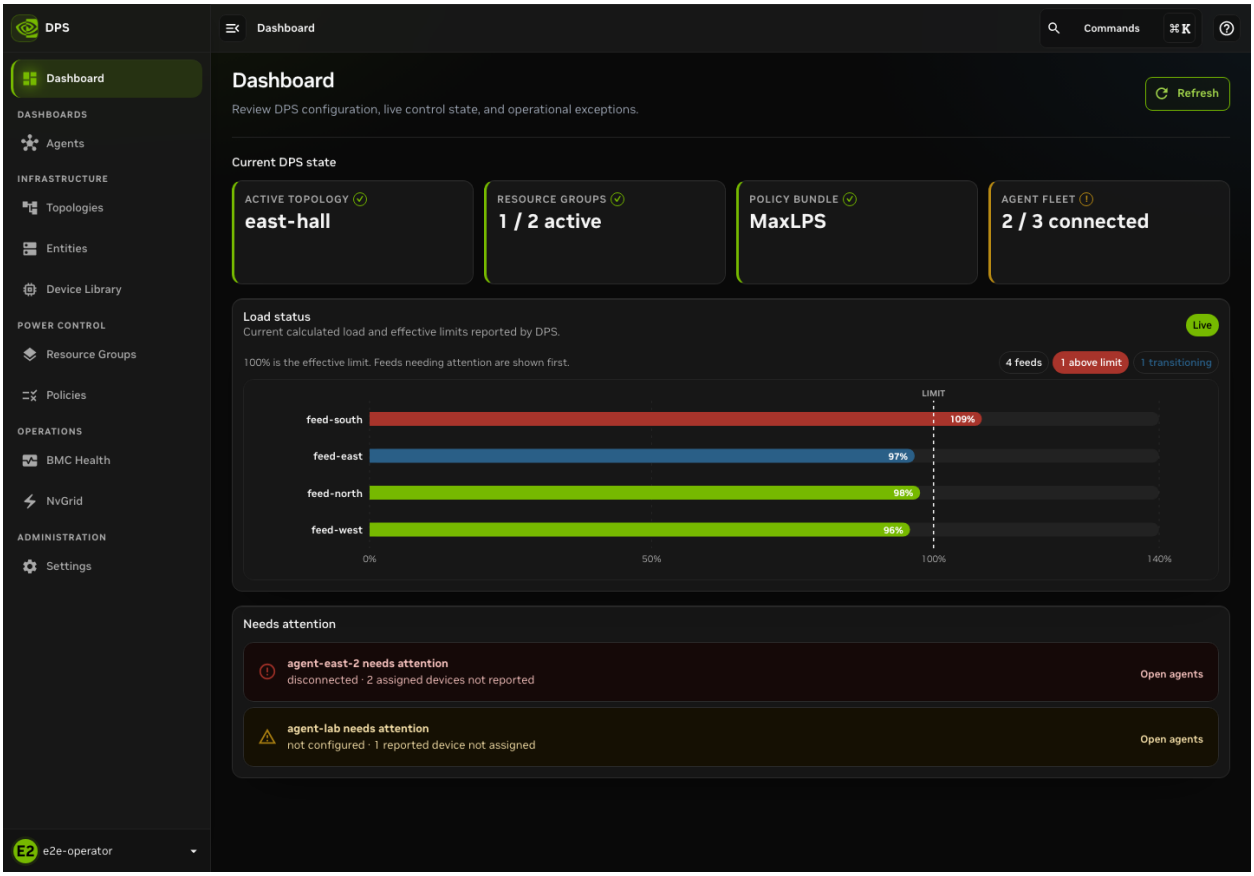
Warning

Validation checks the request that DPS receives at that time. A later edit invalidates the result. Successful validation does not make an import or activation safe by itself.

The workflows in this guide are independent. Use **On this page** to navigate directly to the workflow you need.

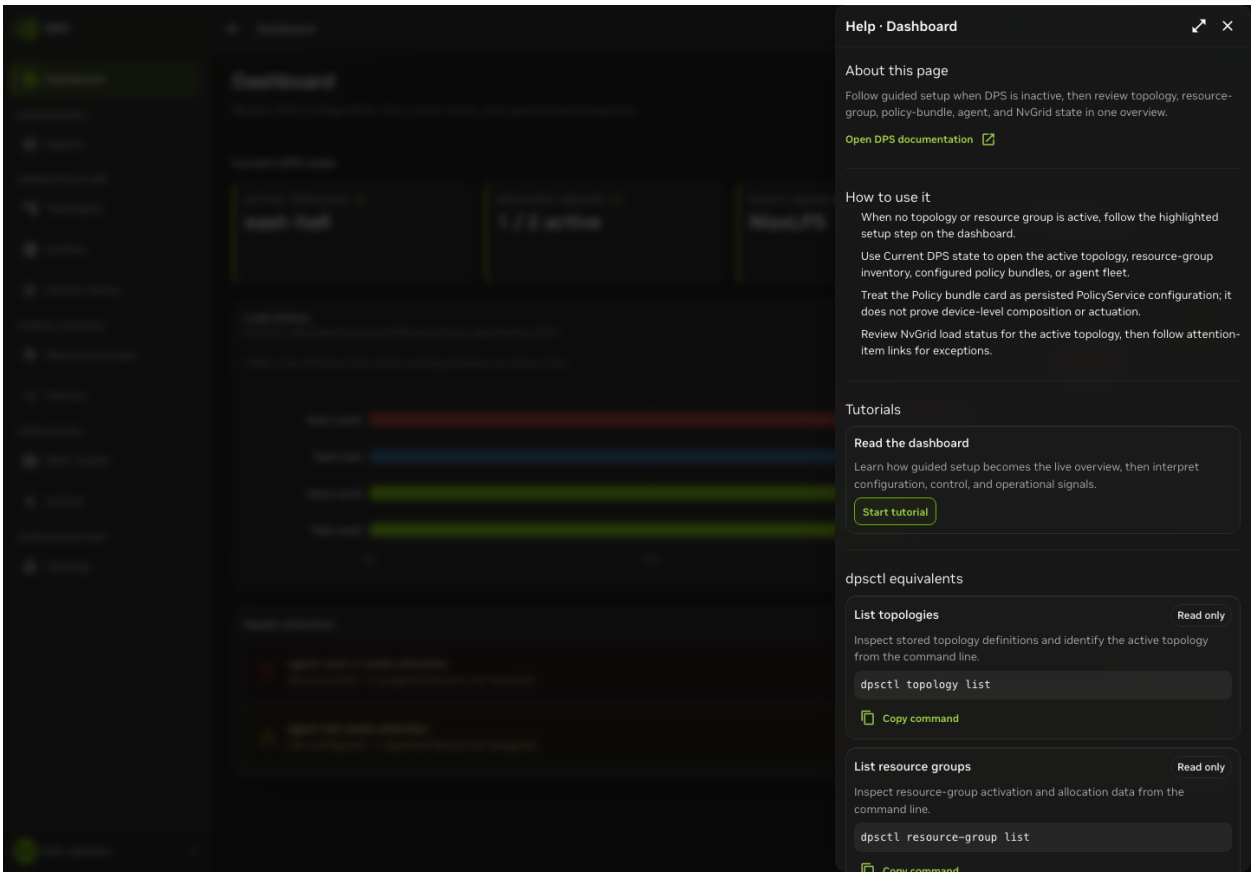
2.5.3.5.6.2 Orient Yourself in the UI

After you sign in, use **Dashboard** as the operator starting point. The left navigation groups infrastructure, power control, operations, and administration views. Use the dashboard to identify the active topology, data-quality signals, and links to detailed views. Dashboard values are the latest reported UI view; they do not replace a successful configuration, activation, or task result.



Dashboard and primary navigation using synthetic data.

Every authenticated page that includes contextual guidance provides **Help for this page** in the top bar. The drawer describes the page, supplies safe-use notes, and can open a read-only in-product tutorial. It highlights controls but does not perform actions for you.



Page Help changes with the route. Open it before you use an unfamiliar page.

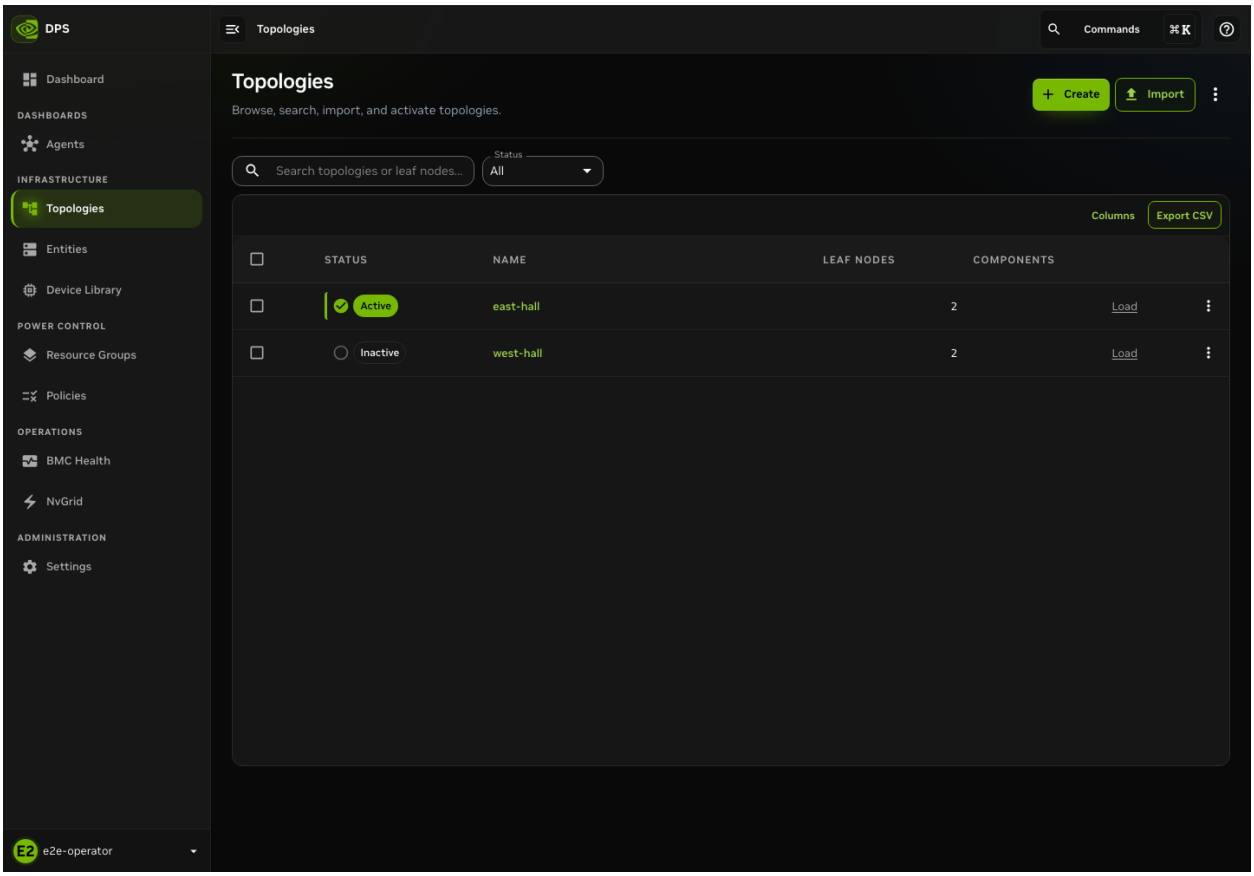
2.5.3.5.6.3 Create, Validate, and Inspect a Topology

A topology models the hierarchy of power domains and compute resources. Build and inspect a candidate while it is inactive. Treat activation as a separate, reviewed change.

2.5.3.5.6.4 Start from the Catalog

Open **Topologies** and identify the active topology before you take action. Search and filter the catalog. The active topology is the one currently used in production. Inactive rows are suitable candidates for inspection, editing, and a later activation decision.

Select **Create** to start a local draft. Select **Clone** when a saved topology is the correct starting point. Cloning changes a local copy, not the source topology. Use **Import** for a canonical JSON manifest that already exists outside the builder.



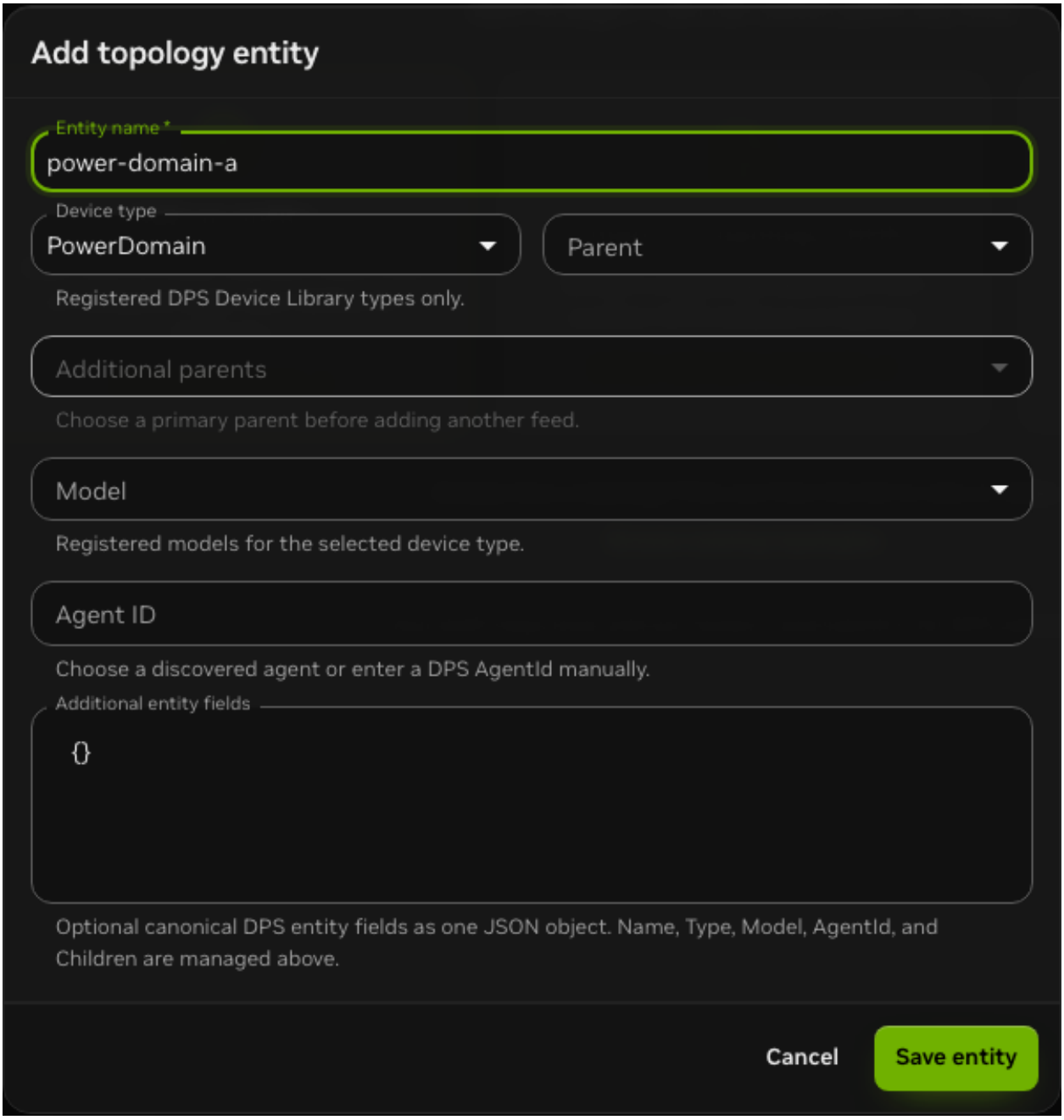
Identify the active topology before you select a candidate row.

2.5.3.5.6.5 Create a Power Domain

The **Topology Builder** is a local workspace. It does not write to DPS until you import the later manifest. The draft remains in the browser, which makes it suitable for review, but it is not a shared server-side record.

To create a power domain at the root of an NVL72 rack topology, complete these steps.

1. Select **Add entity**.
2. Select **PowerDomain** as the root type and enter a unique **Entity name**. This example uses `power-domain-a`.
3. Select a registered model if the entity type requires one, and then select **Save entity**.
4. Enter a clear, unique **Topology name**. Add **Topology properties** only as a valid JSON object.
5. Review the summary chips and **Readiness** panel. They report entity, root, compute, leaf, and depth counts, and identify issues that block validation.

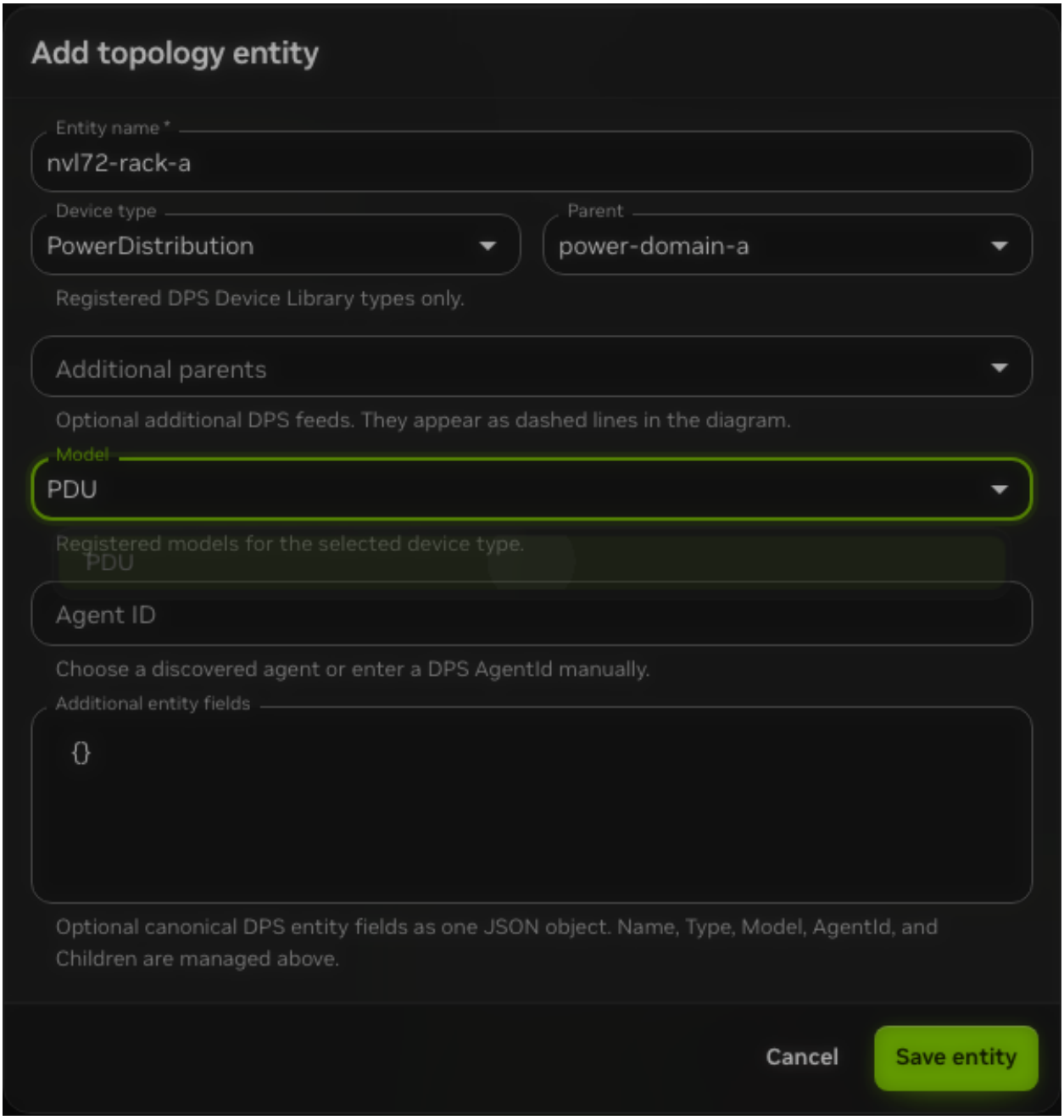


Create `power-domain-a` as the `PowerDomain` root for the NVL72 example.

2.5.3.5.6.6 Add a Rack

Select **+ Child** on `power-domain-a` in the single-line diagram. The entity editor opens with `power-domain-a` selected as **Parent**. Set **Device type** to **PowerDistribution**, select the registered PDU **Model**, name the child `nvl72-rack-a`, confirm its parent, and select **Save entity**.

The rack is a `PowerDistribution` device, not a nested `PowerDomain`. Use the registered Device Library model that represents the rack power-distribution unit. Keep the same parent relationship: `power-domain-a` > `nvl72-rack-a`.



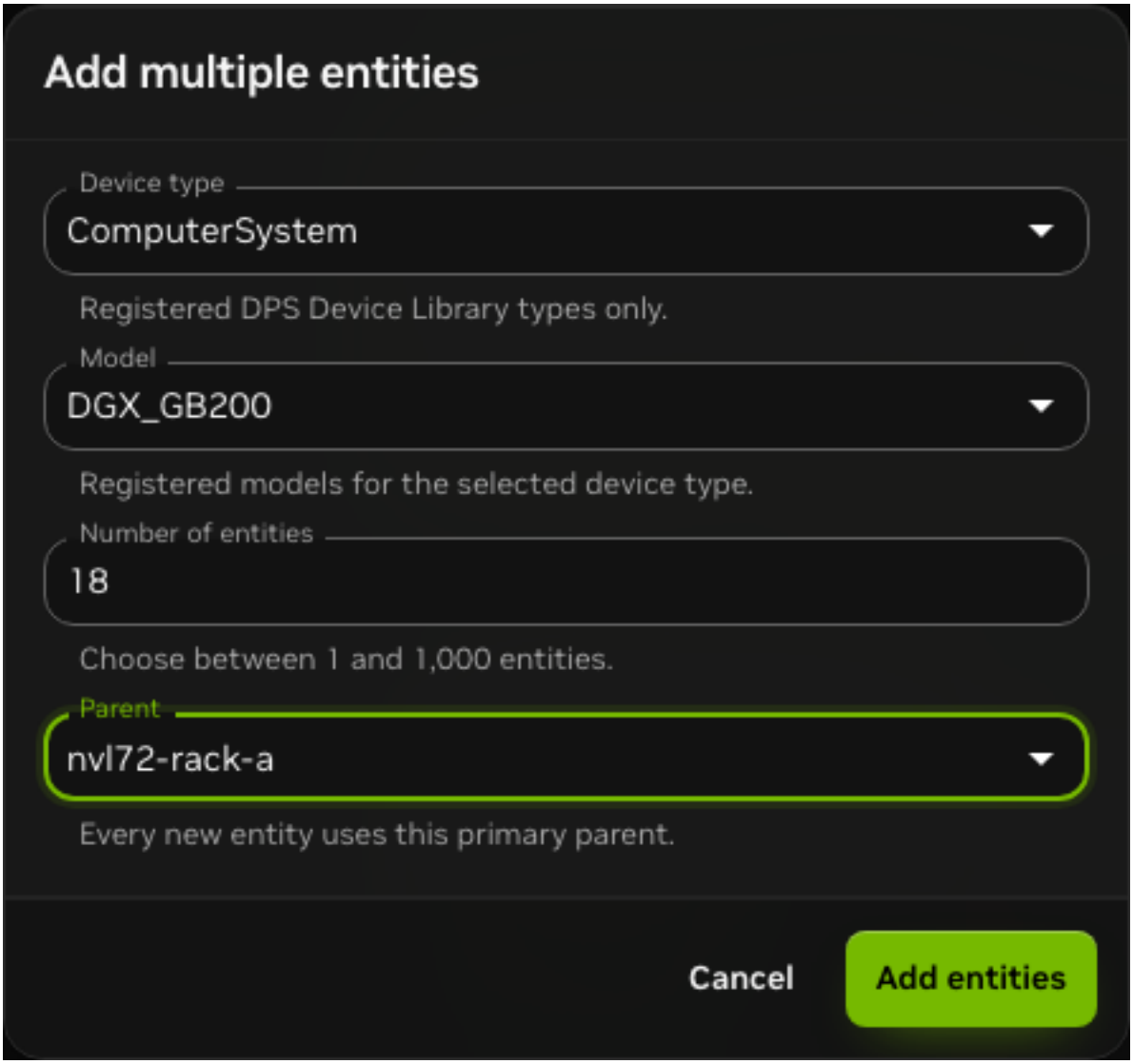
The `nvl72-rack-a` PowerDistribution device uses the PDU model and is parented to `power-domain-a` before you add compute systems.

2.5.3.5.6.7 Add Multiple Entities at Once

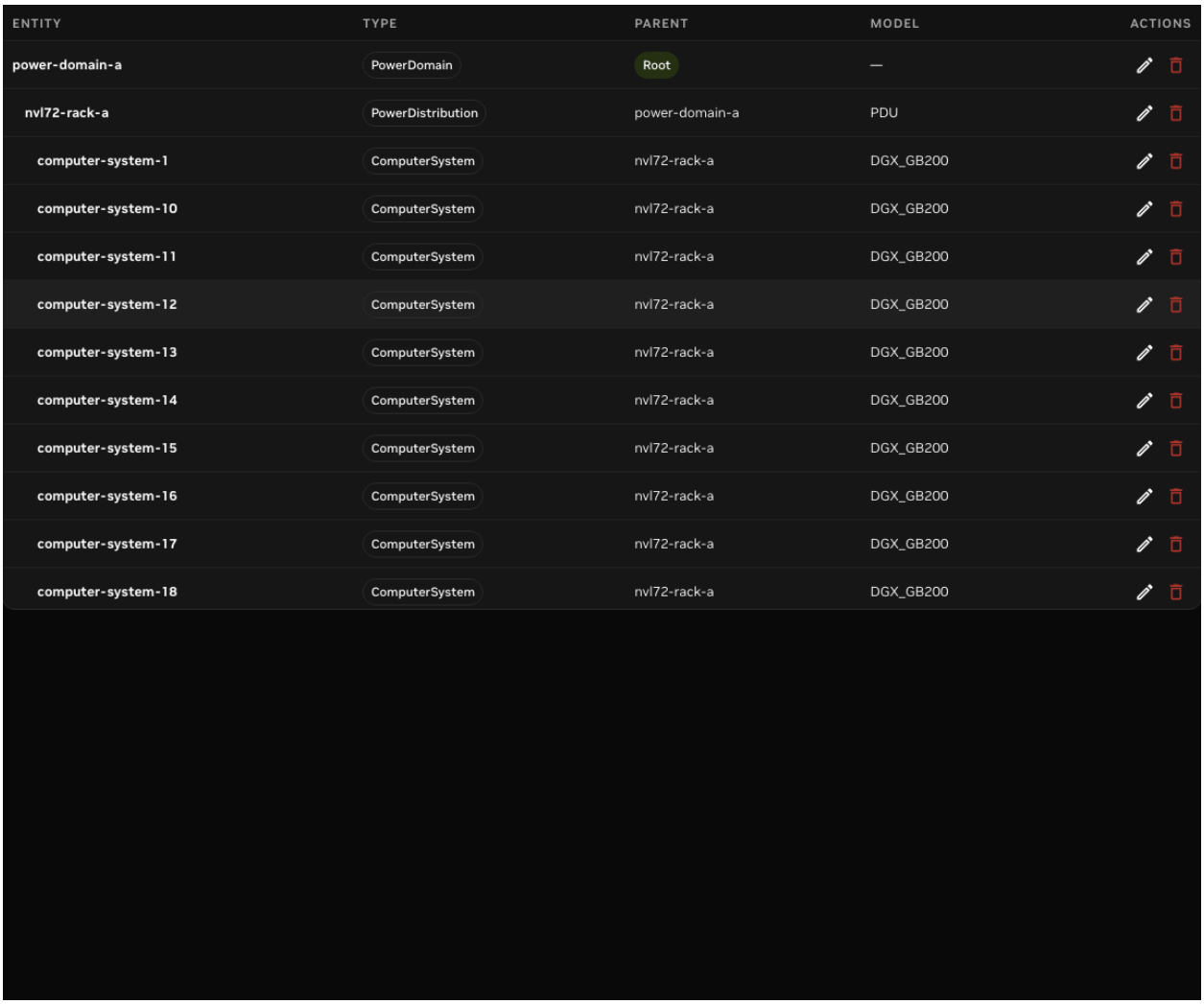
Use **Add multiple** to create uniform systems in a rack without opening a separate editor for each system.

1. Create and save the rack parent. This example uses `nvl72-rack-a`.
2. Select **Add multiple** from the builder actions or **Entity hierarchy**.
3. Set **Device type** to **ComputerSystem**.

4. Select the registered **Model** for every system. This example uses `DGX_GB200`.
5. Enter 18 for **Number of entities**.
6. Set **Parent** to `nvl72-rack-a`. Confirm that the helper text identifies the parent for every new entity.
7. Select **Add entities**. The builder creates `computer-system-1` through `computer-system-18`. Use **Edit** if your naming convention requires more specific names.
8. Confirm the compute count, **Entity hierarchy** parent column, and lines in the single-line diagram before you review the manifest.



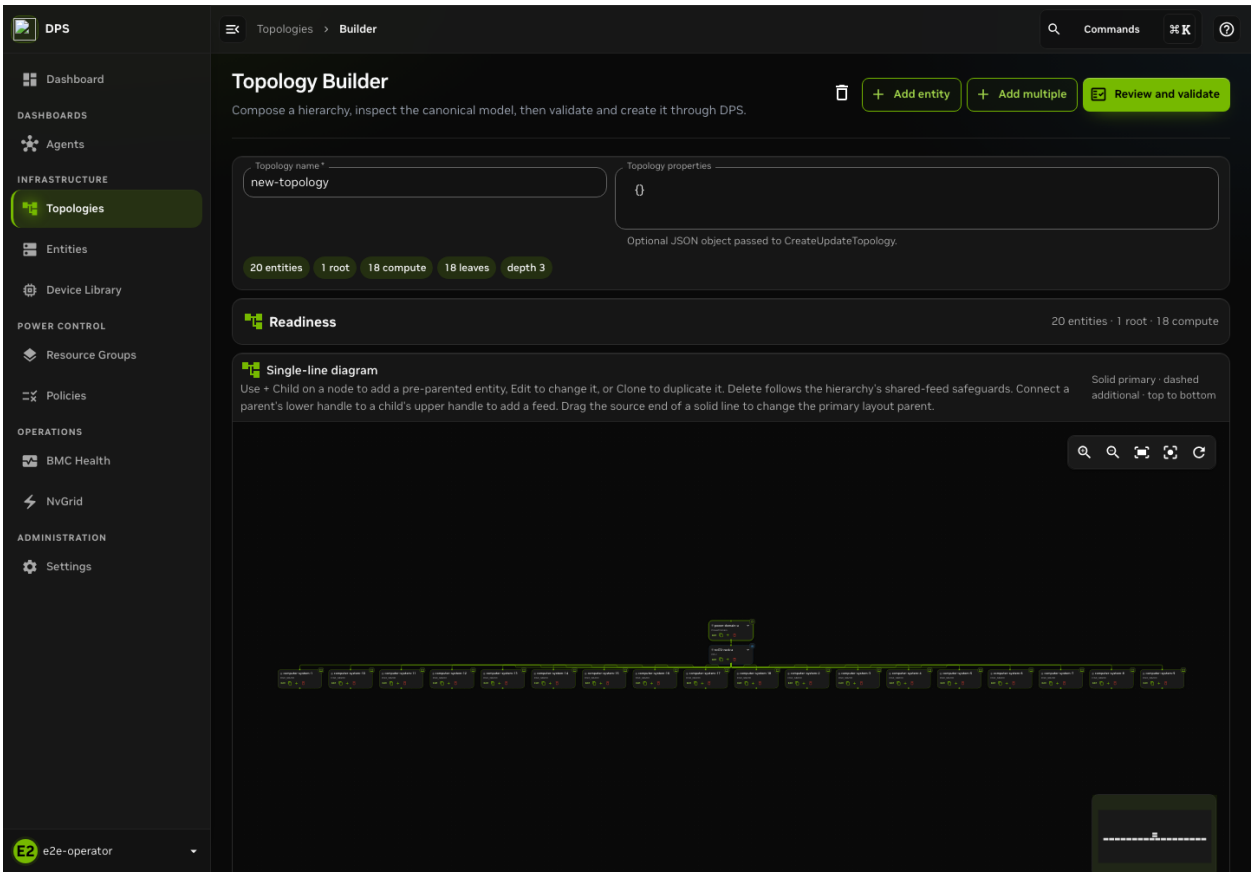
Multi-add configuration for 18 `ComputerSystem` entities with `nvl72-rack-a` as the primary parent.



The entity hierarchy confirms the `PowerDistribution` rack and generated `ComputerSystem` children. `computer-system-18` confirms the full multi-add range.

The single-line diagram and **Entity hierarchy** table show the same local draft. A solid line indicates the primary layout parent. A dashed line indicates an additional feed. Use **Edit** to review type, model, parent, and additional parents. **Fit topology in view** changes only the camera. It does not change the topology model.

Resolve every **Readiness** item before you continue. Common blockers include an empty or duplicate name, an invalid parent relationship, a missing Device Library model, a cycle, and invalid topology-properties JSON. A clean diagram does not validate a topology. **Review and validate** becomes available only when the local draft is ready and the Device Library is available.



Complete local NVL72 example with 20 entities, one root, and 18 compute systems.

2.5.3.5.6.8 Review the Topology

Select **Review and validate**. The builder converts the local draft to a canonical manifest and opens **Import Topology**. DPS has not changed at this point. Review these items before you send anything to the server.

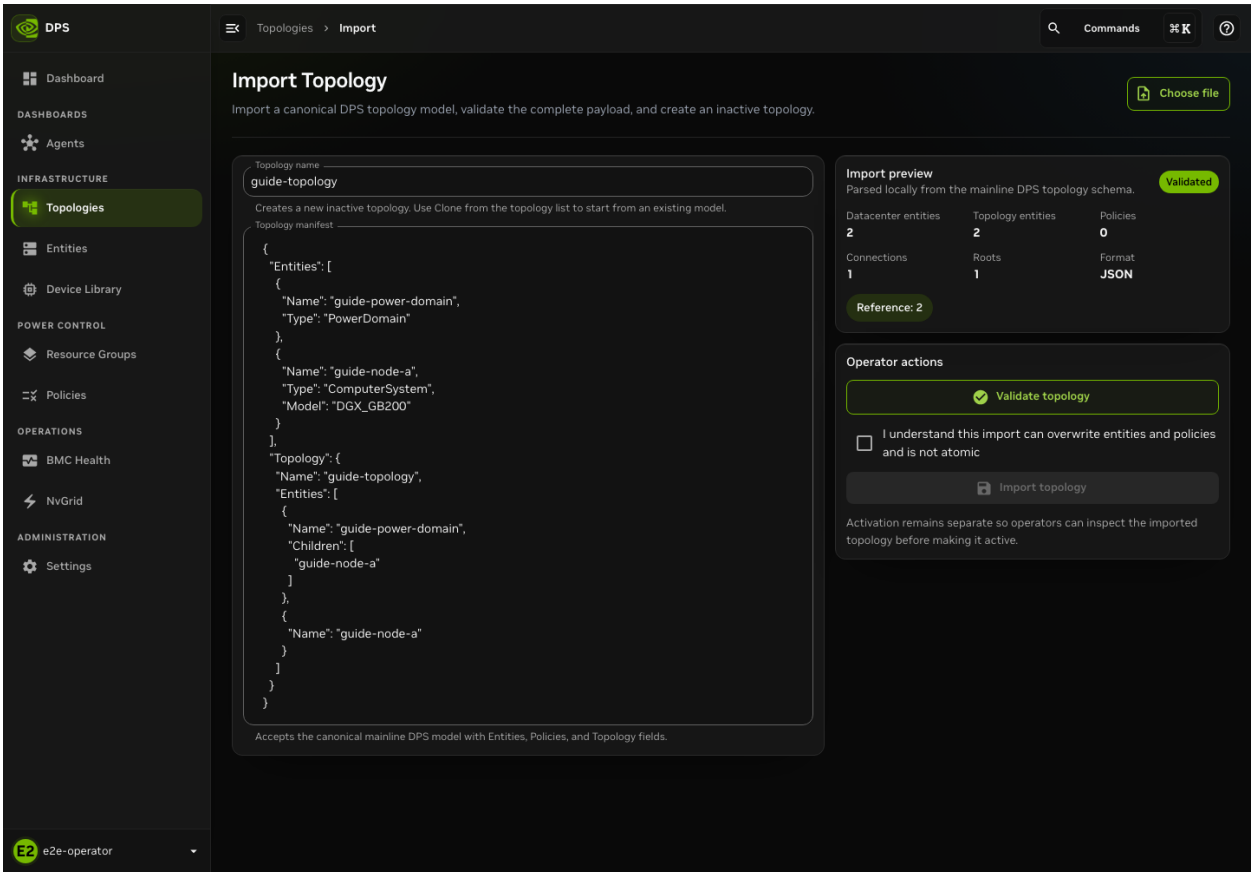
1. Confirm that **Topology name** is the intended name.
2. Confirm that **Import preview** reports the expected counts for data center entities, topology entities, roots, and connections. The policy count must be zero.
3. Confirm that **Topology manifest** contains every entity with the intended type, model, parent and child relationships, and properties.
4. If you change the name or JSON, validate the changed manifest again.

Select **Validate topology** and wait for the manifest to show **Validated**. Validation checks the complete payload at the server. Builder readiness does not replace server validation.

2.5.3.5.6.9 Import Your Built Topology

After validation succeeds, review the **Operator actions** card. A manifest that creates or updates data center entities can complete those writes before a later topology stage fails. The UI acknowledgment can mention policies for compatibility, but current imports must keep the legacy `Policies` array empty; a nonempty array is rejected. Select the acknowledgment only after reviewing the exact manifest, and then select **Import topology**.

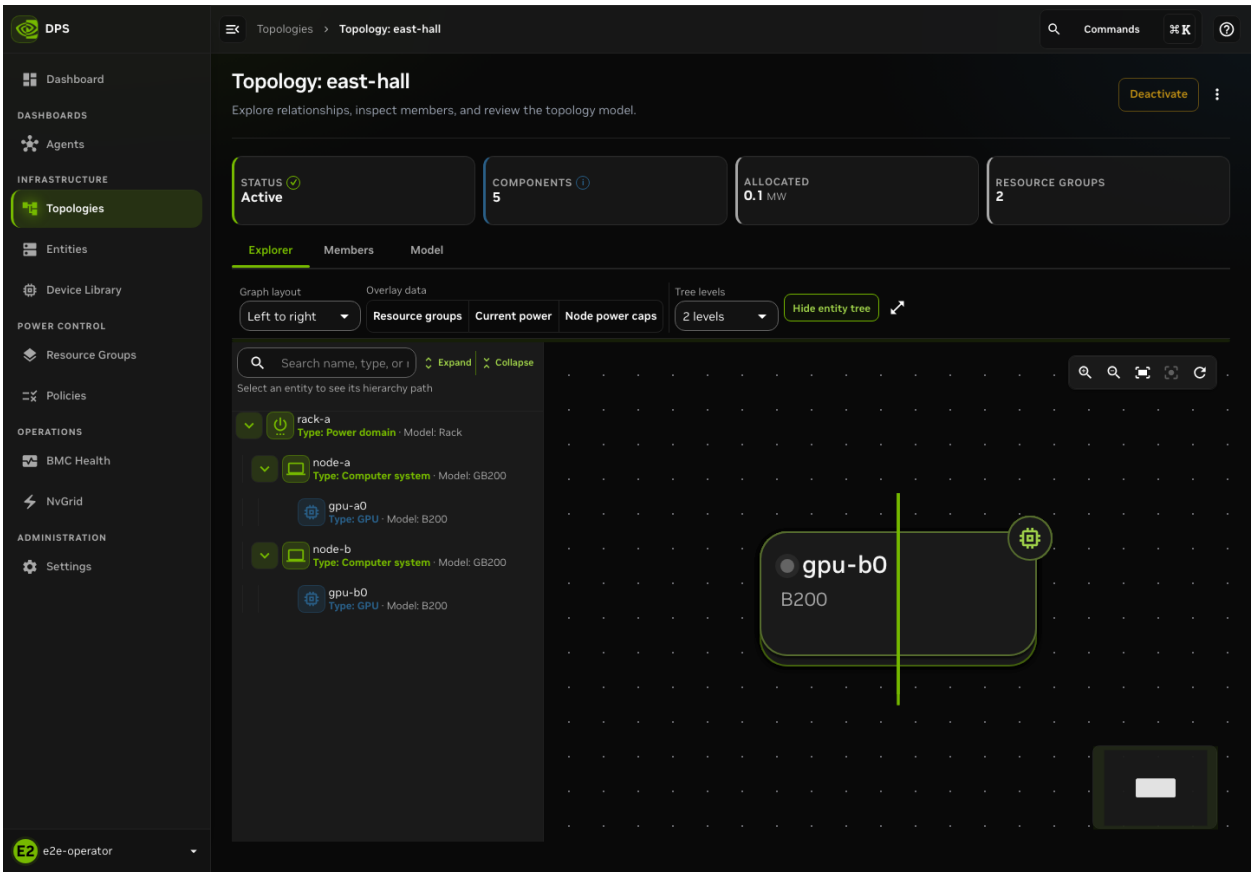
If DPS returns an error, retain the manifest and error details for correction. Do not assume that a failed import fully rolled back. If the import succeeds, it creates an **Inactive** topology. Return to the catalog and confirm the topology name and **Inactive** status before you plan activation.



A validated manifest with the non-atomic write acknowledgment selected. Import remains a separate write.

2.5.3.5.6.10 Inspect a Topology

The import example creates `guide-topology`. Select that topology, or the one you imported, to inspect it. The following screenshot uses the separate `east-hall` example. In **Topology Explorer**, use the entity tree, search, graph layout selector, and **Fit topology in view** to make the hierarchy readable. Select a resource to inspect its details. Compare the explorer with **Members** and **Model** before you make an activation decision. Resource-group, current-power, and node-power-cap overlays report operational information. Treat them as reported state, not as confirmation that a configuration was applied.



Topology Explorer for the `east-hall` topology.

2.5.3.5.6.11 Activate a Topology

Import creates or updates an inactive topology. Activation is a later catalog action that makes that topology active and can deactivate the currently active topology. In the catalog, open the inactive row actions and select **Activate**. The confirmation identifies the candidate and states whether it deactivates a current topology.

For the first activation, decide whether DPS should **Ping hosts before activation**. Set a valid **Minimum successful (%)** threshold. If the dialog lists **Active resource groups**, review them with the owning operator because DPS can reevaluate them after the switchover. Select **Activate** only when the confirmation, topology impact, resource-group impact, and change-control approval match the intended work.

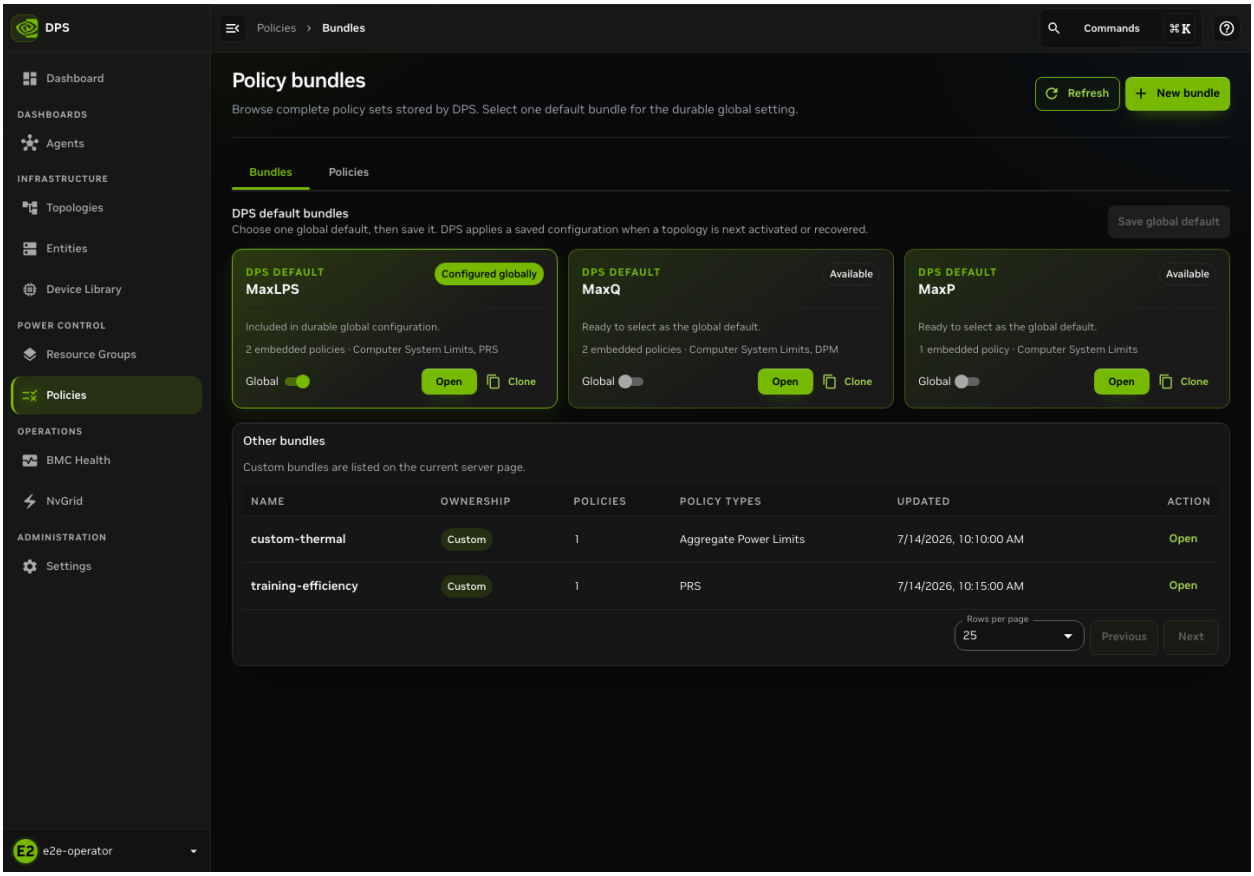
After the server responds, return to the catalog and Explorer. Record the reported active state and any per-node activation result. A graph alone does not prove that policies or reachability checks completed.

2.5.3.5.6.12 Manage Policy Bundles and Set the Global Default

Policy bundles are complete policy sets stored by DPS. The **Policies > Bundles** page separates two areas:

1. **DPS default bundles** are the pinned defaults eligible for the durable global default: **MaxP**, **MaxQ**, and **MaxLPS**.
2. **Other bundles** are custom bundles from the current server page.

DPS supplies the embedded policy definitions, scopes, and write result. Do not infer a bundle effect from its name.

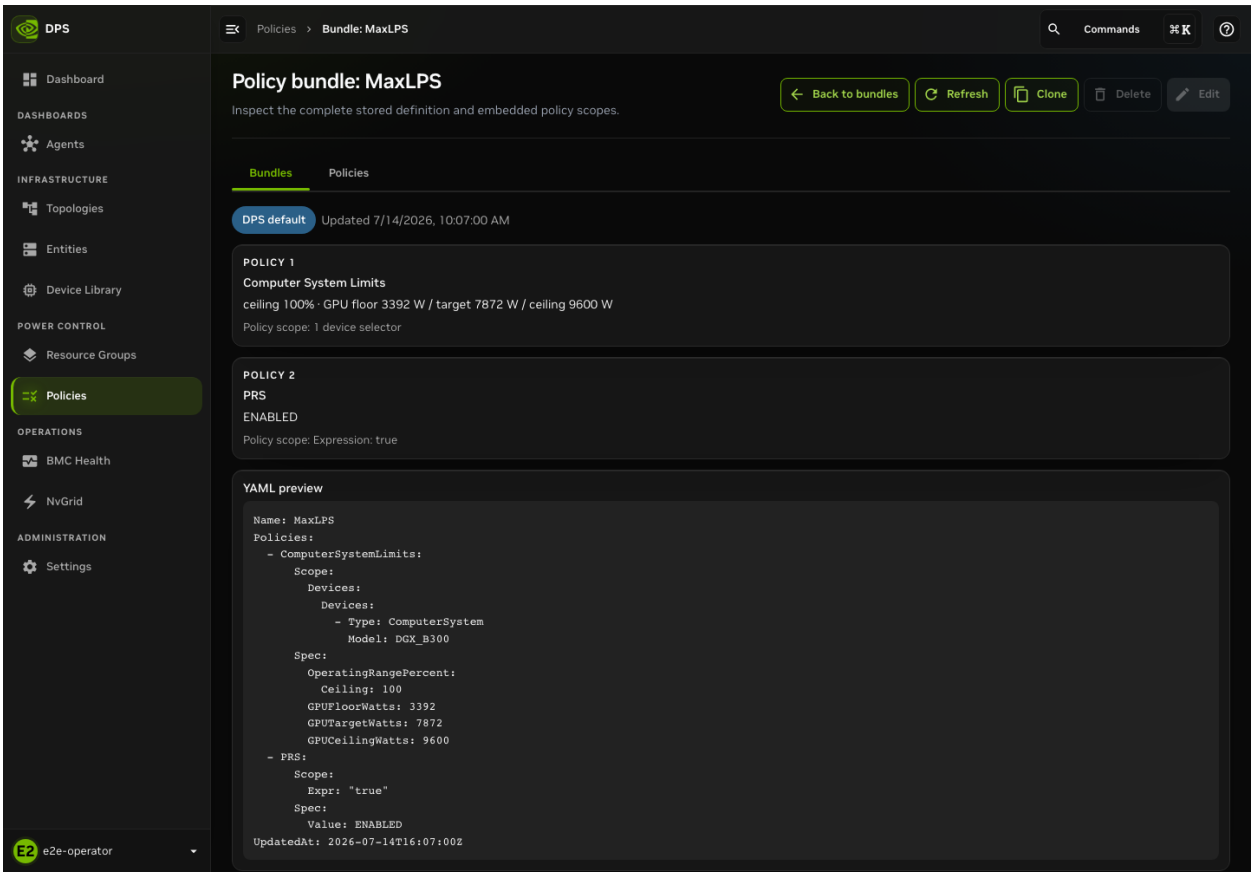


This example uses MaxLPS as the current global setting. MaxQ and MaxP are available alternatives. Custom bundles appear separately.

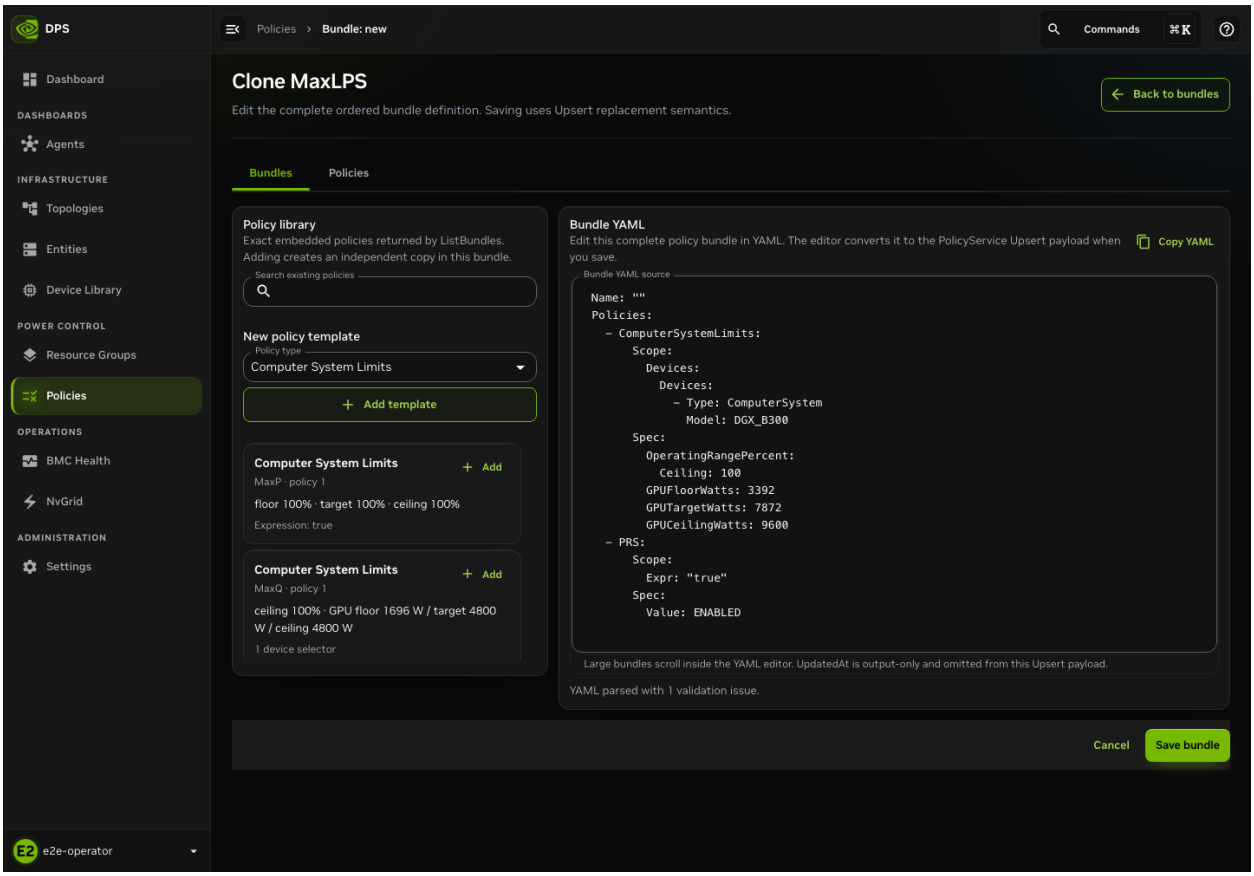
2.5.3.5.6.13 Inspect, Clone, Create, and Edit Bundles

Use this procedure to work with policy bundles.

1. Open a default or custom bundle and inspect the embedded policies and scopes.
2. Select **Clone** to use a DPS default as a starting point, and save the clone under a new custom name.
3. Select **New bundle** to compose a custom bundle. Select **Edit** to modify a custom bundle. Both actions require `policy:write`.
4. Review the complete bundle before you save it. Saving is a server write. It does not select the bundle or apply it anywhere.



Bundle detail displays the stored definition from DPS.



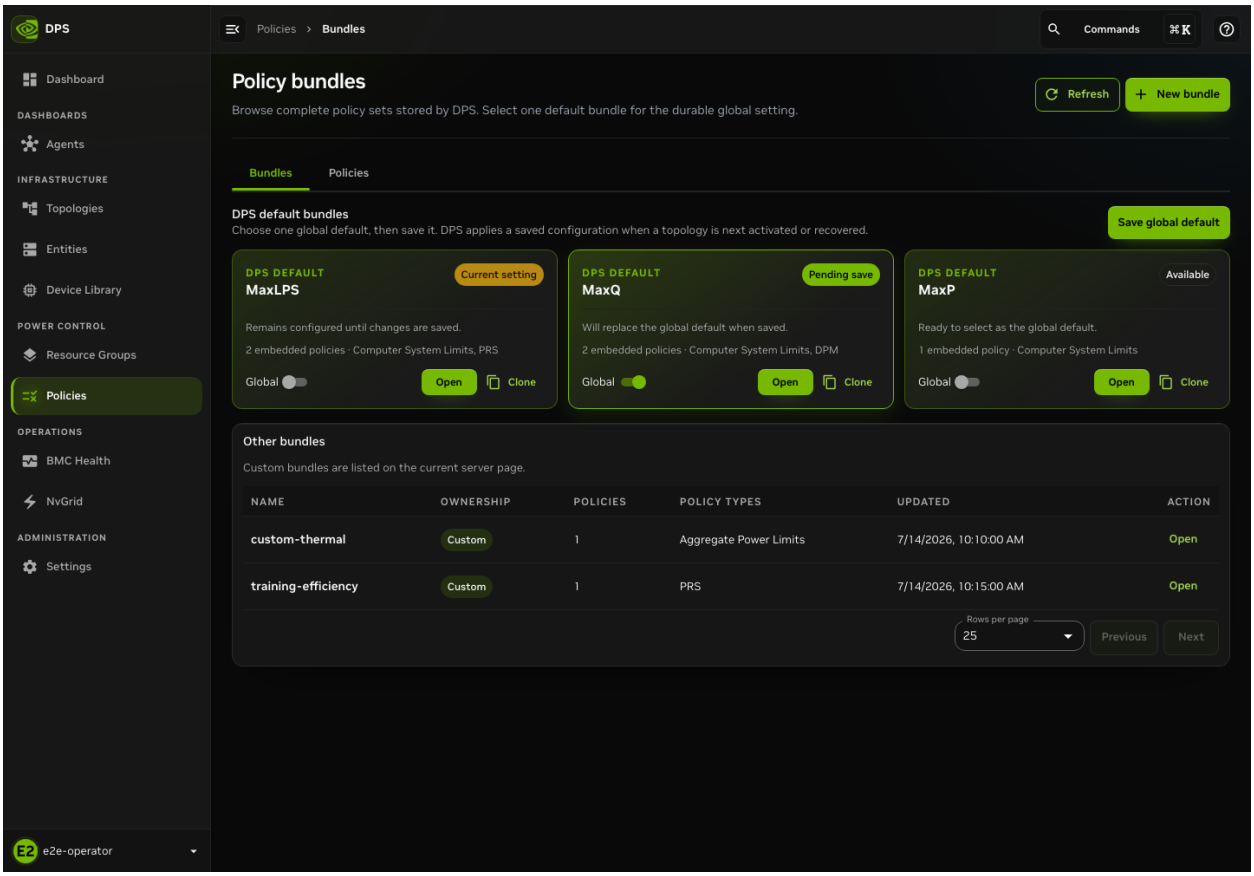
Cloning creates an editable custom-bundle draft. Supply a unique name before you save it.

2.5.3.5.6.14 Set the Global Default Policy Bundle

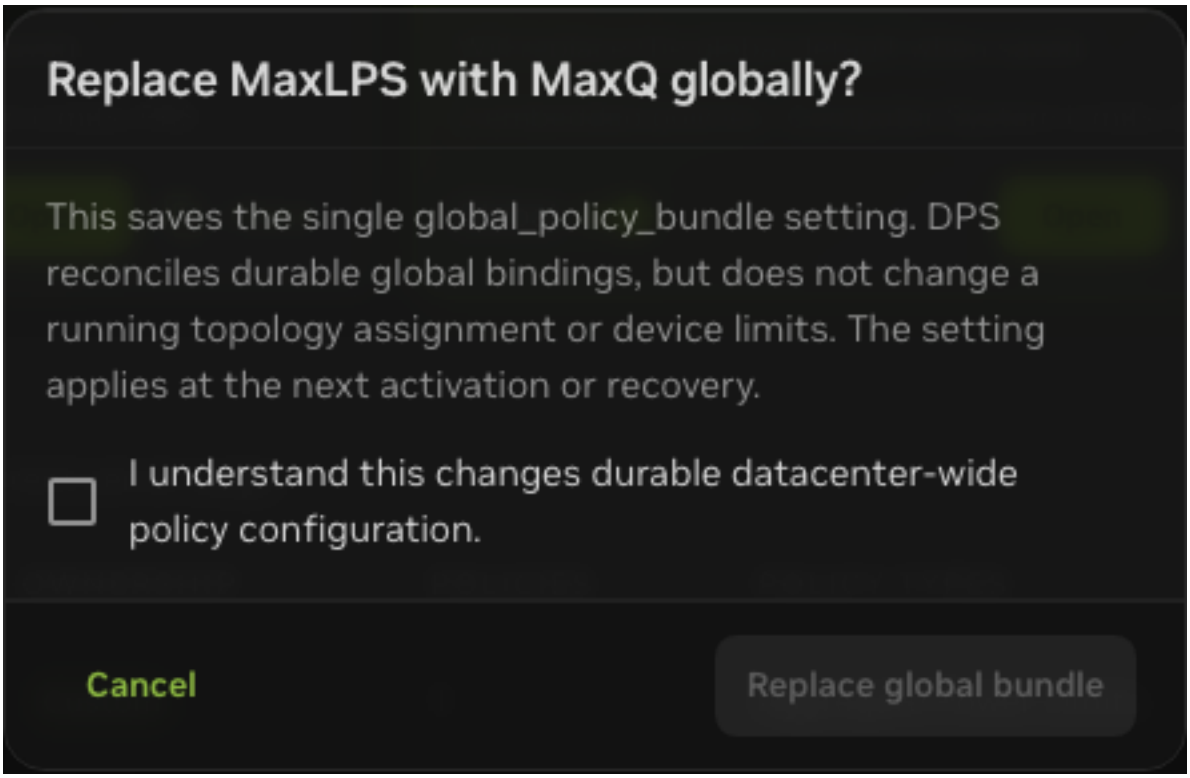
The UI stores the selected default in the durable `global_policy_bundle` configuration setting. This setting is not a command to immediately apply a per-device or per-topology policy.

Out of the box, **MaxLPS** is the configured global default. It remains selected unless an operator saves **MaxP** or **MaxQ** as its replacement, or intentionally removes the global default.

1. Open **Policies > Bundles** and locate **DPS default bundles**.
2. Review each card state. **Configured globally** is the setting read from DPS. **Current setting** remains effective while another choice is staged. **Pending save** identifies the staged replacement.
3. Turn on **Global** for exactly one of **MaxP**, **MaxQ**, or **MaxLPS**. Selecting another default stages a replacement. It does not save the change.
4. Select **Save global default** and review the confirmation. DPS saves one `global_policy_bundle` setting. It does not create a global binding, rewrite a topology or resource-group selection, or change device limits.
5. Acknowledge the change only when future empty bundle selections should resolve to this value. On success, the UI reads the setting back and displays the current state.

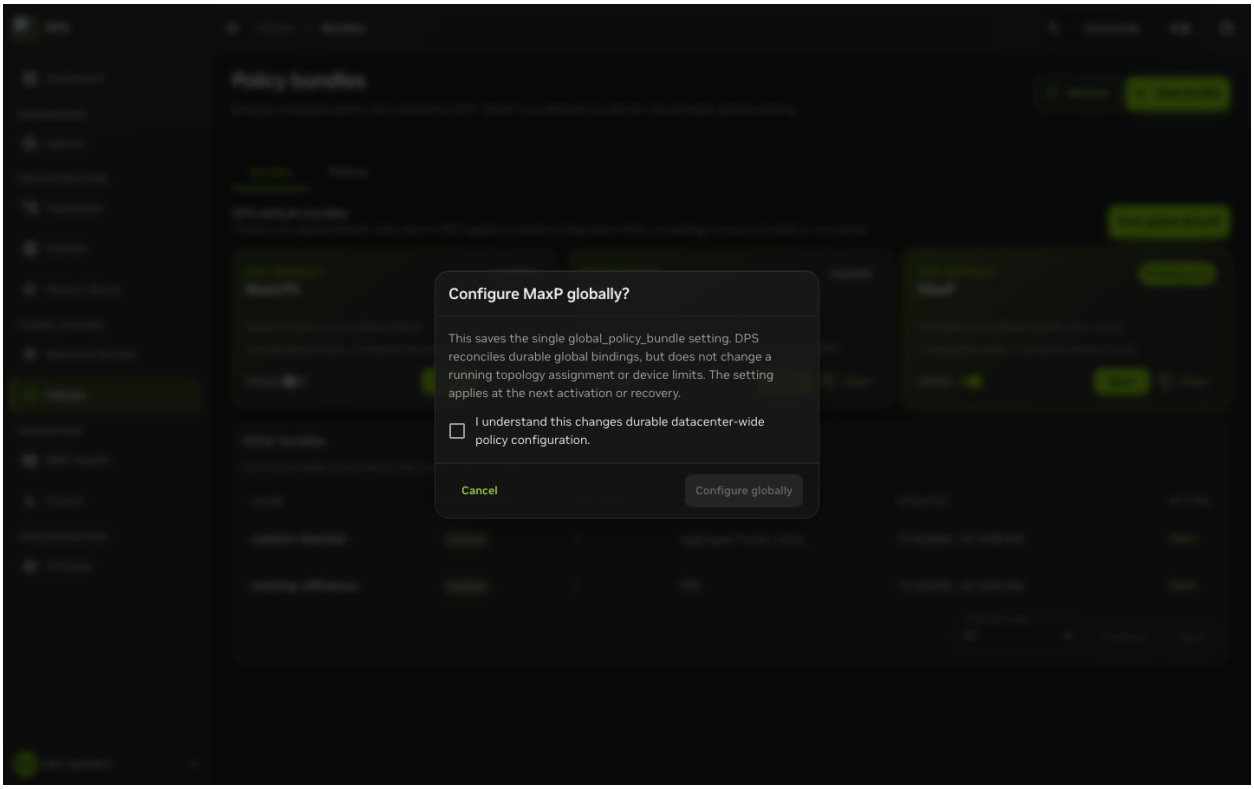


MaxLPS remains the current setting until you save. MaxQ is the staged replacement.



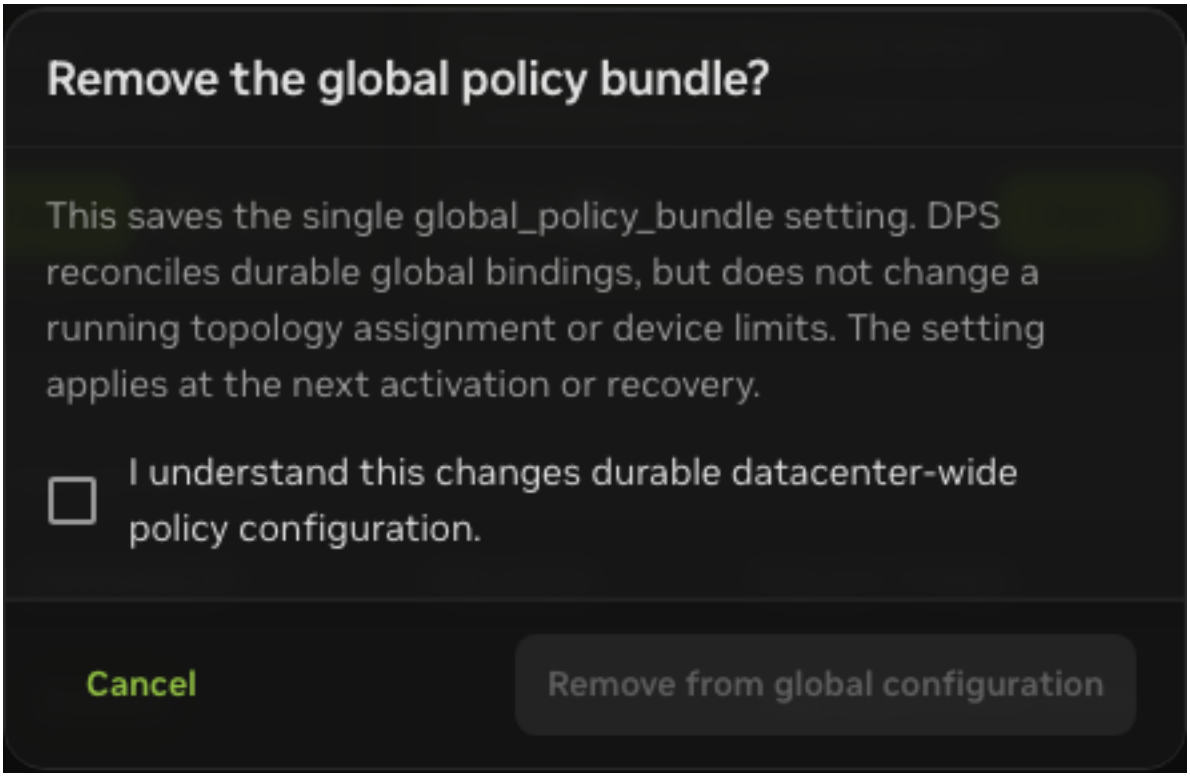
Replacing the default requires acknowledgment of a durable configuration change used for future empty bundle selections.

When no global default exists, selecting a default creates an initial configuration confirmation.



Initial MaxP selection. The same setting and lifecycle boundary applies to every default bundle.

To remove a configured global default, turn off the current bundle **Global** switch and then select **Save global default**. DPS receives an empty `global_policy_bundle` setting only after you acknowledge the removal confirmation.



Removing the global default changes durable configuration. Active topology and device state are unaffected.

If a save fails or the readback does not preserve the selected value, the UI reports an error. Do not assume the bundle is configured. Refresh and inspect the returned state. If the stored `global_policy_bundle` is malformed, the default-card controls remain disabled until DPS returns a valid setting. Use the approved recovery path to correct the configuration.

A configured global default shows which configuration is selected. It does not mean that DPS reactivated an existing topology or applied the policy to a device.

2.5.3.5.6.15 Create and Activate a Resource Group

A resource group combines member resources, a policy-bundle selection, priority, and optional metadata. Resource-group changes can affect allocation outcomes. Use a staged workflow.

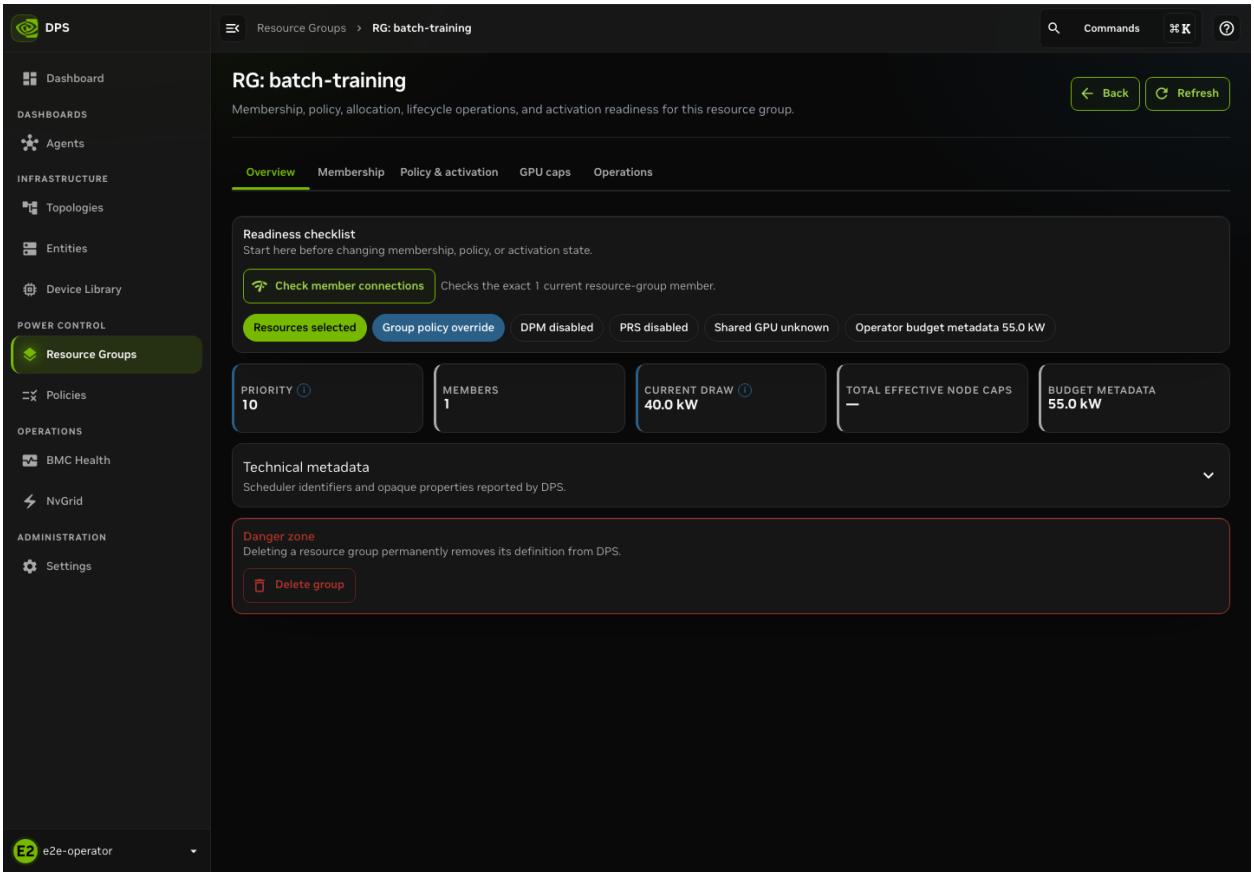
2.5.3.5.6.16 Create the Group and Select Members

Use these steps to create a resource group.

1. Open **Resource Groups** and select **New resource group**.
2. Enter a unique group name. Select a policy bundle explicitly or leave it unset to resolve and persist the current `global_policy_bundle`. An unset value does not categorically mean “inherit the topology policy.”
3. Add members from the active topology or enter exact resource names. Review overlap messages. A resource cannot belong to more than one active resource group at a time.
4. Review priority and advanced metadata before you select **Create group**.

After creation, open the group **Overview**. Start with the **Readiness checklist**, which reports membership, policy, capability, and operator-budget context before lifecycle action.

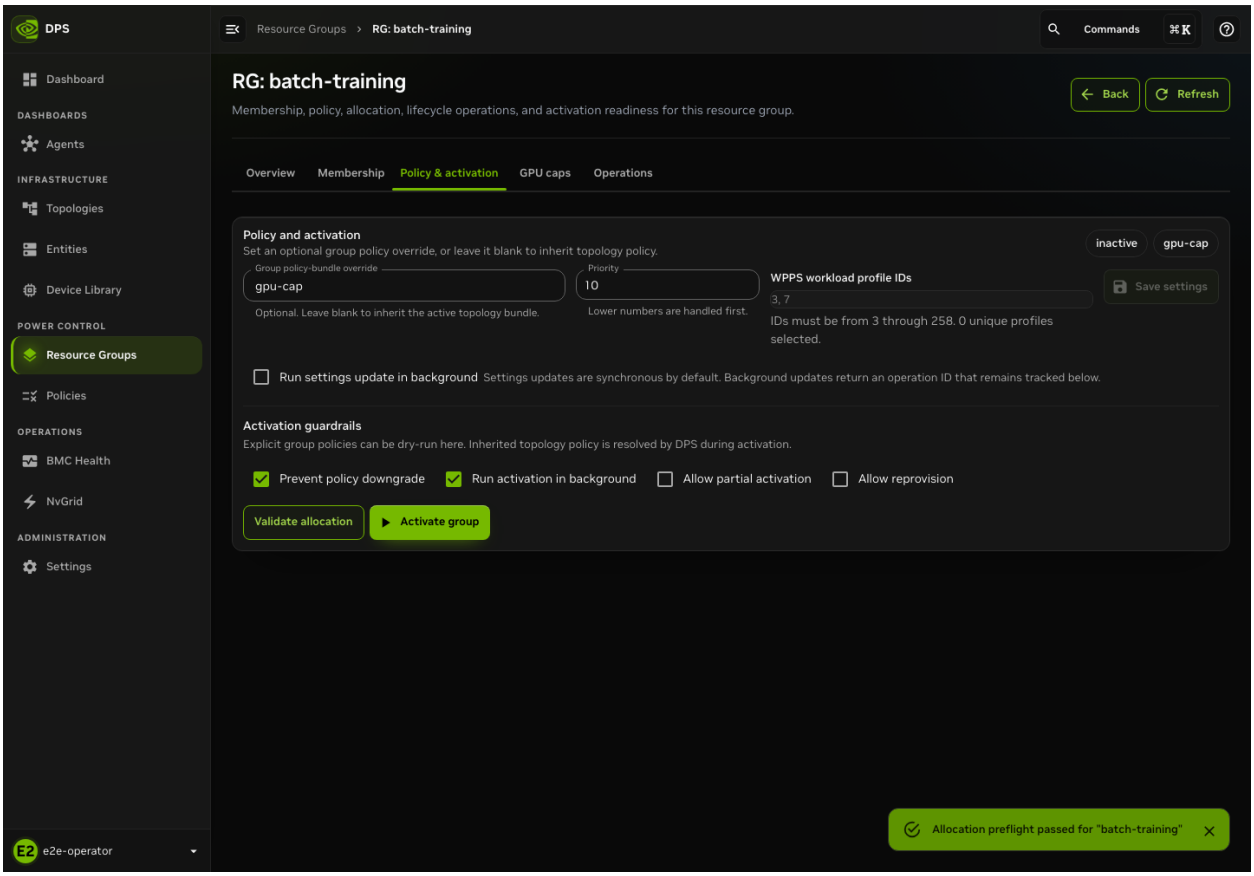
The current server does not support changing a resource group's policy-bundle selection through an update. Select the required bundle when creating the group.



Readiness for the inactive `batch-training` group. Review this state before you edit membership or activation options.

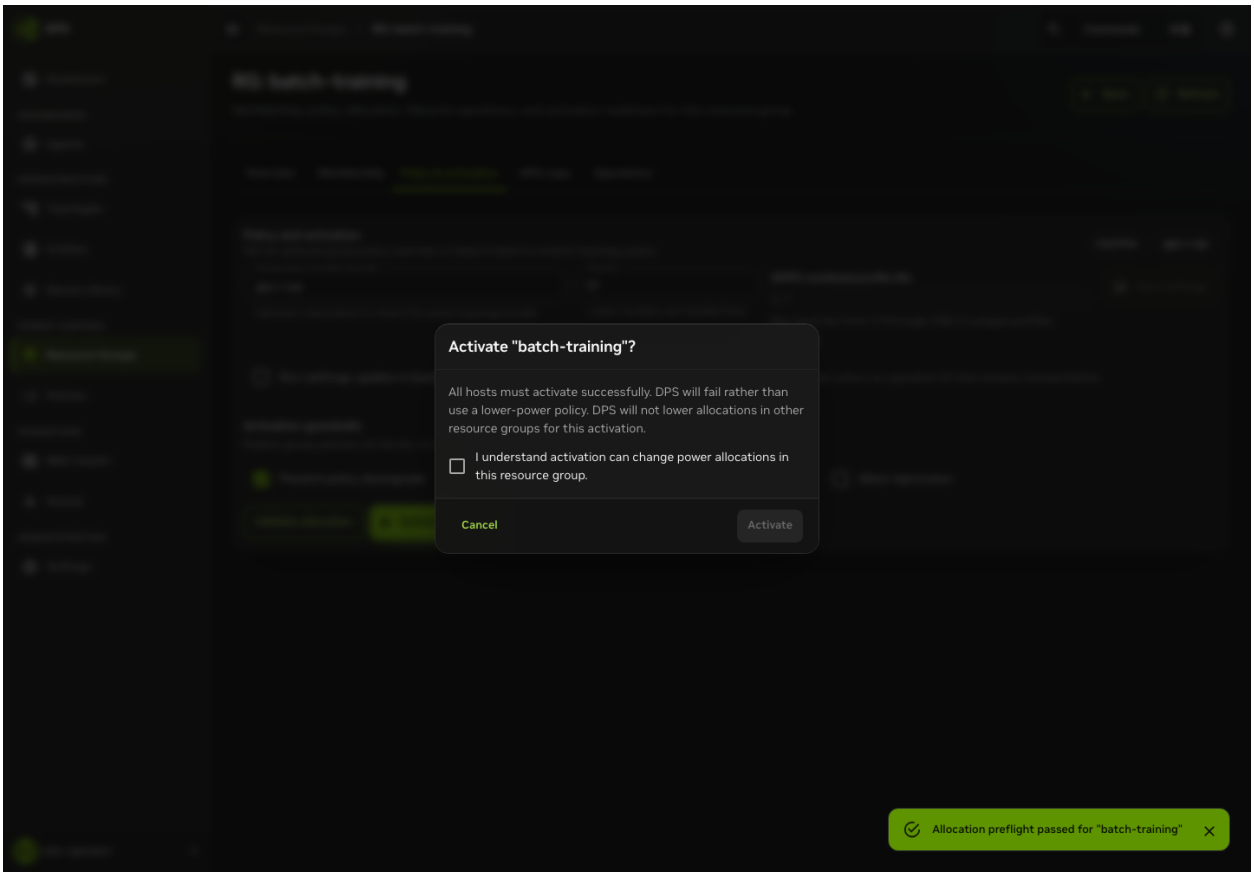
2.5.3.5.6.17 Validate Allocation Before Activation

Open **Policy & activation** and select **Validate allocation**. This is the preflight for current membership, selected policy, and activation guards. Validate again after you change any supported input.



A successful preflight enables the next guarded action. Activation is separate.

Select **Activate group** only after you understand the preflight and allocation impact. The confirmation describes how activation can change resource-group power allocations. Acknowledge it only with the required operational approval.



Activation requires explicit acknowledgment. Retain approved partial-activation and reprovisioning safeguards unless the operational plan requires a change.

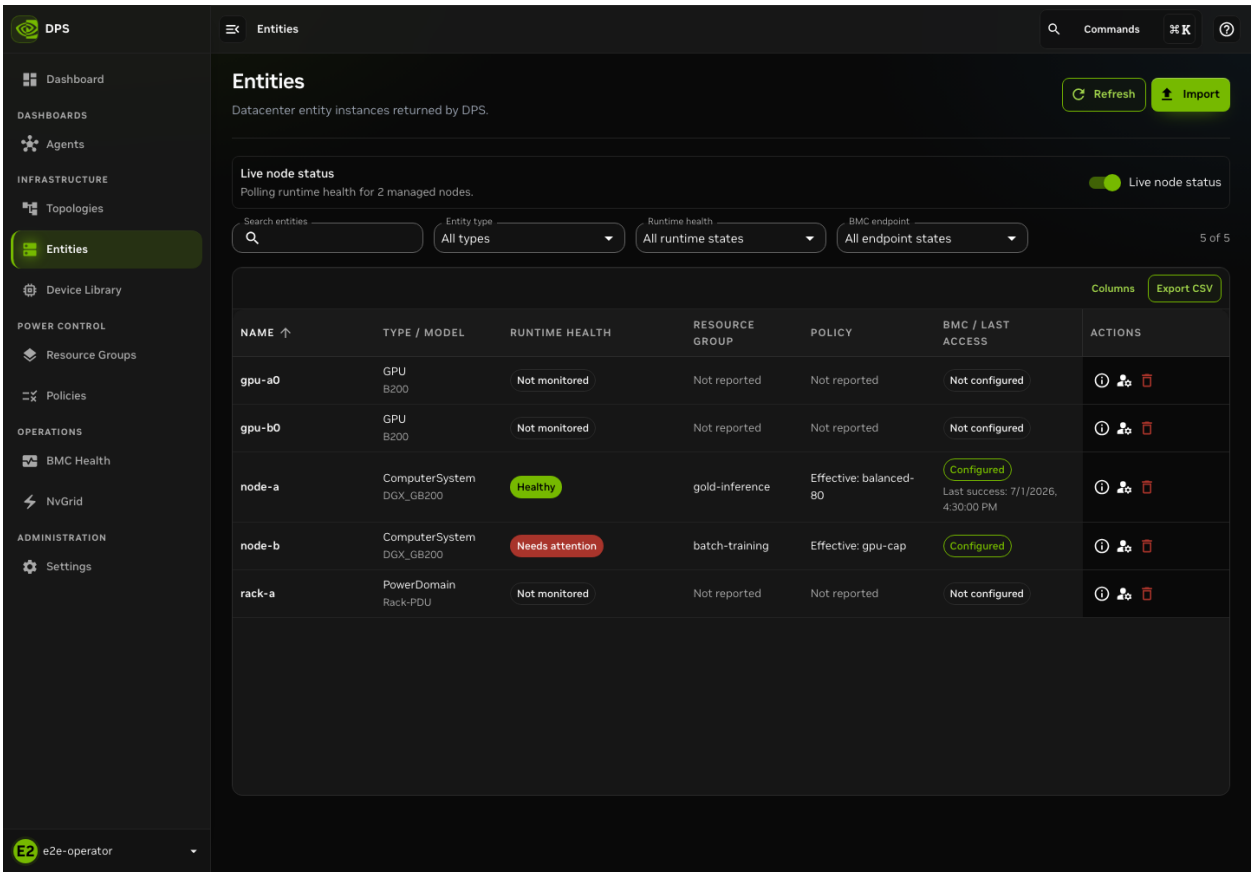
Track the result in the resource-group **Operations** section. A task ID or a request acknowledgment does not prove that every resource accepted its policy. Review the returned per-resource status.

2.5.3.5.6.18 Inspect Operational Views

Use operational views to investigate and verify current state. They do not replace the result of a write, topology lifecycle operation, or resource-group operation.

2.5.3.5.6.19 Entities and Device Templates

Open **Entities** to inspect active-topology inventory. **Live node status** requests the latest status data. Use runtime-health filters to focus the investigation. Inspect a row for entity details, effective policy context, and related resource-group information.



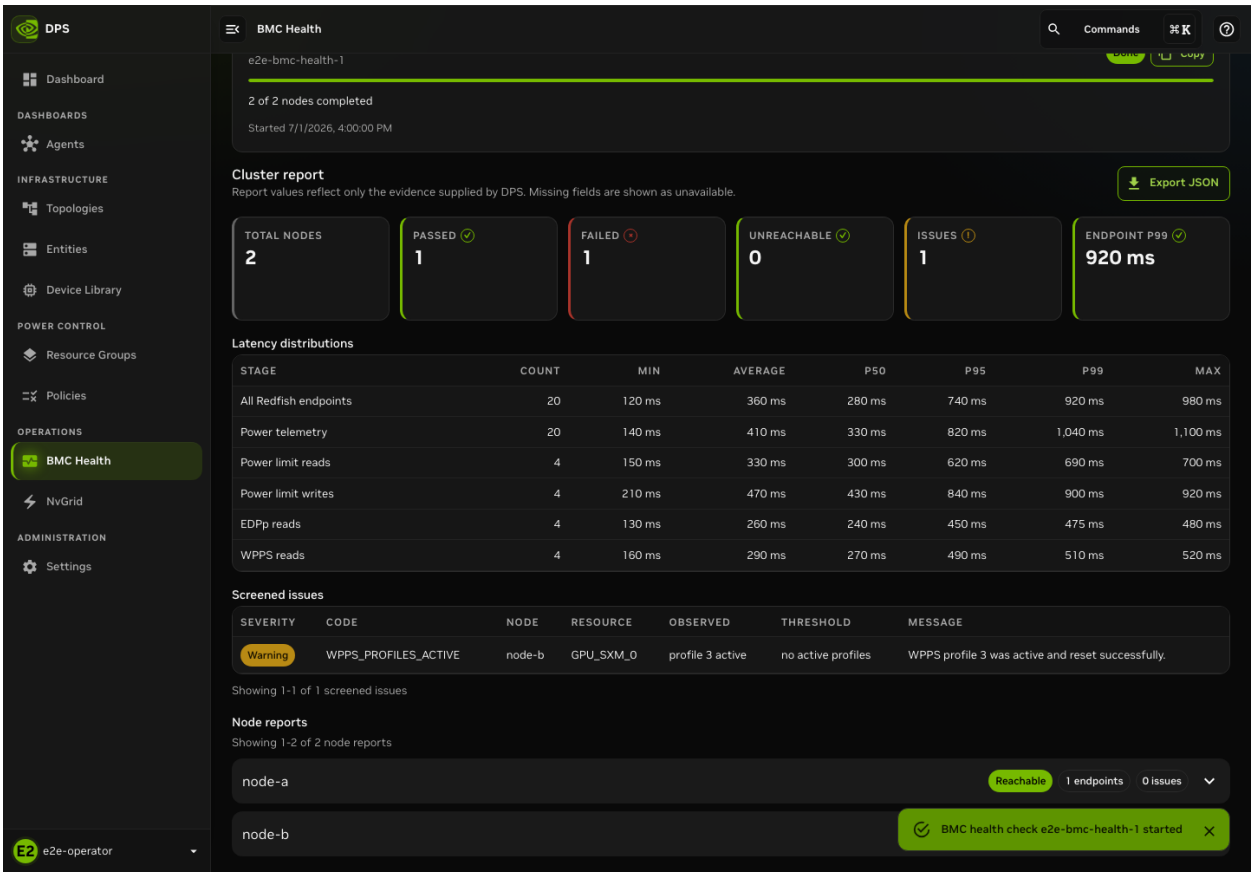
Entity inventory with live-status information enabled.

Use **Device Library** to inspect registered device-template definitions, including supported component and management information. Device templates are inventory definitions. Changing one is a separate write workflow that requires review before import or replacement.

2.5.3.5.6.20 BMC Health Reports

DPS 0.9 requires the selected topology to be active before starting a BMC health check, including in **Read-only** mode. Follow *Managing Topologies* to activate the topology before running this diagnostic.

Open **BMC Health** for report-oriented diagnostics. Prefer **Read-only** mode for endpoint and prerequisite evidence. Start a check only after you confirm the topology and node scope. Retain the returned task ID. Review the cluster summary, latency distributions, issues, and node-specific detail.



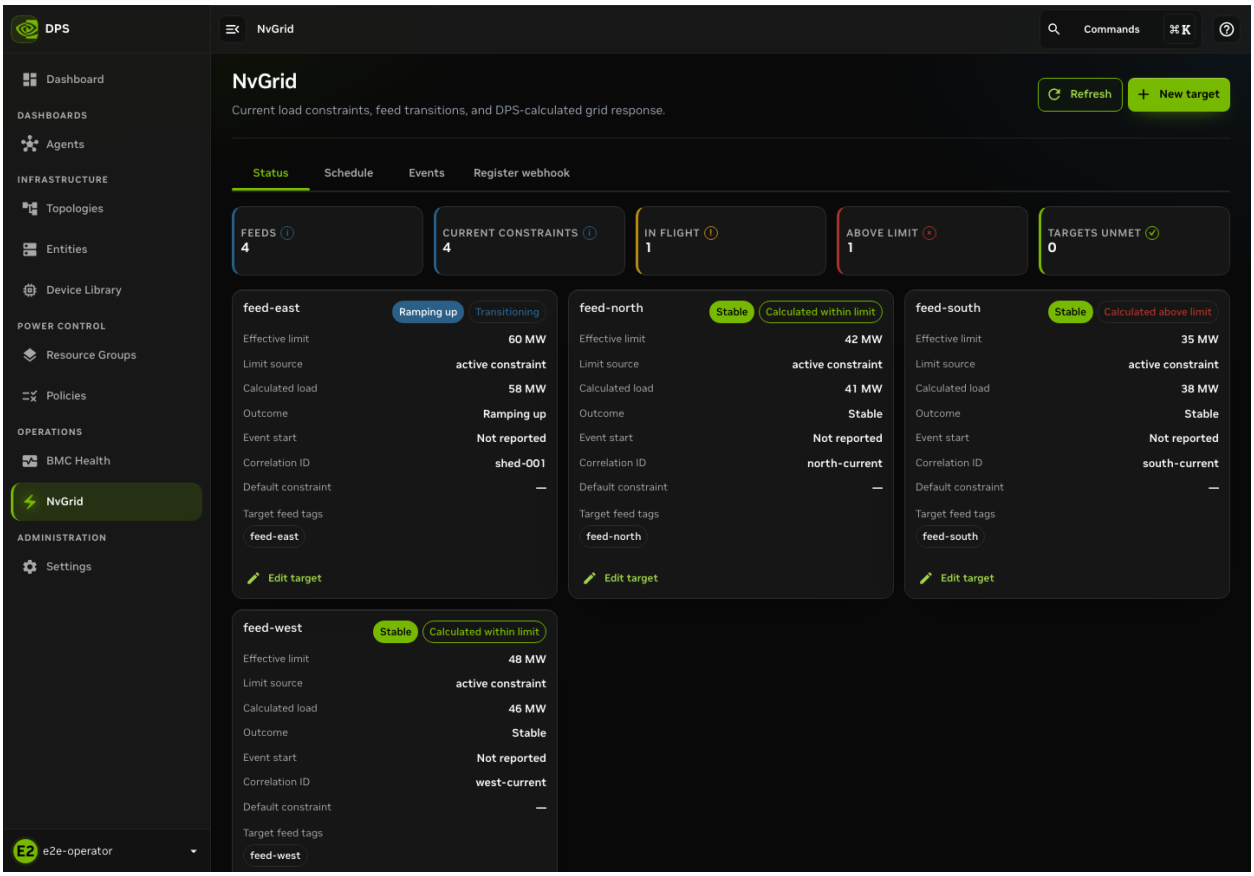
Read-only report. Missing information is not evidence of health.

Deep BMC probes can exercise temporary power-limit writes. They have additional side effects and are outside this read-only procedure. Use a deep probe only with the documented approval and recovery plan.

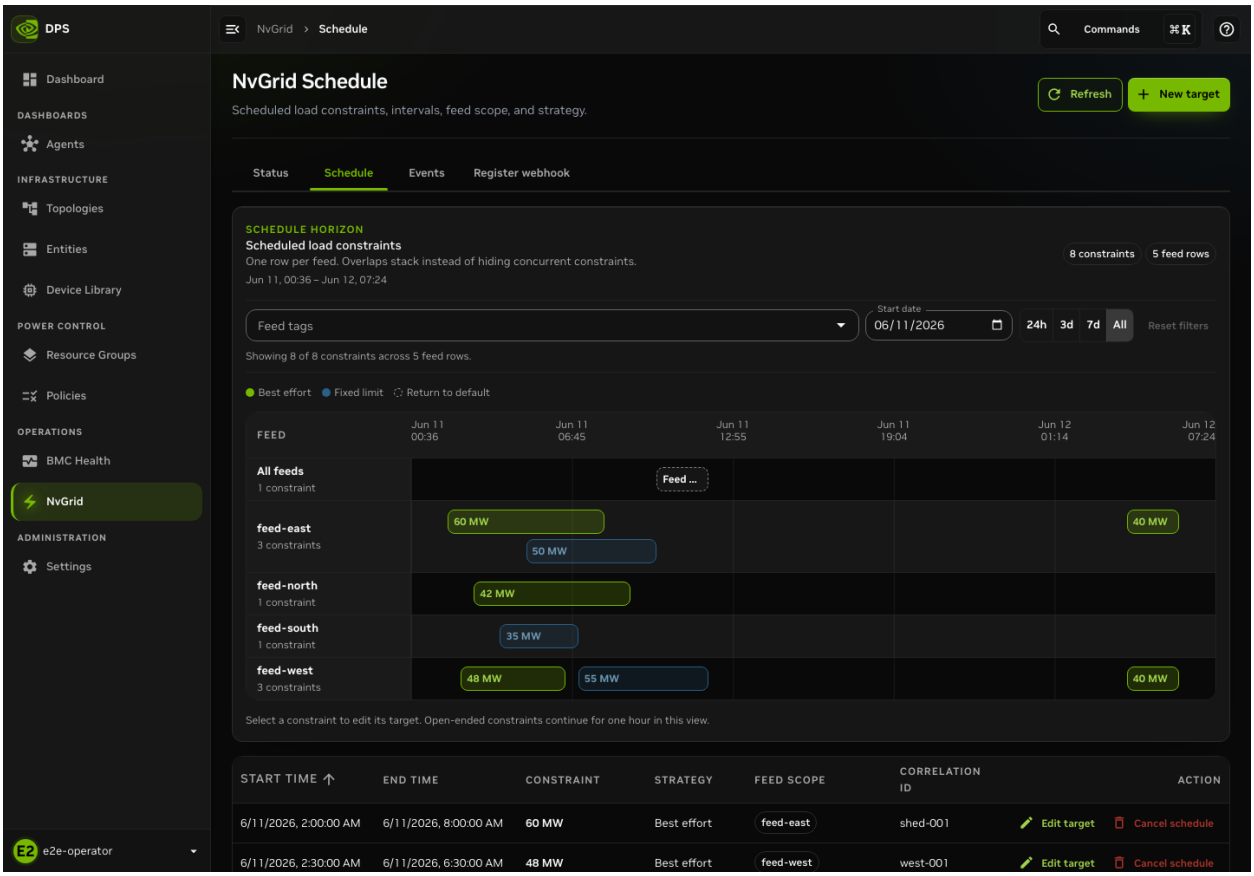
2.5.3.5.6.21 Agents, Allocation, NvGrid, and Settings

Use these views for read-only inspection.

- **Agents** shows current agent information and assignments.
- **Allocation** shows capacity, draw, and resource-group allocation context. This context is not a prediction that a pending activation succeeds.
- **NvGrid** shows feeds, current targets, and scheduled targets. Inspect the affected feeds and timing before you consider a target or webhook mutation.
- **Settings** shows the connected server and deployment settings. Any save is system-wide and requires review.



NvGrid feed status for the example deployment.



NvGrid schedule. Review target timing and affected feeds before you plan an operational change.

2.5.3.5.6.22 Identify Deployment and BMC Issues

Use this procedure to collect evidence and route an issue to the owning team. It does not replace site change control, BMC break-fix procedures, or the result that DPS returns for a lifecycle operation.

2.5.3.5.6.23 Triage Deployment and Access Issues

Start with **Settings > Deployment**. The **DPS server** card shows the server-reported version, commit, build date, and last refresh.

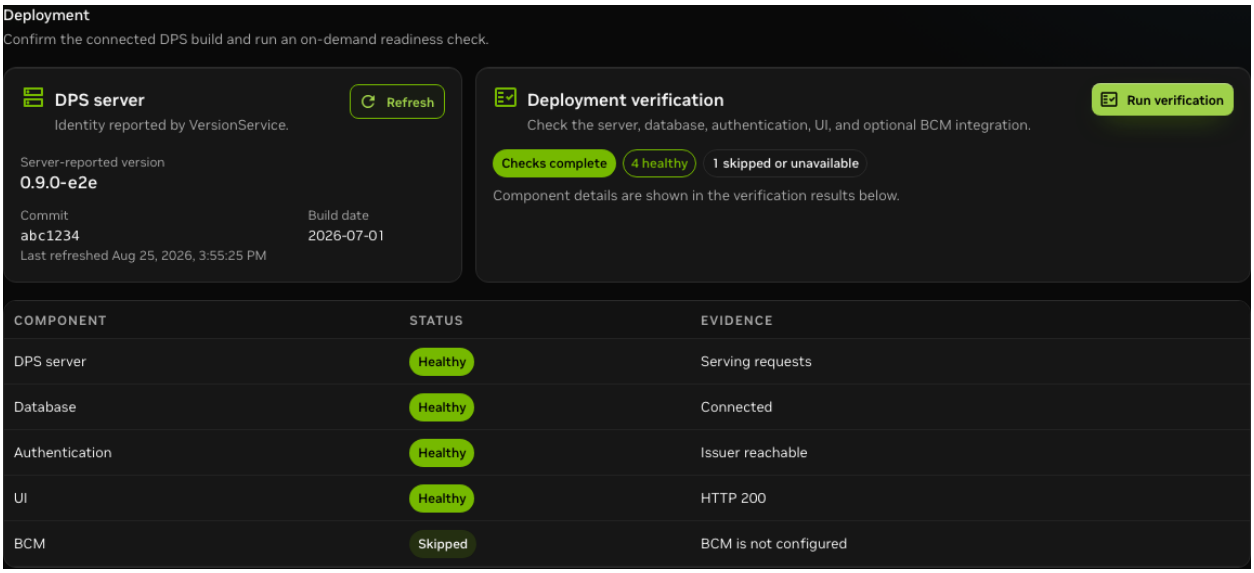
Select **Refresh** when an expected version or build is absent. The UI retains the last successful response. It identifies a failed refresh rather than treating stale information as current.

Select **Run verification** to request the deployment readiness check. Record each reported component separately.

Reported Component	Record	Initial Routing
DPS server	Status, message, server version, commit, and build date.	DPS service owner.
Database	Status and returned diagnostic.	DPS data-service owner.
Authentication	Status and sign-in error, without tokens, callback values, or cookies.	Identity and authentication owner.
UI	Status and affected page or route.	UI deployment owner.
BCM	Status, **Skipped**, or **Not reported** result.	BCM or infrastructure owner when BMC management is expected.

Action required means that DPS reported at least one unhealthy component. **Incomplete**, **Skipped**, and **Not reported** are not healthy results. They show that the check did not return a complete component result.

Do not use an optional BCM result alone to decide that a node BMC is healthy.



The example shows version identity, four healthy components, and an explicitly skipped optional BCM integration. Treat each result as triage input, not as a generic deployment-health claim.

For a sign-in failure, retain the visible error and affected UI URL without its query string. Verify the identity provider and expected role before retrying.

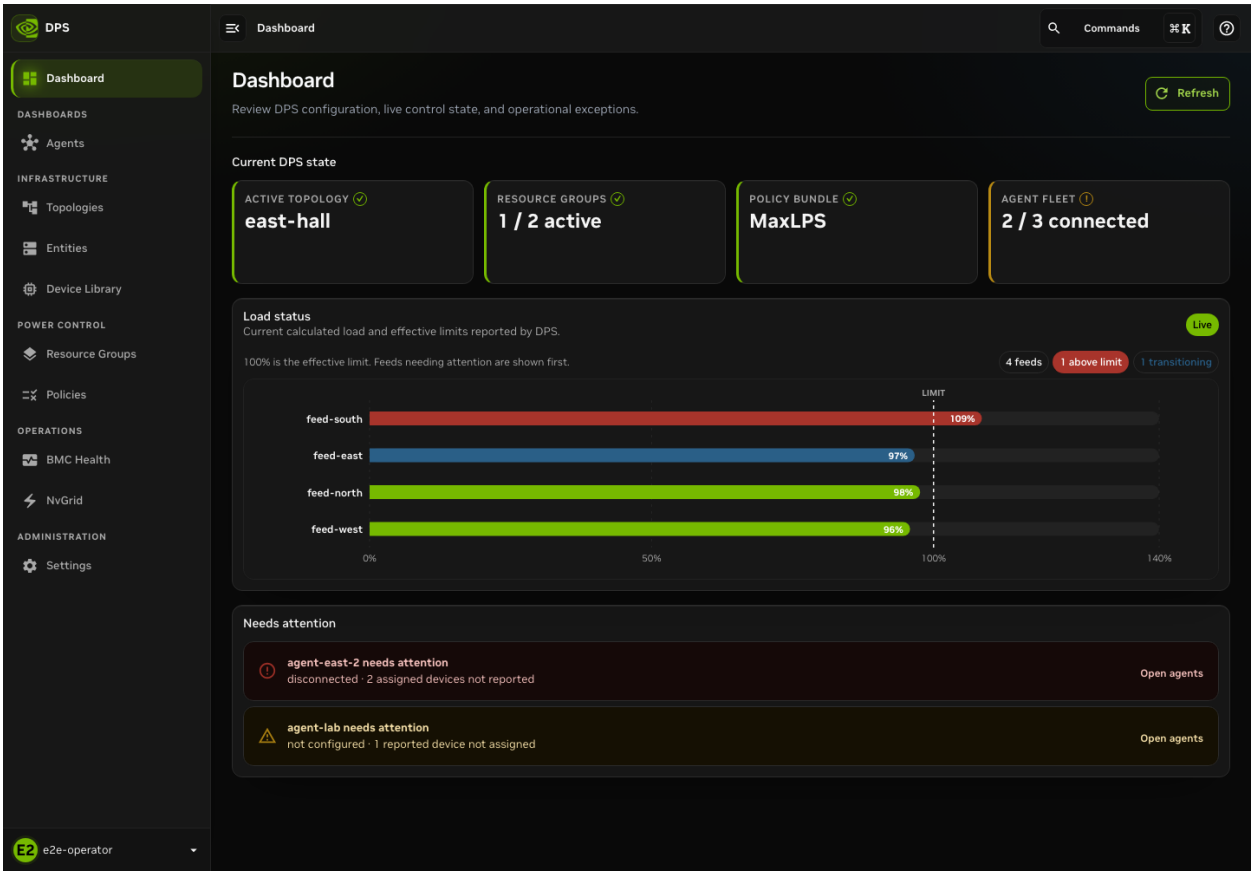
Do not include authorization codes, `state` values, refresh data, bearer tokens, cookies, or passwords in a ticket, chat, image, or exported log.

For an authenticated page that cannot load DPS data, record the route, visible error, time and time zone, and deployment-verification state before you retry. A page-level error can result from access control, service availability, or an unavailable upstream dependency.

Do not diagnose the cause from a browser status alone. Avoid **Save**, **Import**, activation, and other write controls while you collect failure evidence.

2.5.3.5.6.24 Start BMC Triage from Dashboard Attention

The dashboard **Needs attention** area and the **BMC Health** navigation item provide an operational starting point. A baseboard management controller (BMC) result comes from a node report, not from the dashboard. Select **BMC Health** to start a diagnostic workflow. If your deployment presents **Open BMC health** for a BMC- or telemetry-related item, use that action. Start the check in **Read-only** mode and retain its task ID. The resulting report records which selected nodes were reachable and which checks failed.



*Dashboard triage entry point. The red `agent-east-2` item reports a disconnected agent and missing device reports. It does not show that a BMC is unreachable. Use **Operations > BMC Health** to collect a read-only report.*

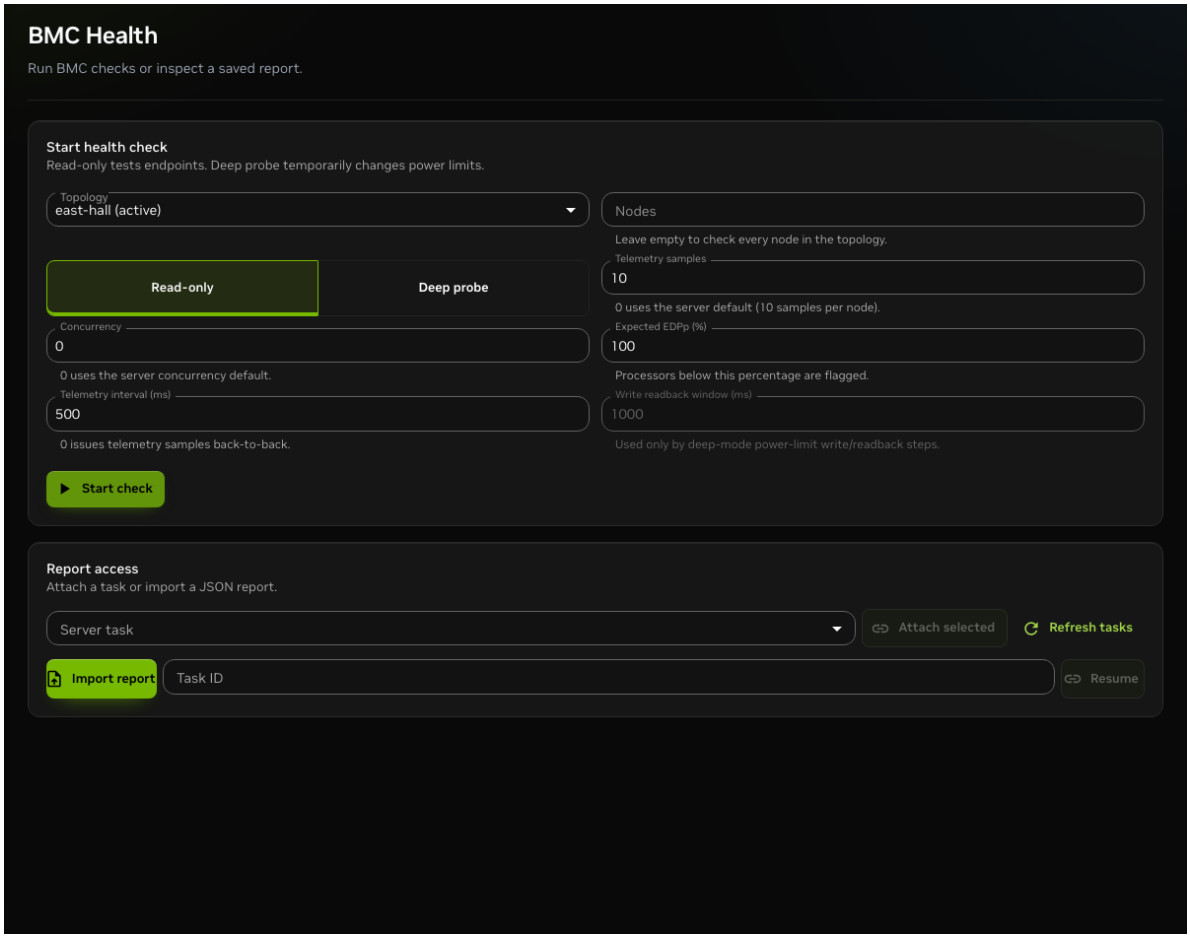
An **Agent fleet** warning is important context, but it does not prove that a node BMC is unavailable. Review the agent assignment and telemetry path, then use the BMC Health report to distinguish agent or data-path coverage from BMC endpoint evidence. A missing telemetry reading is also not a BMC failure by itself. Do not activate a topology, change a policy, or run a deep probe only because the dashboard contains an attention item.

2.5.3.5.6.25 Diagnose a BMC Issue Safely

Use this read-only procedure first.

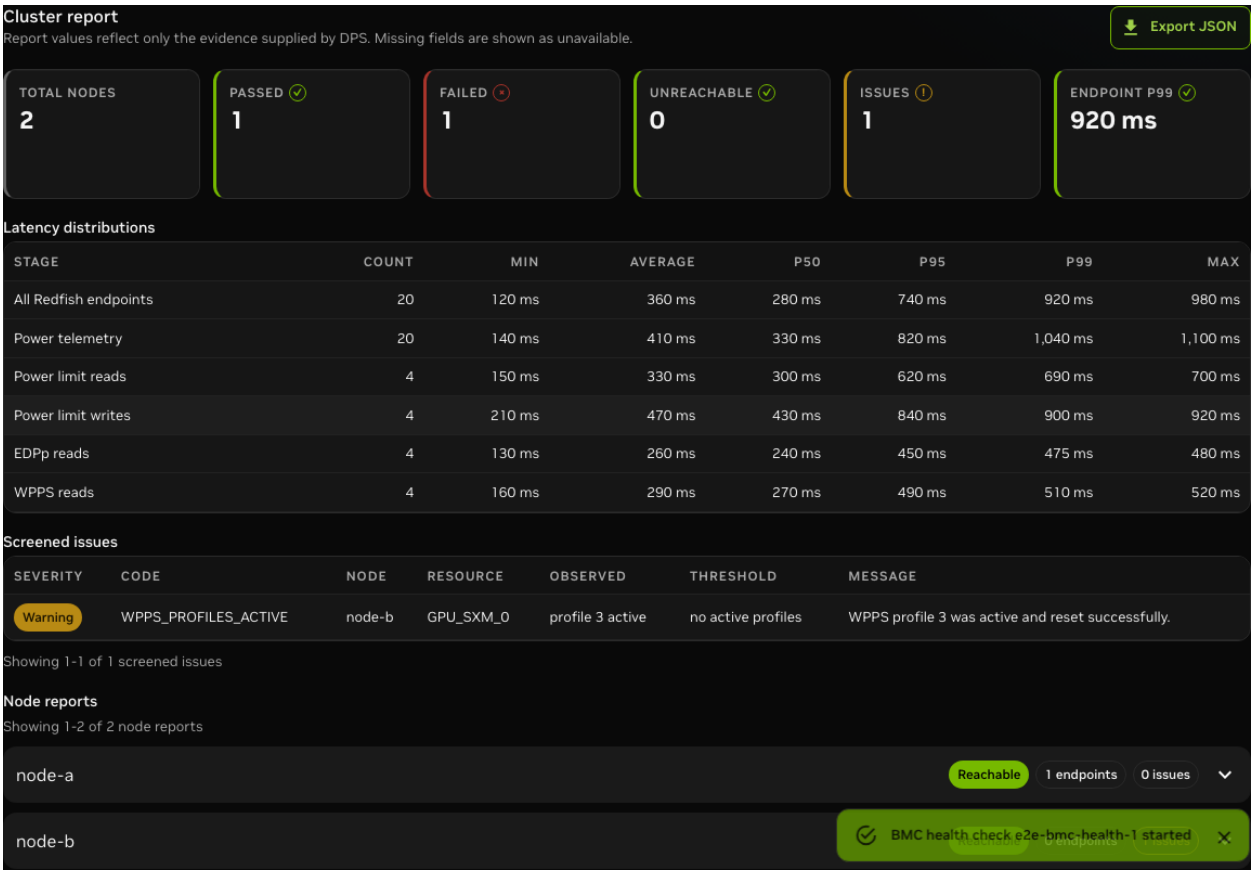
1. Open **BMC Health**. Select the topology and node scope. Retain **Read-only**, which tests endpoints without the deep-probe temporary power-limit changes.
2. Start the check and retain its task ID. If another operator started it, use **Report access > Server task** or enter the known **Task ID**. This attaches to the existing report instead of starting a dupli-

cate probe.



*Initial diagnostic screen. Select scope, retain **Read-only**, and either start a check or attach to an existing task. Do not use a deep probe as the default troubleshooting action.*

3. In **Cluster report**, separate task or report availability from node findings. An unavailable task or report identifies a task-retention issue. Failures and unreachable counts describe evidence from selected nodes.
4. Review **Latency distributions** by stage. Compare count, average, p95, p99, and maximum with a known-good run in the same environment. Do not use a universal latency threshold. Endpoint or telemetry latency requires correlation with BMC, network, and agent telemetry. It does not prove a root cause.
5. Use **Screened issues** to identify the node, resource, observed value, threshold, and DPS message. Open the matching **Node report** for firmware, endpoint data, power-setting evidence, Electrical Design Point (EDPp) settings, Workload Power Profiles Settings (WPPS), and telemetry detail.
6. Export report JSON and retain the task ID before escalation. Include only relevant node names and diagnostics. Remove credentials, IP-address details, and other sensitive site data according to the incident process.



Report with a failed node and a screened WPPS issue. Use the node report to separate an observed symptom from a diagnosis.

Warning

Do not use **Deep probe** as a harmless retry. It can temporarily change power limits. Setpoint restoration is best effort. The probe can disable active WPPS profiles and does not restore them automatically.

Run a deep probe only with a site-approved maintenance window, scope, rollback plan, and authorization.

2.5.3.5.6.26 Operator Checklist

Before you start a write or lifecycle action, confirm these items.

1. The selected topology, resource group, policy bundle, or task is the intended object.
2. The UI shows the expected DPS readback and no unresolved error.
3. You ran validation after the final edit.
4. The confirmation text matches the intended operational scope.
5. A follow-up view or operation result confirms what DPS accepted.

2.5.3.5.7 Managing Resource Groups

2.5.3.5.7.1 Overview

This guide provides step-by-step instructions for managing resource groups in DPS.

For more information, refer to *Resource Groups*.

Note: Resource groups are typically managed by a workload scheduler. This guide demonstrates how to manage them directly.

2.5.3.5.7.2 Prerequisites

- DPS server running and accessible
- `dpsctl` installed and authenticated through *Configure Server Authentication*
- Active topology with entities configured

2.5.3.5.7.3 Basic Resource Group Workflow

2.5.3.5.7.4 Step 1: Create Resource Group

Shared GPU mode defaults to on. The following command creates a shared-GPU resource group. Pass `--shared-gpu-enable=false` if you need per-node GPU caps instead of a group-wide GPU budget.

```
dpsctl resource-group create \
  --resource-group "ml-training-job" \
  --external-id 12345 \
  --policy-bundle MaxP
```

`--priority` is optional and defaults to `0`, the highest precedence. A larger number is lower importance during stealing. Refer to *Resource Group Priority*.

2.5.3.5.7.5 Step 2: Add Hardware Resources

```
dpsctl resource-group add \
  --resource-group "ml-training-job" \
  --entities "node001,node002,node003"
```

2.5.3.5.7.6 Step 3: Activate Power Policies

```
dpsctl resource-group activate \
  --resource-group "ml-training-job"
```

2.5.3.5.7.7 Step 4: (Optional) Configure Per-GPU Power Limits

After activation, use the standalone GPU policy command for fine-grained GPU control on a node in the active resource group:

```
dpsctl gpu-policy \
  --node "node001=500,550,600,700,650,700,550,600"
```

2.5.3.5.7.8 Step 5: Cleanup

```
dpsctl resource-group delete \
  --resource-group "ml-training-job"
```

2.5.3.5.7.9 Dynamic Resource Management

Resources can be added to or removed from resource groups at any time, including after activation. This enables dynamic workload scaling without full resource group deactivation.

Note: Add/remove operations are rejected while a resource group is activating or deactivating. Wait for the operation to complete before modifying resources.

2.5.3.5.7.10 Adding Resources to an Active Resource Group

When adding resources to an active resource group, policies are applied immediately and power is reallocated as needed:

```
dpsctl resource-group add \
  --resource-group "ml-training-job" \
  --entities "node004,node005"
```

2.5.3.5.7.11 Strict Policy Enforcement

Use `--strict-policy` to ensure the configured policy bundle is applied exactly. If power constraints prevent the bundle from being satisfied, the operation fails rather than retrying at the effective policy floor:

```
dpsctl resource-group add \
  --resource-group "ml-training-job" \
  --entities "node004" \
  --strict-policy
```

2.5.3.5.7.12 Controlling Power Reprovisioning

By default, power may be redistributed from other resource groups if needed (power stealing). Use `--allow-reprovision=false` to prevent this:

```
dpsctl resource-group add \
  --resource-group "ml-training-job" \
  --entities "node004" \
  --allow-reprovision=false
```

2.5.3.5.7.13 Removing Resources from an Active Resource Group

When removing resources from an active resource group, remaining topology limits are restored. If no effective limit remains, the idle-policy or hardware-minimum release path applies as described in *Policy Lifecycle and Fallback*:

```
dpsctl resource-group remove \
  --resource-group "ml-training-job" \
  --entities "node004,node005"
```

2.5.3.5.7.14 Troubleshooting

2.5.3.5.7.15 Shared GPU Budget Rejections

GPU policy updates on a shared-GPU resource group can fail even when each node's requested watts look valid in isolation. DPS first fits the post-update group total to the resource-group GPU budget, then checks non-compute topology ancestors such as racks and PDN devices. Refer to *Shared GPU Budget* for the model.

Typical diagnostics include:

- `shared-GPU update violates topology constraint at device:<ancestor>`
- `response violates topology constraints`

The named ancestor is the topology entity whose load would exceed its constraint. Correct the request by lowering the requested GPU caps, spreading the increase across more members, or raising the applicable topology limit, then retry the update.

2.5.3.5.7.16 Verifying Node Status

After activating a resource group, always verify that each individual node's policy was applied successfully:

1. Check that each node in `node_statuses` has `"ok": true`
2. Review any error messages in `diag_msg` fields
3. If any nodes show `"ok": false`, investigate the specific error before proceeding

Example of a failed node:

```
"node004": {
  "status": {
    "ok": false,
    "diag_msg": "BMC connection timeout"
  }
}
```

2.5.3.5.7.17 PRS

Power Reservation Steering (PRS) is a product optionally included and directly integrated with DPS that performs real-time power allocation adjustment for resource groups. PRS consumes telemetry provided by DPS to generate recommendations for power allocations at a per-GPU level, which DPS validates against the PDN constraints before applying. PRS can be controlled in a few ways via DPS:

- Not deployed (configured via Helm chart values option `.Values.prs.enabled`)
- Toggled via global configuration in the DPS WebUI
- Toggled per ResourceGroup during creation with option `--prs-enabled=false` It is recommended to deploy PRS and leave it enabled globally. For Max-P Resource Group, we recommend disabling PRS and leaving it on for all others.

2.5.3.5.7.18 Further Reading

- *Resource Group Concepts* - Understanding resource groups
- *Resource Group Priority* - Default 0, steal versus giveback, and floors
- *Resource Group CLI Commands* - Detailed command reference
- *Power Policies* - Understanding power policy hierarchy

2.5.3.5.8 Setup on BCM clusters

2.5.3.5.8.1 DPS setup on BCM clusters

This document describes the DPS setup steps on clusters with BCM versions 10 and 11.

This guide assumes you have already installed Kubernetes in your BCM environment and you have the appropriate permissions and kubeapi access. DPS has been tested on Kubernetes 1.30.11, 1.31.7, 1.32.3.

2.5.3.5.8.2 Prepare for DPS installation

- Kyverno configuration options:
  - Update the Kyverno policies. Recommended for Production
    - Installing DPS will reveal which policies need to be modified.
  - Delete the policies - *easiest method for dev and test*

```
kubectl get clusterpolicies

kubectl delete clusterpolicies <POLICYNAME>
```

  - Remove Kyverno or do not install Kyverno

2.5.3.5.8.3 Configure and Install DPS

- Plan your DPS configuration
  - Download the Helm chart from NGC. Check DPS Helm Charts on NGC for available versions.

```
# Set your version (e.g., 0.9.0)
VERSION="0.9.0"

# Download the chart
helm fetch https://helm.ngc.nvidia.com/nvidia/charts/dps-${VERSION}.
↪tgz

# Extract the chart
tar -xzf dps-${VERSION}.tgz
```

- The published DPS container images at `nvcr.io` are public. You do not need image pull credentials to install DPS.
- Install DPS

```
helm install dps ./dps \
  --namespace dps \
  -f <YOUR VALUES>.yaml \
  --create-namespace
```

2.5.3.5.8.4 Links

1. DPS on NGC

2.5.3.5.9 Integrating with Slurm

2.5.3.5.9.1 Overview

Integration with HPC workload managers like Slurm can be achieved through a scheduler prolog/epilog configuration. Each HPC job will be represented in DPS as a resource group. The prolog handles creation of the resource group with the selected DPS policy bundle and resource-group settings such as WPPS. The epilog handles the deletion of the resource group and restoring any default policies. If DPS detects insufficient power to start the job at the requested settings it may make adjustments or return a failure.

2.5.3.5.9.2 Example Slurm Integration

You can use the **PrologSlurmctld** and **EpilogSlurmctld** parameters in the *slurm.conf* file.

With this configuration a job will be re-queued if the DPS prolog fails (returns a non-zero exit code). While there can be various reasons for this failure, in general DPS was unable to configure a corresponding resource group with the requested power settings and the job should not run.

```
#slurm.conf
PrologSlurmctld=/usr/share/dps/prolog.sh
EpilogSlurmctld=/usr/share/dps/epilog.sh
```

2.5.3.5.9.3 Example Prolog script

The prolog script runs **before** job execution and:

1. **Creates** a resource group with the job ID as the external identifier
2. **Adds** allocated compute nodes to the resource group
3. **Activates** the effective policies from the selected bundle

```
#!/bin/bash
# Create DPS resource group for power management
JOB_NAME=${SLURM_JOB_NAME}
JOB_ID=${SLURM_JOB_ID}
NODES=${SLURM_JOB_NODELIST}

# Convert Slurm nodelist to comma-separated format
NODE_LIST=$(scontrol show hostname ${NODES} | tr '\n' ',' | sed 's/,$//')

# Create resource group
dpsctl resource-group create \
```

(continues on next page)

(continued from previous page)

```
  --resource-group "${JOB_NAME}" \
  --external-id ${JOB_ID} \
  --policy-bundle MaxP

# Add allocated nodes
dpsctl resource-group add \
  --resource-group "${JOB_NAME}" \
  --entities "${NODE_LIST}"

# Activate power policies
dpsctl resource-group activate \
  --resource-group "${JOB_NAME}" \
  --sync
```

2.5.3.5.9.4 Example Epilog Script

The epilog script runs **after** job completion and deletes the resource group, returning devices to their original power configuration set in the topology.

```
#!/bin/bash
# Slurm Epilog - Clean up resource group

JOB_NAME=${SLURM_JOB_NAME}

# Delete resource group (automatically deactivates)
dpsctl resource-group delete \
  --resource-group "${JOB_NAME}"
```

2.5.3.5.9.5 Job Comment Integration

The following `dps_*` comment keys are a site-defined convention for a custom prolog; they are not built-in Slurm fields or `dpsctl` flags. Keep `dps_policy` only when existing submitters and the prolog already share that schema. The prolog must map its value to an installed policy bundle and pass that name through `dpsctl resource-group create --policy-bundle`.

A site-specific prolog can parse DPS settings from job comments:

```
# Submit job with DPS settings in comment
sbatch --comment="dps_policy:MaxP,dps_prs:false,dps_dpm:true" job_script.sh
```

The prolog can map these site-defined values:

- `dps_policy:<string>` - Site-defined policy-bundle selection consumed by the prolog
- `dps_prs:<bool>` - Disable Power Reservation Steering
- `dps_dpm:<bool>` - Enables Dynamic Power Management
- `dps_wpps:<comma-separated ints>` - Sets workload profile IDs

2.5.3.5.9.6 Troubleshooting

See the Slurm documentation for information on Prolog and Epilog.

The DPS prolog/epilog scripts log to STDOUT by default.

Configure the Slurm service account for your server's authentication setup by following *Configure Server Authentication* and *Service Accounts*. Verify access under the account that runs the prolog and epilog scripts.

2.5.3.5.9.7 Integration with Workload Managers

2.5.3.5.9.8 Slurm Integration Compliance

When integrated with Slurm, DPS compliance works seamlessly:

1. **Job Submission**: User submits job with power requirements
2. **Prolog Execution**: Slurm prolog script calls DPS to create resource group
3. **Compliance Check**: DPS validates power compliance for allocated nodes
4. **Allocation Adjustment**: If allowed, DPS reduces the allocation within the effective bundle limits
5. **Job Execution**: Job runs with compliant power configuration
6. **Epilog Cleanup**: Slurm epilog script removes resource group, freeing power budget

This integration ensures that computational workloads never compromise power infrastructure integrity while maximizing available performance within compliance constraints.

2.5.3.5.10 Credentials and Secrets Configuration

2.5.3.5.10.1 Overview

The DPS server requires various secrets for secure operations, including database credentials, authentication details, and BMC credentials depending on the configured store.

2.5.3.5.10.2 Secrets

Create Secret contents through your approved secret-management workflow. Do not put passwords, tokens, or private keys in Helm values or source control. For Helm deployments, prepare these Secrets as applicable:

- A gRPC server identity referenced by `dps.serverTLS.existingSecret`, with `tls.crt`, `tls.key`, and `ca.crt`.
- API and UI ingress identities referenced by `dps.ingress.tls` and `ui.ingress.tls`, with `tls.crt` and `tls.key`.
- PostgreSQL credentials and trust material as described in *Database Secrets* and *Database TLS*.
- LDAP bind, certificate, and JWT signing material as described in *LDAP Credentials*.
- Direct Redfish credentials as described in *Kubernetes Secret Store*.

2.5.3.5.10.3 Database Secrets

Database Credentials: Essential for the PostgreSQL database connection. These are always file-mounted in the DPS server filesystem.

The DPS server looks for database credentials using the following process:

Username:

- Read from `--db-user` command line parameter, or
- Read from `DB_USERNAME` environment variable, or
- Defaults to "dps" if neither is set

Password:

- Read from file: `<DB_SECRET_MOUNT_PATH>/postgres/password`
- If file reading fails, falls back to credentials store (if configured)
- There is no raw `--db-password` flag or `DB_PASSWORD` environment variable.

File Structure:

```
<DB_SECRET_MOUNT_PATH>/postgres/
    password    # Database password (required)
```

Example of the Password File Contents:

```
# /home/nonroot/secrets/postgres/password
<database-password>
```

The database connection settings can be configured using command line parameters or environment variables. The password itself is file-mounted.

Command Line Parameters

```
dps-server \
  --db-host="localhost" \
  --db-port=5432 \
  --db-name="dps" \
  --db-user="dps" \
  --db-secret-mount-path="/home/nonroot/secrets"
```

Environment Variables

```
export DB_HOST="localhost"
export DB_PORT=5432
export DB_NAME="dps"
export DB_USERNAME="dps"
export DB_SECRET_MOUNT_PATH="/home/nonroot/secrets"
# Password file expected at: /home/nonroot/secrets/postgres/password
dps-server
```

For PostgreSQL certificate verification and client-certificate authentication, refer to *Database TLS*.

Helm Chart Configuration

```
global:
  postgresql:
    auth:
      database: "dps"
      username: "dps"
      existingSecret: "dps-postgresql"
```

The DPS server reads the key selected by `global.postgresql.auth.secretKeys.passwordKey`, which defaults to `password`. The chart-managed PostgreSQL workload reads the key selected by `global.postgresql.auth.secretKeys.userPasswordKey`, which defaults to `database-password`. When chart-managed PostgreSQL is disabled, include the second key only when another enabled chart component requires it.

2.5.3.5.10.4 LDAP Credentials

LDAP Credentials: Required if LDAP authentication is enabled. These include the bind DN, bind password, and TLS certificates for secure LDAP communication. LDAP configuration allows DPS to authenticate users against an external directory service.

Command Line Parameters

```
dps-server \
  --ldap-enabled=true \
  --ldap-server-url="ldaps://openldap.dps.svc.cluster.local:636" \
  --ldap-bind-dn="cn=root,dc=cm,dc=cluster" \
  --ldap-bind-password-file="/home/nonroot/secrets/dps-openldap/bind-password
↪" \
  --ldap-default-role="" \
  --ldap-tls-ca-cert="/home/nonroot/secrets/dps-openldap/ca.crt" \
  --ldap-tls-client-cert="/home/nonroot/secrets/dps-openldap/client.crt" \
  --ldap-tls-client-key="/home/nonroot/secrets/dps-openldap/client.key"
```

Environment Variables

```
export LDAP_ENABLED=true
export LDAP_SERVER_URL="ldaps://openldap.dps.svc.cluster.local:636"
export LDAP_BIND_DN="cn=root,dc=cm,dc=cluster"
export LDAP_BIND_PASSWORD_FILE="/home/nonroot/secrets/dps-openldap/bind-
↪password"
export LDAP_DEFAULT_ROLE=""
export LDAP_TLS_CA_CERT="/home/nonroot/secrets/dps-openldap/ca.crt"
export LDAP_TLS_CLIENT_CERT="/home/nonroot/secrets/dps-openldap/client.crt"
export LDAP_TLS_CLIENT_KEY="/home/nonroot/secrets/dps-openldap/client.key"
dps-server
```

Helm Chart Configuration

LDAP Authentication and Secret Configuration

```
dps:
  auth:
    authenticatorType: "ldap"
    auth_required_groups: "<dps-admin-group>,<dps-automation-group>"
```

(continues on next page)

(continued from previous page)

```
    jwt:
      existingSecret: "dps-jwt-keys"
  ldap:
    enabled: true
    serverUrl: "ldaps://ldap.example.com:636"
    bindDn: "<bind-dn>"
    baseDn: "<base-dn>"
    bindPassword:
      existingSecret: "dps-ldap-bind"
      existingSecretKey: "bind-password"
    defaultRole: ""
    groupRoleMapping: "<dps-admin-group>=admin,<dps-automation-group>
↪=automation"
    certSource: "file"
    tlsCaCert: "/home/nonroot/secrets/dps-openldap/ca.crt"
    tlsClientCert: "/home/nonroot/secrets/dps-openldap/client.crt"
    tlsClientKey: "/home/nonroot/secrets/dps-openldap/client.key"
    volumeMounts:
      - name: ldap-certs
        mountPath: "/home/nonroot/secrets/dps-openldap"
        readOnly: true
    volumes:
      - name: ldap-certs
        projected:
          sources:
            - secret:
                name: "dps-ldap-certs"
                items:
                  - key: ca.crt
                    path: ca.crt
                  - key: client.crt
                    path: client.crt
                  - key: client.key
                    path: client.key
```

Map each permitted identity to exactly one group listed in `dps.auth.auth_required_groups`. Protected API requests fail authorization when an identity matches zero or multiple listed groups, even if login succeeds.

Create `dps-ldap-bind` with the `bind-password` data key and `dps-ldap-certs` with the CA, client certificate, and client key shown. Create `dps-jwt-keys` with `private-key.pem`, which must contain an RSA private key in PEM format of at least 2,048 bits. Use your approved Secret workflow; do not put the bind password or private keys in `values.yaml`.

2.5.3.5.10.5 BMC Redfish API Credentials

The DPS server requires BMC API credentials for each managed node in the topology. It can read these credentials from BCM, Kubernetes Secrets, or files.

2.5.3.5.10.6 Credentials Store Types

- **bcm**: The Base Command Manager (BCM) is NVIDIA's cluster management solution that provides centralized management of datacenter resources, including secure storage and retrieval of BMC (Baseboard Management Controller) credentials for Redfish API access. This store uses the BCM API to fetch credentials. Requires BCM version 11. Use this store when your infrastructure is managed by BCM to enable seamless integration and centralized credential management.
- **k8sSecret**: Use in Kubernetes environments for native secret management. Supports in-cluster and cross-namespace access.
- **file**: Use for secrets stored as files. You can mount these files into a pod or place them directly on the filesystem. Works with static credentials that don't change often. Can be used in both Kubernetes and bare-metal environments.

Choose based on your environment: BCM for managed clusters, Kubernetes secrets for cloud-native, file for simple file-based setups.

2.5.3.5.10.7 Credentials Store Configuration

2.5.3.5.10.8 Setting the Credentials Store Type

The Credentials Store Type can be configured using the `--bmc-credentials-store` parameter or by setting the `BMC_CREDENTIALS_STORE` environment variable. This configuration accepts the following values:

- `bcm`
- `k8sSecret`
- `file`

Default Configuration:

- If no credentials store type is specified, DPS server defaults to `k8sSecret`
- This means BMC Redfish API credentials will be fetched from Kubernetes secrets using the native Kubernetes API
- The server will attempt to retrieve BMC credentials from Kubernetes secrets in the same namespace
- If no BMC credentials are found in Kubernetes secrets, BMC Redfish API operations will not be available

Below you can find detailed configuration examples for each credentials store type.

2.5.3.5.10.9 Credentials Store Options

2.5.3.5.10.10 BCM Store

Create the BCM password and TLS material through an approved Secret workflow. Do not pass the password on the command line or through a shell environment variable.

```
# Helm Example
dps:
  credentialsStore: "bcm"
  bcm:
    baseURL: "https://master.dps.svc.cluster.local:8443"
```

(continues on next page)

(continued from previous page)

```
    auth:
      basic:
        username: "root"
        passwordSecret:
          existingSecret: "bcm-auth"
          secretKeys:
            passwordKey: "password"
      cert:
        clientCertPath: "/secrets/bcm/client.cert"
        clientKeyPath: "/secrets/bcm/client.key"
        caCertPath: "/secrets/bcm/ca.cert"
  secrets:
    - name: "dps-bcm-certs"
      secretKey: "client.cert"
      mountPath: "/secrets/bcm/client.cert"
    - name: "dps-bcm-certs"
      secretKey: "client.key"
      mountPath: "/secrets/bcm/client.key"
    - name: "dps-bcm-certs"
      secretKey: "ca.cert"
      mountPath: "/secrets/bcm/ca.cert"
```

Kubernetes Secret Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: dps-bcm-certs
type: Opaque
data:
  client.cert: <base64-encoded-client-cert>
  client.key: <base64-encoded-client-key>
  ca.cert: <base64-encoded-ca-cert>
```

Configuration Parameters:

- `BMC_CREDENTIALS_STORE`: Set to “bcm”
- `BCM_BASE_URL`: BCM server headnode API URL (required)
- `BCM_USERNAME`: Username for basic authentication (optional if using cert auth)
- `BCM_PASSWORD`: Password for basic authentication (optional if using cert auth)
- `BCM_CLIENT_CERT`: Path to client certificate (optional if using basic auth)
- `BCM_CLIENT_KEY`: Path to client key (optional if using basic auth)
- `BCM_CA_CERT`: Path to CA certificate (optional)

2.5.3.5.10.11 Kubernetes Secret Store

```
# Environment Variables Example
BMC_CREDENTIALS_STORE=k8sSecret
# No additional parameters needed - secrets configured in cluster
```

```
# Command Line Example
dps-server --bmc-credentials-store=k8sSecret
```

```
# Helm Example
dps:
  credentialsStore: "k8sSecret"
  k8sSecret:
    labelSelector: "app.kubernetes.io/component=bmc-credentials"
```

Configuration Parameters:

- BMC_CREDENTIALS_STORE: Set to "k8sSecret"
- Each BMC Secret must match the configured label selector. The default is `app.kubernetes.io/component=bmc-credentials`.
- The Secret name must match `Redfish.SecretName` in the topology, and its `bmc` data field must contain a JSON or YAML map with `username` and `password`.

Create one Secret for each credential boundary in the same namespace as DPS. Replace the placeholders through your approved Secret workflow, and do not commit a populated manifest.

Kubernetes BMC Secret example

```
apiVersion: v1
kind: Secret
metadata:
  name: node001-bmc
  namespace: dps
  labels:
    app.kubernetes.io/component: bmc-credentials
type: Opaque
stringData:
  bmc: |
    {
      "username": "<bmc-user>",
      "password": "<bmc-password>"
    }
```

Keep `dps.k8sSecret.labelSelector` at its restrictive default or replace it with an equally restrictive site label. When a Redfish proxy provides backend BMC authentication, follow *Redfish Proxy Configuration* instead of creating a direct BMC Secret for DPS.

2.5.3.5.10.12 File Store

The File Store reads BMC credentials from a specific filesystem structure. When you set SECRET_MOUNT_PATH="`/secrets`", the store expects the following directory structure:

```
/secrets/
    baremetal/
        node1/
            bmc
        node2/
            bmc
```

Where:

- node1, node2 are the hostnames of your BMC nodes
- bmc is a file containing Redfish API credentials in JSON or YAML format

Deployment Options:

- **Kubernetes**: Mount secrets as volumes using Helm (recommended for containerized deployments)
- **Bare Metal**: Create directory structure directly on filesystem (for non-containerized deployments)

Example of the bmc file content:

```
{
  "username": "<bmc-user>",
  "password": "<bmc-password>"
}
```

Or in YAML format:

```
username: <bmc-user>
password: <bmc-password>
```

```
# Environment Variables Example
BMC_CREDENTIALS_STORE=file
SECRET_MOUNT_PATH="/secrets"
```

```
# Command Line Example
dps-server \
  --bmc-credentials-store=file \
  --secret-mount-path="/secrets"
```

```
# Helm Example
dps:
  credentialsStore: "file"
  dbSecretsMountPath: "/secrets"
  extraVolumeMounts:
    - name: bmc-credentials-node1
      mountPath: "/secrets/baremetal/node1"
      readOnly: true
    - name: bmc-credentials-node2
      mountPath: "/secrets/baremetal/node2"
      readOnly: true
  extraVolumes:
    - name: bmc-credentials-node1
      secret:
        secretName: "redfish-node1"
        items:
          - key: "bmc"
            path: "bmc"
            mode: 0400
    - name: bmc-credentials-node2
      secret:
```

(continues on next page)

(continued from previous page)

```
        secretName: "redfish-node2"
        items:
          - key: "bmc"
            path: "bmc"
            mode: 0400
```

Kubernetes Secret Examples:

```
apiVersion: v1
kind: Secret
metadata:
  name: redfish-node1
type: Opaque
data:
  bmc: <base64-encoded-bmc-credentials>

---
apiVersion: v1
kind: Secret
metadata:
  name: redfish-node2
type: Opaque
data:
  bmc: <base64-encoded-bmc-credentials>
```

Configuration Parameters:

- `BMC_CREDENTIALS_STORE`: Set to "file"
- `SECRET_MOUNT_PATH`: Base directory where secrets are mounted (required)

2.5.3.5.10.13 Best Practices

- Use Kubernetes Secrets for sensitive data.
- Keep certificate verification enabled under the relevant integration, such as `dps.redfish.tls.insecureSkipVerify: false` for direct Redfish.
- Regularly rotate secrets.

2.5.3.5.11 Adding Nodes to DPS

2.5.3.5.11.1 Overview

This guide provides step-by-step instructions for adding new compute nodes to an existing DPS (Dynamic Power Software) configuration. The process involves creating BMC credentials, updating the topology configuration, and ensuring the new nodes are properly integrated into the power management system.

2.5.3.5.11.2 Prerequisites

- DPS server running and accessible
- `dpsctl` installed and authenticated through *Configure Server Authentication*
- Existing topology already configured and active

- Access to the new node's BMC (Baseboard Management Controller)
- BMC credentials for the new node
- Device specifications for the new node type (if not already defined)

2.5.3.5.11.3 Step 1: Verify Current Configuration

Before adding new nodes, verify your current DPS configuration:

```
# Check current topology status
dpsctl topology list

# List current entities
dpsctl topology list-entities

# Check active topology
dpsctl topology list --active=true
```

Example Output:

```
{
  "topologies": [
    {
      "topology_name": "datacenter",
      "is_active": true,
      "leaf_node_names": ["node001", "node002"]
    }
  ]
}
```

2.5.3.5.11.4 Step 2: Create BMC Credentials Secret

DPS requires BMC credentials to communicate with each node. Copy the example in *Kubernetes Secret Store* to `node003-secret.yaml`, then set `metadata.name` to the exact `Redfish.SecretName` used for the node.

After replacing the placeholders, apply the Secret through your approved workflow. For a direct Kubernetes workflow:

```
kubectl apply -f node003-secret.yaml
```

Confirm that the object exists and has the selector label:

```
kubectl get secrets --namespace dps \
  --selector app.kubernetes.io/component=bmc-credentials
```

2.5.3.5.11.5 Step 3: Verify Device Specifications

Ensure the device specification for your new node type exists in DPS:

```
# List available device specifications
dpsctl device list
```

If the device specification does not exist, import it:

```
# Import device specifications (if needed)
dpsctl device upsert devices.yaml
```

Example device specification for DGX GB200:

```
- type: ComputerSystem
  description: NVIDIA GB200 Compute Tray (Bianca)
  model: DGX_GB200
  spec:
    devices:
      - type: CPU
        model: Grace
        count: 2
      - type: GPU
        model: GB200
        count: 4
    minLoadWatts: 1000
    maxLoadWatts: 6667
    processorModulesCount: 2
```

2.5.3.5.11.6 Step 4: Create a Replacement Topology Configuration

Create a topology file that includes the existing nodes and the new nodes. Give the topology a new name because `dpsctl topology import` rejects a name that already exists.

2.5.3.5.11.7 Option A: Export Current Topology and Modify

```
# Export current topology
dpsctl topology export --topology datacenter > current-topology.json
```

Edit the exported file to add the new node and change `Topology.Name`:

```
{
  "Entities": [
    {
      "Type": "PowerDomain",
      "Name": "PD-A",
      "OperatingLimit": {
        "PowerValue": {"Value": 1150000, "Type": "W"},
        "PowerFactor": 0.9
      }
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_GB200",
      "Name": "node001",
      "Redfish": {
        "URL": "https://node001-bmc.example.com",
        "SecretName": "node001"
      }
    },
    {
```

(continues on next page)

(continued from previous page)

```
      "Type": "ComputerSystem",
      "Model": "DGX_GB200",
      "Name": "node002",
      "Redfish": {
        "URL": "https://node002-bmc.example.com",
        "SecretName": "node002"
      }
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_GB200",
      "Name": "node003",
      "Redfish": {
        "URL": "https://node003-bmc.example.com",
        "SecretName": "node003"
      }
    }
  ],
  "Topology": {
    "Name": "datacenter-expanded",
    "Entities": [
      {
        "Name": "PD-A",
        "Children": ["node001", "node002", "node003"]
      },
      {
        "Name": "node001"
      },
      {
        "Name": "node002"
      },
      {
        "Name": "node003"
      }
    ]
  }
}
```

2.5.3.5.11.8 Option B: Create a New Topology File

Create `updated-topology.json` with all nodes, including the new node. Use a topology name that does not already exist.

2.5.3.5.11.9 Step 5: Validate the Updated Topology

Validate the topology file before importing:

```
# Validate the updated topology
dpsctl topology validate updated-topology.json
```

Expected Output:

```
{
  "status": {
    "ok": true,
    "diag_msg": "Topology validation passed"
  }
}
```

If validation fails, fix the errors before proceeding. For validation and import details, refer to *Import a Topology*.

2.5.3.5.11.10 Step 6: Import the Replacement Topology

Import the validated topology:

```
dpsctl topology import updated-topology.json
```

Expected Output:

```
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

Import creates the missing node entity and a new inactive topology. It leaves the current topology active. Because topology JSON does not contain a policy bundle field, import does not copy the current topology's persisted selection. The server resolves the replacement topology selection from the current `global_policy_bundle`. If you must preserve a different explicit selection, create the replacement through `CreateUpdateTopology` and set `CreateUpdateTopologyRequest.CreateRequest.policy_bundle`.

2.5.3.5.11.11 Step 7: Verify the Import

Verify that the replacement topology contains the new node and remains inactive. An unfiltered `dpsctl topology list` still includes the current active topology. Confirm the replacement with `--active=false`. JSON output omits `is_active` when the value is `false`. The persisted `policy_bundle` is the current `global_policy_bundle` value, `MaxLPS` on a default deployment.

```
dpsctl topology list-entities
dpsctl topology list --active=false
```

Expected Output:

```
{
  "topologies": [
    {
      "topology_name": "datacenter-expanded",
      "leaf_node_names": ["node001", "node002", "node003"],
      "policy_bundle": "MaxLPS"
    }
  ]
}
```

2.5.3.5.11.12 Step 8: Switch to the Replacement Topology

Deactivate the current topology, then activate the replacement. Activation configures the replacement topology's nodes on the agents.

```
dpsctl topology deactivate --topology datacenter
dpsctl topology activate --topology datacenter-expanded --ping-hosts
```

Flags:

- `--ping-hosts` Test connectivity of all hosts (recommended)
- `--at-least-percent-hosts` Minimum percent of hosts that must be reachable (default: 50)

Note

DPS 0.9 requires an active topology before `dpsctl check connection` or `dpsctl verify bmc-health start` can check nodes. Passing `--topology` does not bypass this requirement. Confirm the replacement topology appears in `dpsctl topology list --active`, and reactivate it if it was deactivated during the update:

```
dpsctl topology activate --topology datacenter-expanded
```

2.5.3.5.11.13 Step 9: Test Node Connectivity

After activation, test connectivity to the new node's BMC:

```
dpsctl check connection --topology datacenter-expanded --nodes node003
```

Expected Output:

```
{
  "total_nodes": 1,
  "success_nodes": 1
}
```

You can also test connectivity to all nodes in the topology:

```
dpsctl check connection --topology datacenter-expanded
```

Expected Output:

```
{
  "total_nodes": 3,
  "success_nodes": 3
}
```

2.5.3.5.11.14 Step 10: Verify Power Management

Verify that the new node is properly integrated into power management:

```
# Check node status
dpsctl check status
```

(continues on next page)

(continued from previous page)

```
# Confirm that the selected policy bundle exists
dpsctl policy bundle list
```

2.5.3.5.11.15 Troubleshooting

2.5.3.5.11.16 Common Issues and Solutions

2.5.3.5.11.17 1. BMC Connection Failures

Symptoms:

- Node connection check fails
- Power policy application errors

Solutions:

- Verify BMC credentials are correct
- Check network connectivity to BMC
- Ensure BMC is accessible from DPS server
- Verify BMC Redfish API is enabled

```
# Test BMC connectivity manually
curl -k -u admin:password https://node003-bmc.example.com/redfish/v1
```

2.5.3.5.11.18 2. Device Specification Not Found

Symptoms:

- Validation errors about unknown device type/model

Solutions:

- Import the required device specifications
- Verify the device type and model match existing specifications

```
# Import device specifications
dpsctl device upsert devices.yaml
```

2.5.3.5.11.19 3. Secret Not Found

Symptoms:

- Authentication errors when connecting to BMC

Solutions:

- Verify the secret name matches the `SecretName` in the topology
- Check that the secret is in the correct namespace
- Ensure the secret contains the correct `bmc` key

```
# Verify secret exists
kubectl get secret node003 -n dps

# Confirm the Secret has the required key without printing its value
kubectl describe secret node003 -n dps
```

2.5.3.5.11.20 4. Topology Name Already Exists

Symptoms:

- Import fails with `failed to create topology`.

Solutions:

- Choose a new `Topology.Name` for the replacement topology.
- Run `dpsctl topology list` to confirm that the name is unused.
- Revalidate and import the file.

2.5.3.5.11.21 5. Power Policy Application Issues

Symptoms:

- Nodes not responding to power policy changes
- Power limit errors

Solutions:

- Verify the node supports the specified power policy plugin
- Check that the BMC supports the required Redfish endpoints
- Ensure the device specification includes the correct `powerPolicyPlugin`

2.5.3.5.11.22 Related Documentation

- ***Import a Topology*** - Topology validation and import
- ***Managing Topologies*** - Post-import topology management
- ***Credentials and Secrets Configuration*** - Detailed secrets management
- ***Device Specifications*** - Understanding device capabilities
- ***Entities*** - Node configuration details
- ***Redfish API*** - BMC communication protocols

2.5.3.5.12 Backing up with Velero

2.5.3.5.12.1 Overview

DPS supports Kubernetes native backup, migration, and disaster recovery solutions. DPS recommends establishing a regular backup schedule tailored to your organizational needs and Recovery Point Objective.

2.5.3.5.12.2 Example Using Velero

Velero is a popular open-source tool for backing up and restoring Kubernetes resources. This example demonstrates a basic backup and restore process using Velero with MinIO.

1. Create a Velero Backup:

```
velero backup create dps-backup --include-namespaces dps
```

2. Verify Backup Creation:

```
velero backup describe dps-backup
```

This command displays detailed information about the backup, including its status, start time, and included resources. Ensure the `phase` indicates success.

3. Restore from Backup:

```
velero restore create --from-backup dps-backup
```

This command initiates a restore process using the previously created backup.

4. Verify DPS Deployment:

```
kubectl get all -n dps
```

Confirm that the DPS deployment has been successfully restored by listing all resources in the `dps` namespace. Check that all expected resources are present and running as expected.

2.5.3.5.13 Managing NvGrid

2.5.3.5.13.1 Overview

This guide provides step-by-step instructions for managing power constraints using DPS NvGrid.

For more information, refer to *NvGrid Concepts*.

Note: NvGrid is typically integrated with grid management systems or demand response platforms. This guide demonstrates manual operations for testing and understanding the workflow. For complete integration examples, refer to the *Grid Integrator Guide*.

2.5.3.5.13.2 Prerequisites

- DPS server running with NvGrid enabled
- `dpsctl` installed and authenticated through *Configure Server Authentication*
- Active topology with power feed configuration

2.5.3.5.13.3 Basic NvGrid Workflow

2.5.3.5.13.4 Step 1: Discover Power Feeds

```
# Get feed metadata
dpsctl nvgrid get-metadata
```

Output shows:

- Feed tags (identifiers for electrical circuits)

- Power minimum and maximum limits
- Default constraints

2.5.3.5.13.5 Step 2: Schedule Load Target

```
# Set 6MW target for 1 hour
dpsctl nvgrid set-load-target \
  --value=6 \
  --unit=megawatt \
  --start-time="2025-10-24T15:00:00Z" \
  --end-time="2025-10-24T16:00:00Z" \
  --best-effort=true

# Target specific feeds
dpsctl nvgrid set-load-target \
  --value=5 \
  --unit=megawatt \
  --start-time="2025-10-24T15:00:00Z" \
  --end-time="2025-10-24T16:00:00Z" \
  --feed-tags="feed-a1,feed-a2" \
  --correlation-id="curtailment-001"

# Reset to default constraint
dpsctl nvgrid set-load-target --default
```

2.5.3.5.13.6 Step 3: Query Schedule

```
# View load schedule for next 4 hours
dpsctl nvgrid get-schedule \
  --start-time="2025-10-24T14:00:00Z" \
  --end-time="2025-10-24T18:00:00Z"

# Get current active target
dpsctl nvgrid get-current
```

2.5.3.5.13.7 Step 4: Monitor Status

```
# Check current power status
dpsctl nvgrid get-status
```

Status shows:

- Current load target per feed
- Calculated power consumption
- In-flight status (whether power adjustment is in progress)
- Correlation ID of active constraint

2.5.3.5.13.8 Webhook Integration

2.5.3.5.13.9 Register Webhook

```
# Register webhook for all events
dpsctl nvgrid register-webhook \
  --url="https://grid-solution.example.com/nvgrid/webhook"

# Register for specific events
dpsctl nvgrid register-webhook \
  --url="https://grid-solution.example.com/nvgrid/webhook" \
  --events="start-ramp-down,end-ramp-down"
```

Webhook Requirements:

- URL must be in the NvGrid webhook whitelist (configured in Helm values)
- Endpoint must accept HTTP POST with JSON payload
- Must return 200 OK for successful receipt

2.5.3.5.13.10 Unregister Webhook

```
dpsctl nvgrid unregister-webhook \
  --url="https://grid-solution.example.com/nvgrid/webhook"
```

2.5.3.5.13.11 Advanced Scenarios

2.5.3.5.13.12 Overlapping Schedules

Most recently scheduled target takes precedence:

```
# Schedule 6MW from 4pm-7pm
dpsctl nvgrid set-load-target --value=6 --unit=megawatt \
  --start-time="2025-10-24T16:00:00Z" --end-time="2025-10-24T19:00:00Z"

# Schedule 8MW from 5pm-6pm (overrides previous)
dpsctl nvgrid set-load-target --value=8 --unit=megawatt \
  --start-time="2025-10-24T17:00:00Z" --end-time="2025-10-24T18:00:00Z"
```

Result:

- 4pm-5pm: 6MW active
- 5pm-6pm: 8MW active (most recent override)
- 6pm-7pm: 6MW active (reverts to earlier schedule)
- After 7pm: Default constraint

2.5.3.5.13.13 Resetting to Default Constraint

```
# Reset to default power constraint for a time period (using --default)
dpsctl nvgrid set-load-target \
  --default \
  --start-time="2025-10-24T16:00:00Z" \
```

(continues on next page)

(continued from previous page)

```
  --end-time="2025-10-24T19:00:00Z"

# Reset to default (using --value=0, equivalent)
dpsctl nvgrid set-load-target \
  --value=0 \
  --unit=watt \
  --start-time="2025-10-24T16:00:00Z" \
  --end-time="2025-10-24T19:00:00Z"

# Reset immediately and indefinitely
dpsctl nvgrid set-load-target --default

# Reset specific feeds
dpsctl nvgrid set-load-target \
  --default \
  --feed-tags="feed-a1,feed-a2"
```

Note: The `--default` flag resets the power constraint to the default (zero) value. This is mutually exclusive with `--value` and `--unit`. Alternatively, you can achieve the same result by setting `--value=0` with any unit (e.g., `--value=0 --unit=watt`).

2.5.3.5.13.14 Monitoring

2.5.3.5.13.15 Prometheus Metrics

NvGrid exposes metrics for monitoring:

- `grpc_duration` - API response time
- `grpc_requests` - API request count
- `nvgrid_system_status_load_target_watts` - Current load target per feed
- `nvgrid_system_status_calculated_load_watts` - DPS-calculated load per feed
- `nvgrid_system_status_in_flight` - Power event status
- `nvgrid_feed_schedule_load_target_watts` - Scheduled targets with status

Query these metrics from Prometheus to track grid integration health and power management effectiveness.

2.5.3.5.13.16 Troubleshooting

2.5.3.5.13.17 Webhook Registration Fails

Error: “webhook URL not whitelisted”

Solution: Add the webhook URL to the whitelist in DPS Helm values:

```
dps:
  nvgrid:
    webhook:
      whitelist:
        - "https://your-grid-solution.example.com/webhook"
```

2.5.3.5.13.18 Load Target Not Applied

Check:

1. Verify schedule is active: `dpsctl nvgrid get-current`
2. Check system status: `dpsctl nvgrid get-status`
3. Ensure DPM is enabled on resource groups
4. Verify workloads are running (power can only be reduced from active jobs)

2.5.3.5.13.19 Target Not Met

NvGrid uses `best_effort` strategy:

- Target may not be achieved if it requires terminating jobs
- Check `in_flight` status - power adjustment may still be in progress
- Verify sufficient DPM-enabled workloads exist for power reduction

2.5.3.5.13.20 Further Reading

- *NvGrid Concepts* - Understanding NvGrid architecture
- *NvGrid CLI Reference* - Complete command documentation
- *Resource Groups* - Workload power management

2.5.3.5.14 Database TLS

2.5.3.5.14.1 Overview

The DPS server connects to PostgreSQL using the `pgx` driver, which supports TLS modes from plain-text (`disable`) through full certificate verification (`verify-full`) and PostgreSQL client-certificate authentication (mutual TLS).

Enable database TLS when any of the following apply:

- PostgreSQL uses a certificate issued by a private or internal CA that is not in the pod's system trust roots.
- Your operations policy requires `sslmode=verify-full` (hostname verification).
- PostgreSQL is configured with `cert` authentication and requires a client certificate and private key instead of (or in addition to) a password.

This guide covers:

- Installation — which Helm values and environment variables control database TLS.
- Enablement — step-by-step to turn on server-cert verification and/or mutual TLS.
- Troubleshooting — certificate rotation, handshake-failure diagnostics, and the `dps_db_tls_handshake_failures_total` metric.

2.5.3.5.14.2 Installation

Database TLS is configured under `global.postgresql` in `values.yaml` and by a matching set of environment variables or CLI flags on the `dps-server` binary.

2.5.3.5.14.3 Helm values

Values key	Purpose	Default
`global.postgresql.sslMode`	Postgres `sslmode` — one of `disable`, `allow`, `prefer`, `require`, `verify-ca`, `verify-full`	`""` (effective default `disable`)
`global.postgresql.sslRootCert`	Path inside the pod to a PEM CA certificate used to verify the Postgres server	`""` (feature off; uses system roots)
`global.postgresql.sslCert`	Path inside the pod to a PEM client certificate (mutual TLS)	`""`
`global.postgresql.sslKey`	Path inside the pod to a PEM client private key (mutual TLS; pair with `sslCert`)	`""`

All three certificate and key fields hold filesystem paths, not certificate contents. Mount the material through `dps.secrets`. Leave `global.postgresql.sslMode` unset, or set it to `disable`, only for deployments whose approved network design does not use PostgreSQL TLS.

For 0.9 overlays, move `database.sslMode`, `database.sslRootCert`, `database.sslCert`, and `database.sslKey` to the corresponding `global.postgresql.*` keys. The chart reads the nested keys.

2.5.3.5.14.4 Environment variables

Variable	Maps to	Notes
`DB_SSLMODE`	`global.postgresql.sslMode`	PostgreSQL `sslmode`
`DB_TLS_CA_FILE`	`global.postgresql.sslRootCert`	Path to the CA certificate
`DB_TLS_CERT_FILE`	`global.postgresql.sslCert`	Path to the client certificate
`DB_TLS_KEY_FILE`	`global.postgresql.sslKey`	Path to the client private key

2.5.3.5.14.5 CLI flags

```
dps-server \
  --db-ssl-mode="verify-full" \
  --db-tls-ca-file="/secrets/postgres/ca.crt" \
  --db-tls-cert-file="/secrets/postgres/client.crt" \
  --db-tls-key-file="/secrets/postgres/client.key"
```

2.5.3.5.14.6 Enablement

The following walk-through enables `sslmode=verify-full` with both a CA certificate and mutual-TLS client credentials. Adjust the set of files you mount depending on whether you need server-cert verification only, client auth only, or both.

2.5.3.5.14.7 1. Create the Kubernetes Secret

Place your CA cert, client cert, and client key into a single `Opaque` Secret:

```
kubectl create secret generic dps-postgres-tls \
  --from-file=ca.crt=/path/to/ca.crt \
  --from-file=client.crt=/path/to/client.crt \
  --from-file=client.key=/path/to/client.key \
  -n dps
```

If you only need server-cert verification (no client auth), omit `client.crt` and `client.key`.

2.5.3.5.14.8 2. Set the Helm values

In your `values.yaml` overlay:

```
global:
  postgresql:
    sslMode: "verify-full"
    sslRootCert: "/secrets/postgres/ca.crt"
    sslCert: "/secrets/postgres/client.crt"
    sslKey: "/secrets/postgres/client.key"

dps:
  secrets:
    - name: dps-postgres-tls
      secretKey: ca.crt
      mountPath: /secrets/postgres/ca.crt
    - name: dps-postgres-tls
      secretKey: client.crt
      mountPath: /secrets/postgres/client.crt
    - name: dps-postgres-tls
      secretKey: client.key  # pragma: allowlist secret
      mountPath: /secrets/postgres/client.key  # pragma: allowlist secret
```

The paths under `global.postgresql` must match the `mountPath` entries in `dps.secrets`. The Helm template emits the corresponding `DB_TLS_*` certificate environment variables into the `dps-server` pod only when the values are non-empty. If `global.postgresql.sslMode` is left unset, the chart defaults to `disable`. Use `verify-ca` or `verify-full` for server-certificate verification.

2.5.3.5.14.9 3. Apply the chart

```
helm upgrade --install dps ngc/dps \
  --namespace dps \
  --version "<release-version>" \
  --values values.yaml \
  --wait
```

2.5.3.5.14.10 4. Verify the handshake

After the `dps-server` pod becomes ready, confirm that:

- The pod log contains a `postgres connecting` entry with `sslmode=verify-full` and `mtls_enabled=true` (neither the DSN nor the password is ever logged).
- The `dps_db_tls_handshake_failures_total` counter is 0 (scrape `/metrics` or query Prometheus directly).
- Server health endpoints respond normally (no repeated restart loop).

If the counter is non-zero or the pod fails to start, refer to *Troubleshooting*.

2.5.3.5.14.11 Troubleshooting

2.5.3.5.14.12 Certificate rotation

Certificate rotation is not automatic. After rotating a certificate file on disk, the `dps-server` pod must be restarted to pick up the new certificate (the server loads X.509 key pairs once at startup). Integrate the restart with your approved certificate-rotation procedure.

Operationally, this means:

```
kubectl rollout restart statefulset/dps-server -n dps
```

after any rotation of the ca.crt / client.crt / client.key contents inside the Secret.

2.5.3.5.14.13 Handshake-failure metric

`dps_db_tls_handshake_failures_total` is a counter with a bounded `reason` label. It increments once per failed TLS handshake during pool initialization:

reason label	Meaning	Typical cause
cert_pars	CA, client cert, or client key could not be parsed as PEM	Wrong file format, truncated secret, or CRLF injected into PEM
cert_veri	Server cert chain does not chain to the configured CA	Wrong CA, server cert issued by a different CA, or server cert expired
host-name_mism	`sslmode=verify-full` and the server cert SAN does not match the connection host	The configured Postgres host (`global.postgresql.host` / `DB_HOST`) does not match the cert SAN — use a hostname that matches the server certificate
proto-col	TLS negotiation failed at protocol level	TLS version mismatch, cipher mismatch, or Postgres not configured for TLS
auth_reje	TLS handshake succeeded but Postgres rejected the client cert / role	Client cert CN does not match a Postgres role, or `pg_hba.conf` entry requires password

Query example:

```
sum by (reason) (rate(dps_db_tls_handshake_failures_total[5m]))
```

Non-zero rate on any label is an operational issue. `cert_parse` is almost always a Secret-contents problem; `cert_verify` / `hostname_mismatch` are CA or server-cert issues; `auth_rejected` points at Postgres server-side `pg_hba.conf`.

2.5.3.5.14.14 Common failures

- **Pod CrashLoopBackOff with "open /secrets/postgres/...: no such file"** — a path under `global.postgresql` is a typo or does not match any `mountPath` in `dps.secrets`. Correct the path and reapply.
- **Pod CrashLoopBackOff with "permission denied"** — the mounted key file mode, owner, or pod security context does not let the DPS process read it. For the default non-root DPS pod, prefer a group-readable Secret projection such as `0440` with `dps.securityContext.fsGroup` set to the reader group. Use `0400` only when the file owner is the container UID or the file is otherwise guaranteed readable by that UID.
- **`sslcert` supplied but no `sslkey`** — pgx rejects this at `ParseConfig`. Supply both or neither.

2.5.3.5.14.15 Related pages

- *Credentials and Secrets Configuration*
- *Deploy Guide*
- *Troubleshooting*

2.5.4. Integrating DPS

Dynamic Power Software (DPS) sits between the systems that already run your data center — schedulers, inventory databases, BMCs, and grid services — and the hardware whose power it manages. This page routes to the authoritative guide for each integration.

Documentation for several of these exists in more than one place. Where that is the case, this page names the page to work from.

2.5.4.1 Integrate with a workload scheduler

DPS does not place workloads. It manages the power available to them, so a scheduler remains the owner of what runs where.

- ***Integrating with Slurm*** — the procedure for connecting DPS to Slurm. Start here.
- ***Slurm*** — how the integration works and what DPS expects the scheduler to own.
- ***Workload Scheduling Integrator SDK*** — for building an integration with a scheduler DPS does not already support.

2.5.4.2 Integrate with an inventory or cluster manager

DPS needs to know what hardware exists and how it is connected. Rather than maintaining that inventory separately, import it from the system that already holds it.

- ***Import External Entities*** — import devices from Nautobot or Base Command Manager. Start here.
- ***DPS Nautobot Integration - Current Use Cases*** — how Nautobot records map onto DPS entities.
- ***Base Command Manager*** — how DPS works alongside BCM.

- ***Installing DPS in BCM*** — for a cluster that BCM already manages.

2.5.4.3 Integrate with the electrical grid

Grid integration lets a utility signal a constraint and lets DPS respond by reducing draw across the fleet.

- ***Managing NvGrid*** — schedule and manage grid power constraints. Start here for operating an existing integration.
- ***NvGrid*** — the model behind grid constraints and load targets.
- ***Grid Integrator SDK*** — for building a grid intelligence service against the NvGrid gRPC API.

2.5.4.4 Integrate with BMCs

- ***Redfish API*** — how DPS uses Redfish to read telemetry and apply limits.
- ***Redfish Proxy Configuration*** — for BMCs that are not directly reachable from the DPS services.

2.5.4.5 Build against the API

Everything DPS does is available over its gRPC API, and the simulator serves the same interface as real hardware.

- ***gRPC API reference*** — the `nvidia.dcpower.v1` service and message definitions.
- ***Developer Guides*** — building custom integrations and extensions.
- ***Optimization Developer SDK*** — for developing power and performance optimization systems.
- ***DPS SDK and Simulator*** — run the simulator first. Integrations built against it target the same API as production.

2.5.4.6 Related

- ***Deploying DPS*** — install DPS against your hardware.
- ***Operating DPS*** — day-to-day operation once an integration is running.

2.5.5. Troubleshooting

Use this guide when DPS returns an error, a health check fails, or a component does not behave as expected. Start with *Quick Diagnosis*, then jump to the symptom that matches what you observed.

This page covers operator-facing failures across installation, authentication, topology, resource groups, baseboard management controller (BMC) connectivity, the database, and Power Reservation Steering (PRS). For step-by-step recovery during the NVL72 inference power pilot, refer to *Troubleshoot the MaxLPS power pilot*.

2.5.5.1 Safety and Evidence

Preserve evidence before you change production state:

- Capture the full `dpsctl` JSON response, including `status.diag_msg` and `status.error.code`. That `code` value is the DPS error code, such as `AUTHENTICATION_FAILED` or `DATABASE_ERROR`. Prefer the code over matching the entire message string, because messages can change between releases. On the gRPC wire the same value appears as `ErrorInfo.reason`; `dpsctl` JSON does not emit a `reason` field for that code.

- Record the DPS server version (`dpsctl server-version`), the `dpsctl` client version, the Helm chart version, the Kubernetes namespace, and the topology or resource-group name.
- Collect only the log lines that surround the failure. Redact passwords, tokens, BMC credentials, certificate private keys, and any workload or tenant identifiers that are not required to diagnose the issue.
- Prefer read-only commands (`dpsctl verify`, `dpsctl check connection`, `dpsctl topology list`, `kubectl get`, `kubectl logs`) until you have identified a documented remediation.
- This guide does not authorize mutating commands such as `dpsctl topology activate`, `dpsctl topology deactivate`, `dpsctl topology remove`, `dpsctl resource-group activate`, `dpsctl resource-group delete`, BMC reset, `nvidia-smi -pl`, `kubectl rollout restart`, or `helm upgrade`. Use an approved procedure that defines preconditions, recovery verification, and rollback. Topology deactivation and resource-group deletion do not restore GPU Total Graphics Power (TGP).

2.5.5.2 Quick Diagnosis

Run these checks in order. If a check fails, use the matching section. The UI Ingress is not in the control path for BMC, Redfish, or PRS operations, so an unhealthy `ui` result does not prevent you from collecting evidence for those incidents.

1. Confirm `--host`, `--port`, and TLS flags match the deployed API endpoint. Refer to the *dpsctl global options*.
2. Authenticate through the method configured for your deployment. Use *dpsctl login* for LDAP or OIDC user sign-in, or *Service Accounts* for service credentials. For certificate authentication, follow *Configure Service-Specific mTLS*.

   If authentication fails, go to *Cannot Connect or Log In*. Run `dpsctl verify --auth` only after an API request succeeds.
3. Confirm the client can reach the API:

   ```
   dpsctl --host <dps-api-host> --port <grpc-port> server-version
   ```
4. Check component health:

   ```
   dpsctl verify
   ```

   The response can include `dps_server`, `database`, `auth`, `ui`, and `bcm` `ComponentStatus` objects. Default-valued fields are omitted from `dpsctl` JSON, so use `status.ok` and the fields that are present instead of matching an expected set of `false` values. On a deployment without a registered Base Command Manager (BCM) check, the `bcm` object contains only `"name": "BCM"`, and `status.ok` is `false` with a generic deployment-verification diagnostic. To recheck one component, pass its flag, for example `dpsctl verify --database`.
5. If BMC or Redfish symptoms remain, confirm the intended topology appears in `dpsctl topology list --active=true` before checking node reachability. DPS 0.9 requires an active topology for this check, even with `--topology`. If it is inactive, skip this check and follow *Managing Topologies* through your approved activation procedure.

   ```
   dpsctl check connection --topology <topology-name>
   ```
6. Inspect Kubernetes objects in the DPS namespace (default `dps`):

```
kubectl get pods -n dps
kubectl logs -n dps -l app=dps-server --all-containers --since=15m --
↪tail=200
```

2.5.5.3 Errors and Diagnosis

2.5.5.3.1 Cannot Connect or Log In

Use this section when `dpsctl` cannot open a gRPC channel, `dpsctl login` fails, or commands return authentication or authorization codes.

Reported codes can include `AUTHENTICATION_FAILED`, `AUTHENTICATION_INVALID_CREDENTIALS`, `AUTHENTICATION_USER_NOT_FOUND`, `AUTHENTICATION_SERVICE_UNAVAILABLE`, `AUTHORIZATION_FAILED`, `NETWORK_ERROR`, and `DEADLINE_EXCEEDED`.

Use these diagnosis steps:

1. Confirm `--host`, `--port`, and TLS flags (`--ca-cert-path`, `--client-cert-path`, `--client-key-path`) match the deployed API endpoint. Refer to the *dpsctl global options*.
2. Match the login flow to the deployment. For LDAP, confirm the account can bind and matches the configured search base and user filter. For OIDC, confirm the issuer, client ID, and requested scopes, and use browser login. Refer to *Configure LDAP Mode* or *Configure OIDC Mode*.
3. If a token is rejected, check its expiry and the server's token configuration. For DPS-issued tokens, check the signing key. For OIDC, check issuer, audience, scopes, client ID restrictions, and JWKS access.
4. If login succeeds but an API request returns `AUTHORIZATION_FAILED`, check `dps.auth.auth_required_groups`. When this setting is nonempty, the identity must match exactly one listed group. OIDC group extraction must match the provider's group claim. In external authentication mode, inspect the proxy's authorization decision. Repeating login does not change access.
5. For mTLS, check that the client certificate chains to the trusted CA and its common name is allowed for the requested service. Confirm the network path preserves the certificate through TLS passthrough. Refer to *Configure Service-Specific mTLS*.
6. After API access succeeds, use `dpsctl verify --auth` to check LDAP connectivity or local JWT configuration. This check does not verify OIDC provider configuration or external proxy policies. Use API results and the relevant server or proxy logs for those setups.
7. If the authentication service is temporarily unavailable, wait and retry. Persistent `AUTHENTICATION_SERVICE_UNAVAILABLE` requires an administrator to inspect the service and its dependencies.

2.5.5.3.2 Deployment Component Unhealthy

Use this section when `dpsctl verify` reports `healthy: false` for a component after *Deploy DPS*.

The following table maps each component to evidence to collect:

Com-po-nent	What to check	Potential cause
dps_s	`kubectl get pods -n dps` and `kubectl logs -n dps -l app=dps-server --all-containers --since=15m --tail=200`	Pod not `Running`, crash loop, or API not reachable from the client
datab	Postgres pod, connection secret, and *Database TLS*	Reachability, credentials, or TLS handshake failure (`DATABASE_ERROR`)
auth	LDAP connectivity, bind credentials, and certificates, or local JWT configuration	Bind failure or missing configuration. This check does not validate OIDC or external proxy authentication.
ui	`dps-ui` Ingress in the same namespace, with a host and at least one rule	Missing Ingress, DNS, or TLS for the UI hostname
bcm	Whether BCM is part of the deployment, then its credential secret and network path	If no BCM check is registered, the `bcm` object contains only `"name": "BCM"` and makes `status.ok` `false`. If BCM is expected, this result indicates that the integration is not registered.

After an approved configuration change, rerun only the affected component, for example `dpsctl verify --dps-server --database`.

2.5.5.3.3 Topology Validation or Import Fails

Use this section when `dpsctl topology validate` or `dpsctl topology import` fails. The validate command returns `ValidationError` entries whose `error` field identifies the failure class. Import performs the same semantic checks during topology creation, but reports them as a gRPC error rather than returning the `validation_errors` array.

This guide covers the following validation classes:

- `invalid_model` — the file does not match the topology JSON schema.
- `device_not_found` — an entity `Type` and `Model` pair is not in the device registry. Run `dpsctl device list` and correct `Model` to an exact match.
- `invalid_name` / `invalid_secret_name` — disallowed characters in an entity, topology, policy, or Kubernetes Secret name.
- `duplicate_entity` — two entities share the same `Name`.
- `referenced_entity_not_found` / `disconnected_graph` / `circular_dependency` — the parent and child graph is incomplete or cyclic.
- `invalid_connection` — a parent device type cannot legally own the listed child.

Fix the file and rerun `dpsctl topology validate` until it reports success, then import. If import returns `failed to create topology`, run `dpsctl topology list` first. An existing topology with the same name produces that message and `DATABASE_ERROR`; retrying the unchanged import cannot resolve the collision. If the name is not present, collect `dpsctl verify --database` and server logs to distinguish connectivity or persistence failures.

For the full validation list in the NVL72 pilot, refer to *Why Does Topology Validation or Import Fail?*. For schema and import guidance, refer to *Import a Topology*.

2.5.5.3.4 Topology or Resource Group Lifecycle Fails

Use this section when `dpsctl topology activate`, `dpsctl resource-group add`, or related commands fail.

Reported codes can include `NOT_FOUND`, `ALREADY_EXISTS`, `VALIDATION_ERROR`, `NODE_NOT_IN_TOPOLOGY`, `DEVICE_NOT_IN_RESOURCE_GROUP`, and `INVALID_OPERATION`.

Use these diagnosis steps:

1. `topology <name> not found` or `NOT_FOUND` — the `--topology` value does not match an imported topology. Run `dpsctl topology list`.
2. `topology <name> is already active` — the topology is already the control path. Confirm with `dpsctl topology list --active=true` and skip activate. Do not deactivate as a diagnosis step. Deactivate does not restore GPU TGP.
3. `no active topologies` / `INVALID_OPERATION` when adding resource-group members — run `dpsctl topology list --active=true` and record the intended topology. Do not activate a topology from this general diagnosis path.
4. `some entities are not in the active topology` / `INVALID_OPERATION` — correct the node names against the active topology. If the topology omitted intended nodes, create a corrected replacement topology or use the `CreateUpdateTopology` API update operation.
5. `resource(s) not in topology` / `NODE_NOT_IN_TOPOLOGY` — the listed names are absent from the topology. Correct `--entities` against `dpsctl topology list`.
6. `some entities are already in the resource group` / `ALREADY_EXISTS` — those nodes are already members of this group. Remove duplicates from the request.
7. `some entities are reserved by another allocating resource group` — the code depends on the operation. Adding resources returns `ALREADY_EXISTS`; activating a resource group returns `INVALID_OPERATION`. Record the conflicting group metadata and stop. Do not deactivate another group from this general diagnosis path.
8. `resource(s) not in resource group` / `DEVICE_NOT_IN_RESOURCE_GROUP` — a remove request listed nodes that are not members of this group. Correct the names; do not delete another resource group.
9. `power budget exceeded ... reprovision is false` — admission detected a new or worsened overload in the proposed device or ancestor allocation. Capture the effective policy, active resource groups, topology limits, and the reported device metadata. `dps.prs.headroomPercent` does not determine this admission result. Do not change policy caps or topology budgets from this general diagnosis path.
10. Per-node failures in a `--sync` activate response — those BMCs rejected the set-limit request. With the topology active, run a node-scoped read-only BMC health check:

    ```
    dpsctl verify bmc-health start --topology <topology-name> --nodes <node1>,
    ↪<node2> --skip-writes --wait --summary-only
    ```

    Check the overall `status.ok` value after the task completes. If it is `false`, the server cleans up partially activated nodes. If the request succeeds under its partial-activation rule, accepted nodes remain active. The default rule requires at least 80% of hosts. In either case, query the resource group again instead of inferring final state from individual request results.

Refer to *Managing Resource Groups* for the intended lifecycle, and to *What If Resource Group Creation Fails?* in the NVL72 runbook for the exact pilot command failures.

2.5.5.3.5 BMC or Redfish Calls Fail

Use this section when topology activation, resource-group activation, or `dpsctl check connection` reports BMC failures, or when codes such as `REDFISH_REQUEST_UNAUTHORIZED`, `REDFISH_REQUEST_ERROR`, or `NETWORK_ERROR` appear.

DPS 0.9 requires the topology to be active before `dpsctl check connection` or `dpsctl verify bmc-health start` can probe its nodes. Passing `--topology` or `--skip-writes` does not bypass this requirement. If the command returns `INVALID_OPERATION` with `no agent configuration available`, check `dpsctl topology list --active`. If the intended topology is inactive, complete the activation procedure from *Managing Topologies* before retrying. Continue network and credential diagnosis without these probes if activation has failed.

Use these diagnosis steps:

1. Confirm BMC network reachability from the cluster and that the Kubernetes BMC credential secret exists for the node. Refer to *Credentials and Secrets Configuration*.
2. `REDFISH_REQUEST_UNAUTHORIZED` — the BMC rejected the credentials or the session. Rotate or correct the secret; do not embed passwords in topology JSON.
3. Connection timeouts — confirm routing, firewalls, and any *Redfish Proxy Configuration*.
4. For a read-only deep probe of firmware, power-limit drift, Workload Power Profiles Settings (WPPS), Electrical Design Point (EDPp), and BMC latency, start a node-scoped check. Bare `dpsctl verify bmc-health` only lists subcommands. Do not pass `--force-writes` from this page:

   ```
   dpsctl verify bmc-health start --topology <topology-name> --nodes <node-
   ↪name> --skip-writes --wait --summary-only
   ```

   Use *Run a BMC Health Check* to interpret the report and decide whether a write-enabled check is appropriate. Issue-specific BMC reset and in-band `nvidia-smi -pl` remediation in *BMC Health Check* applies only to the NVL72 inference power pilot. Issue codes such as `BMC_UNREACHABLE`, `IB_OOB_LIMIT_DRIFT`, and `POWER_WRITE_READBACK_MISMATCH` are documented in *BMC Health FAQ*. Use the write-cycle check, BMC reset, and in-band `nvidia-smi -pl` only when you have the authorization required for those changes.

Mutating BMC remediations (reset, in-band `nvidia-smi -pl`) change GPU power limits. Do not run them from this general troubleshooting path.

2.5.5.3.6 Database Errors

Use this section when `dpsctl verify --database` is unhealthy, pods crash on Postgres connect, or commands return `DATABASE_ERROR`.

Use these diagnosis steps:

1. Confirm the Postgres pod or external instance is running and that the connection secret matches `global.postgresql`.
2. Confirm network connectivity from `dps-server` to the database host and port.
3. For TLS failures, inspect `dps_db_tls_handshake_failures_total` and the `reason` label (`cert_parse`, `cert_verify`, `hostname_mismatch`, `protocol`, `auth_rejected`). Refer to *Troubleshooting*.

4. After rotating CA or client certificates, record the mounted Secret version and the `dps-server` pod start time. The process must reload changed key pairs, but this guide does not prescribe a restart without an approved deployment rollback.
5. Review Postgres logs and `dps-server` logs for the same timestamp. Do not paste connection strings or passwords into an evidence bundle.

2.5.5.3.7 PRS Is Not Updating GPU Limits

Use this section when the Power Steering controller is not Ready, reports reconciliation errors, or the observed BMC/DCGM limit time series does not move under load.

Use these diagnosis steps:

1. Confirm the resource group is active with `prs_enabled: true`.
2. For the bundled Power Steering deployment, confirm both `power-steering.enabled: true` and `dps.prs.enabled: true` in the Helm values. An external Power Steering deployment can disable the subchart only when `dps.prs.hostPort` and `dps.prs.controllerHostPort` point to the external services.
3. Check the Power Steering controller logs:

```
kubectl logs -n dps \
  -l app.kubernetes.io/component=controller,app.kubernetes.io/part-
↪of=power-steering \
  -c controller --since=15m --tail=200
```

4. If BMC latency is high, record `dps.prs.schedulerIntervalSeconds` and the measured latency. Do not change the scheduler interval from this general diagnosis path. Refer to *How Do I Check Power Steering?* and *Account for Slow BMCs*.

2.5.5.4 Error Code Index

`dpsctl` JSON responses put the lookup key in `status.error.code`. On the gRPC wire, the same value is `ErrorInfo.reason` with domain `dcpower.nvidia.com`. Use the code, not the full message. The following table maps codes covered by this guide to a diagnosis section:

Code	Start here
AUTHENTICATION_FAILED	*Cannot Connect or Log In*
AUTHENTICATION_INVALID_CREDENTIALS	*Cannot Connect or Log In*
AUTHENTICATION_USER_NOT_FOUND	*Cannot Connect or Log In*
AUTHENTICATION_SERVICE_UNAVAILABLE	*Cannot Connect or Log In*
AUTHORIZATION_FAILED	*Cannot Connect or Log In*
NETWORK_ERROR	*Cannot Connect or Log In* or *BMC or Redfish Calls Fail*
DEADLINE_EXCEEDED	*Quick Diagnosis*; then BMC latency if the call was a Redfish operation
DATABASE_ERROR	*Database Errors*
FILE_ERROR	Confirm the referenced file exists, is readable, and is not truncated
NOT_FOUND	*Topology or Resource Group Lifecycle Fails*
ALREADY_EXISTS	*Topology or Resource Group Lifecycle Fails*
INVALID_OPERATION	*Topology or Resource Group Lifecycle Fails*
VALIDATION_ERROR	*Topology Validation or Import Fails*
NODE_NOT_IN_TOPOLOGY	*Topology or Resource Group Lifecycle Fails*
DEVICE_NOT_IN_RESOURCE_GROUP	*Topology or Resource Group Lifecycle Fails*
REDFISH_REQUEST_UNAUTHORIZED	*BMC or Redfish Calls Fail*
REDFISH_REQUEST_ERROR	*BMC or Redfish Calls Fail*
INVALID_CONFIGURATION / INVALID_ARGUMENT	Check Helm values, topology JSON, and command flags against the linked task guide
INTERNAL_ERROR	Collect redacted `dps-server` logs and *escalate*

Codes such as INVALID_POLICY_LIMIT_MIN and POWER_USAGE_EXCEEDED indicate a policy or allocation constraint. Capture the effective policy and reported device metadata instead of retrying the same activation request unchanged.

2.5.5.5 Escalation

Stop mutation attempts and use your approved escalation process when any of the following conditions is true:

- The error code is INTERNAL_ERROR, or it is not listed in this guide, and read-only diagnosis did not identify a safe fix.
- A documented remediation failed, or the system state is worse after the attempt.
- A BMC remains unreachable, unauthorized, or incompatible with the required firmware after the checks in *BMC or Redfish Calls Fail*. For the NVL72 inference power pilot, continue with *BMC Health FAQ*.

- The next step would mutate power limits, firmware, or cluster configuration without a documented rollback.
- You need a custom device definition for hardware that is not in the shipped device registry.

Do not keep applying mutating commands to gather more data. Preserve the last successful and last failed outputs.

2.5.5.5.1 What to Include

Prepare a redacted bundle that contains:

- DPS server version, `dpsctl` version, Helm chart version, and Kubernetes namespace.
- The failing command, timestamp (UTC), topology name, resource-group name, and node names.
- The customer-visible error code (`status.error.code` in `dpsctl` JSON) and `diag_msg`, with secrets removed.
- `dpsctl verify` output and, when relevant, `dpsctl check connection` or a BMC health-check summary.
- Redacted excerpts of `dps-server`, database, authentication, or PRS logs around the failure.
- Hardware model, BMC firmware, and whether the node is in the NVL72 pilot topology.

Never include passwords, tokens, private keys, full Secret YAML, or unredacted BMC URLs that embed credentials.

This guide does not define a customer support channel, support severity levels, or response times. Use your organization's approved incident and NVIDIA support process. If no approved NVIDIA case-intake route is established, preserve the evidence bundle and contact the owner of your DPS deployment.

2.5.5.6 Related Documentation

Use the following pages for detailed procedures and reference information:

- *Deploy DPS*
- *dpsctl verify*
- *dpsctl verify bmc-health*
- *Troubleshoot the MaxLPS power pilot*
- *Database TLS*
- *Credentials and Secrets Configuration*
- *Redfish API*

2.5.6. Reference

- ***dpsctl Commands*** - Complete command-line interface reference
- ***API Documentation*** - gRPC API and protocol buffer definitions
- ***Schema Reference*** - JSON schema for topology validation

2.5.6.1 APIs

This section contains the API documentation for Dynamic Power Software (DPS).

- ***nvidia.dcpower.v1*** - Current stable API version

2.5.6.1.1 Interacting with the gRPC API

2.5.6.1.1.1 Using dpsctl

Configure client access for your server's setup using *Configure Server Authentication*. The following `dpsctl` example uses LDAP username/password login:

LDAP Login and API Requests

```
# Authenticate with DPS
dpsctl login --username alice

# List all topologies
dpsctl topology list

# Get server version
dpsctl server-version

# List resource groups
dpsctl resource-group list

# Check system status
dpsctl verify --dps-server
```

2.5.6.1.1.2 Using grpcurl

For direct gRPC API access, use `grpcurl`. The following example obtains a DPS-issued token from an LDAP deployment. For OIDC, obtain the token from your identity provider as described in *Configure OIDC Mode*.

Prepare a request file through your secret-management workflow. Include a `passwordCredential` object with `username` and `password` fields. The following command reads that file from standard input.

LDAP Token Retrieval and gRPC Requests

```
# Get a token first
grpcurl -d @ \
  api.dps.your.domain:443 \
  nvidia.dcpower.v1.AuthService/Token \
  < /path/to/password-credential.json

# Supply the returned access token through your secret-management workflow.
# The following requests use DPS_ACCESS_TOKEN.

# List all available services
grpcurl -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
```

(continues on next page)

(continued from previous page)

```
  api.dps.your.domain:443 \
  list

# Get server version
grpcurl -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
  api.dps.your.domain:443 \
  nvidia.dcpower.v1.VersionService/GetVersion

# List topologies
grpcurl -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
  api.dps.your.domain:443 \
  nvidia.dcpower.v1.TopologyManagementService/ListTopologies
```

2.5.6.1.1.3 nvidia.dcpower.v1

The current stable API version for NVIDIA Dynamic Power Software (DPS).

These pages are generated from the proto definitions in the `dcpower` repository. Do not edit them by hand; edit `docs/grpc_md.tmpl` instead.

2.5.6.1.1.4 v1/allocation_validation.proto

Allocation validation messages for dry-run allocation feasibility analysis.

This file contains messages for validating whether devices can be allocated with a policy bundle without actually creating a resource group or applying policies to hardware. Used for pre-flight checks and "what-if" allocation feasibility analysis.

2.5.6.1.1.5 Messages

2.5.6.1.1.6 ValidateAllocationRequest

ValidateAllocationRequest contains parameters for dry-run allocation feasibility validation.

Field	Type	Description
device_names	repeated `string`	Device names to validate. Must contain at least one item, and each entry must not be empty.
policy_bundle	`string`	Policy bundle name to evaluate. Must not be empty.
strict	`bool`	none

2.5.6.1.1.7 ValidateAllocationResponse

ValidateAllocationResponse is the streaming response for allocation validation First message contains overall validation result, subsequent messages contain per-device details

Field	Type	Description
oneof response.allocation_validation	`ValidateAllocationResponse.AllocationValidationResult`	Overall allocation validation result (sent once as first message)
oneof response.per_device_allocatic	`ValidateAllocationResponse.PerDeviceAllocationValidationRe`	Per-device allocation validation result (streamed per device)

2.5.6.1.1.8 ValidateAllocationResponse.AllocationPolicySelection

AllocationPolicySelection contains the policy-bundle selection outcome for this allocation.

Field	Type	Description
requested_policy_bundle	`strin`	none
selected_policy_bundle	`strin`	none
effective_policy_floor_selec	`bool`	Indicates that at least one device would be admitted at its effective policy floor instead of its target. The selected bundle remains unchanged.

2.5.6.1.1.9 ValidateAllocationResponse.AllocationPowerReclamationRequirements

AllocationPowerReclamationRequirements contains power reclamation requirements if allocation needs power from other RGs If this field is set (non-null), power reclamation from other RGs would be required for activation. If null, the allocation can proceed without affecting other resource groups. TODO: Not yet implemented - always nil in current version

Field	Type	Description
affected_resource_group_names	repeated `string`	none
watts_to_reclaim	`double`	none
diagnostic_note	`string`	none

2.5.6.1.1.10 ValidateAllocationResponse.AllocationValidationResult

AllocationValidationResult contains overall validation outcome (sent once as first message)

Field	Type	Description
status	`Status`	Operation status
allocation_would_succeed	`bool`	Validation outcome - would the allocation succeed?
allocation_policy_selection	`ValidateAllocationResponse.AllocationPolicySelection`	Policy-bundle selection for the allocation
allocation_power_reclamatio	`ValidateAllocationResponse.AllocationPowerReclamationRequire`	Power reclamation requirements (null if no reclamation needed)

2.5.6.1.1.11 ValidateAllocationResponse.DeviceAllocationStatus

DeviceAllocationStatus contains per-device allocation validation results

Field	Type	Description
policy_apply_statu	`PolicyApplyStatus`	For ComputerSystem entities - policy that would be applied
power_distribut	`ValidateAllocationResponse.PowerDistributionEntityStat`	For power distribution entities (PDU/PSU/PowerDomain) - capacity utilization
effective_policy_floo	`bool`	Indicates that the bundle would be admitted at the device's effective policy floor instead of its target (ComputerSystem only).

2.5.6.1.1.12 ValidateAllocationResponse.PerDeviceAllocationValidationResult

PerDeviceAllocationValidationResult contains validation result for a single device (streamed per device)

Field	Type	Description
device_id	`string`	Device identifier
device_allocation_statu	`ValidateAllocationResponse.DeviceAllocationStatus`	Allocation status for this device

2.5.6.1.1.13 ValidateAllocationResponse.PowerDistributionEntityStatus

PowerDistributionEntityStatus contains power distribution entity status for PDU/PSU/PowerDomain entities

Field	Type	Description
allocated_watts	`double`	none
capacity_watts	`double`	none

2.5.6.1.1.14 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.15 v1/auth.proto

2.5.6.1.1.16 Services

2.5.6.1.1.17 AuthService

AuthService provides authentication functionality

2.5.6.1.1.18 Token

rpc Token(`TokenRequest`) returns `TokenResponse`

Token generates new access and refresh tokens using credentials

2.5.6.1.1.19 RefreshToken

rpc RefreshToken(`RefreshTokenRequest`) returns `TokenResponse`

RefreshToken generates new access and refresh tokens using existing refresh token

2.5.6.1.1.20 Logout

rpc Logout(`LogoutRequest`) returns `LogoutResponse`

Logout revokes native browser session tokens.

2.5.6.1.1.21 Messages

2.5.6.1.1.22 LogoutRequest

LogoutRequest contains native session tokens to revoke.

Field	Type	Description
refresh_token	`string`	none
access_token	`string`	none

2.5.6.1.1.23 LogoutResponse

LogoutResponse confirms completion of an idempotent logout request.

2.5.6.1.1.24 PasswordCredential

PasswordCredential contains username and password credentials

Field	Type	Description
username	`string`	none
password	`string`	none

2.5.6.1.1.25 RefreshTokenRequest

RefreshTokenRequest contains refresh token for generating new tokens

Field	Type	Description
refresh_token	`string`	none

2.5.6.1.1.26 TokenRequest

TokenRequest contains credentials for token generation Currently only password credential is fully implemented

Field	Type	Description
oneof credential.password_credentia	`Password-Credential`	Uncomment when implementing API key authentication ApiKeyCredential api_key_credential = 2; Additional credential types can be added here

2.5.6.1.1.27 TokenResponse

TokenResponse contains generated access and refresh tokens

Field	Type	Description
access_token	`string`	none
refresh_token	`string`	none

2.5.6.1.1.28 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.29 v1/bmc_health.proto

BMC health check service messages.

Defines requests and responses used by the deep BMC pre-requisite health check, which validates Redfish endpoint reachability, firmware inventory, power setting controls, IB/OOB drift, EDPp/WPPS state, and telemetry across cluster nodes.

2.5.6.1.1.30 Messages

2.5.6.1.1.31 BMCHealthCheckHandle

BMCHealthCheckHandle is returned immediately from StartBMCHealthCheck. The task continues running in the background on the server; callers poll GetBMCHealthCheckStatus and GetBMCHealthCheckReport using task_id to observe progress and retrieve the final report.

task_id is stable for the lifetime of the task in the server's task manager. After the task completes it remains queryable for at most –zombie-lifetime (default 10m), after which a status or report lookup returns NotFound. A server restart drops all task state, so callers must re-run on restart.

Field	Type	Description
task_id	`string`	Unique task identifier; pass back to status / report RPCs.
name	`string`	Human-readable task name (e.g. "bmc_health_check").
status	`string`	Initial status string from the task manager (e.g. "Started").
started_at	`google.protobuf.Timestamp`	Server-side timestamp when the task was started.

2.5.6.1.1.32 BMCHealthCheckProgress

BMCHealthCheckProgress reports how far the runner has advanced through the in-scope entities. Both counters reset to zero between runs; they are not durable.

Field	Type	Description
nodes_total	`int32`	Total entities to probe; populated once the task has loaded its entity set from the topology store (zero until then).
nodes_comple	`int32`	Entities that have produced a response (real, synthesized timeout, or dispatch error).

2.5.6.1.1.33 BMCHealthCheckReportRequest

BMCHealthCheckReportRequest fetches the final report. If the task is still running this returns done=false with no payload; callers should continue polling (or use the dpsctl –wait shortcut).

Field	Type	Description
task_id	string	Task ID returned by StartBMCHealthCheck.
summary_only	bool	When true, the server strips per-node NodeReport entries from the response and returns only Status, ClusterSummary, and Issues. Useful on large clusters where the full per-node detail (firmware lists, endpoint stats, telemetry samples) inflates the payload past what an operator wants to scroll through, and for scripted polling where only the pass/fail summary matters. The underlying task is unaffected; a subsequent call with summary_only=false returns the full report (until the task exits the zombie window).

2.5.6.1.1.34 BMCHealthCheckReportResponse

BMCHealthCheckReportResponse is the response of GetBMCHealthCheckReport.

Field	Type	Description
done	bool	True when the underlying task is finished (success or failure). When false the report field is unset; poll again later.
status	string	Task manager status string at the time of the lookup (“Started” while running, “Done” / “Failed” once finished).
report	BMCHealthCheckResponse	Cluster-level result. Populated only when done=true and the task did not fail before any data could be collected.
error_messag	string	Human-readable error string from the task’s terminal error, if any. Empty on a clean run.

2.5.6.1.1.35 BMCHealthCheckRequest

BMCHealthCheckRequest is the request for running a deep BMC pre-requisite health check.

Concurrency note: the server enforces an at-most-one gate that is SERVER-WIDE, not topology- or node-scoped. While any BMC health check is in flight, a subsequent StartBMCHealthCheck for a different topology (or a disjoint Nodes subset) is rejected with AlreadyExists and the existing task_id. This is the intended first-version behavior because in practice topologies on the same server have a high node intersection and serializing protects shared BMCs from overlapping load and write-cycle interleaving. A future revision may loosen this to allow disjoint runs to proceed in parallel.

Field	Type	Description
topology_name	`string`	Name of the topology whose nodes are to be checked.
nodes	repeated `string`	Optional explicit list of node names to check. If empty, all nodes in the topology are checked.
oneof _samples_per_telemetry.samples_per_	optional `int32`	Optional number of telemetry samples to collect per node.
oneof _concurrency.concurrency	optional `int32`	Optional maximum number of nodes to check concurrently.
oneof _skip_writes.skip_writes	optional `bool`	Optional flag to skip write/restore operations on BMCs.
oneof _expected_edpp_pct.expected_edpp	optional `int32`	Expected steady-state EDPp current value, in percent. The screen rule flags a processor whose EDPp current_pct is below this floor. Default (when unset) is 100. Allowed range when set: 1..200.
oneof _telemetry_interval_ms.telemetry_interva	optional `int32`	Sleep between telemetry samples on each node, in milliseconds. Zero means no inter-sample delay (samples issued back-to-back). Allowed range when set: 0..60_000 (1 minute). The dpsctl CLI defaults to 500ms; direct API callers that omit the field get the server-side default (no delay, matching the legacy back-to-back semantics).
oneof _write_resolution_timeout_ms.wr	optional `int32`	Per-PATCH readback window for the write-cycle probe, in milliseconds. After each PATCH on PowerLimitWatts.SetPoint the probe polls the readback on a small backoff schedule until the value matches or this ceiling elapses. Zero (or unset) uses the bmcprobe default (1s). Operators on very slow BMC fleets can raise this, but values much above 5s tend to amplify failures on already-stuck BMCs rather than recover them. Allowed range when set: 0..60_000 (60s).
per_node_timeout	`google.protobuf.Duration`	Optional cumulative timeout for the complete health check of each node, including all telemetry samples. Redfish request and operation timeouts continue to bound each individual HTTP call. When unset, the server uses the current 5-minute default. Allowed range when set: 5 minutes..48 hours.
oneof _prs_latency_check.prs_latency_c	optional `PRSLatencyCheck`	Optional PRS-to-BMC roundtrip latency check (DO-007). Its presence is the opt-in: when set, the server patches the PRS scheduler interval (if interval_override is set), runs the BMC probe, then observes the PRS cluster_apply histogram over the window and stores the latency artifact in BMCHealthCheckResponse.prs_latency_artifact_json. A nil message skips the observation entirely.

2.5.6.1.1.36 BMCHealthCheckResponse

BMCHealthCheckResponse contains the results of a BMC pre-requisite health check.

Field	Type	Description
status	`Status`	none
node_reports	repeated `NodeReport`	none
cluster_summary	`ClusterSummary`	none
issues	repeated `Issue`	none
prs_latency_artifa	`bytes`	JSON-encoded latencyacceptance.Artifact from the DO-007 PRS latency check. Present only when the request set prs_latency_check.

2.5.6.1.1.37 BMCHealthCheckStatus

BMCHealthCheckStatus is the response of GetBMCHealthCheckStatus.

Field	Type	Description
task_id	`string`	Echoes the looked-up task_id.
name	`string`	Task name (constant for the lifetime of the task).
status	`string`	Task manager status string: “Init”, “Started”, “Done”, “Failed”.
started_at	`google.protobuf.Timestamp`	Time the task was started.
completed_at	`google.protobuf.Timestamp`	Time the task completed; unset while running.
diag_mess	`string`	Diagnostic message when the task panicked. Empty otherwise.
progress	`BMCHealthCheckProgress`	Coarse progress counters. Available even before completion.
done	`bool`	True when the task has reached a terminal state (success, failure, or cancellation) and the report is ready to fetch via GetBMCHealthCheckReport.
error_messa	`string`	Human-readable error string from the task’s terminal error, if any (e.g. a loadData failure). Empty when no error was recorded.

2.5.6.1.1.38 BMCHealthCheckStatusRequest

BMCHealthCheckStatusRequest queries the current status of a previously started BMC health check.

Field	Type	Description
task_id	`string`	Task ID returned by StartBMCHealthCheck.

2.5.6.1.1.39 ClusterSummary

ClusterSummary aggregates pass/fail counts and latency statistics across the cluster.

Field	Type	Description
total_nodes	`int64`	Nodes in the topology scope at run start (the full set of entities the task intended to probe). Populated by the aggregator from its input view count, independent of how many entries actually carried a usable NodeReport.
passed	`int64`	none
failed	`int64`	none
endpoint_latency_m	`DurationStats`	none
telemetry_latency_ms	`DurationStats`	none
power_get_laten	`DurationStats`	Aggregate latency, in milliseconds, for power-limit GETs across all nodes/processors.
power_set_laten	`DurationStats`	Aggregate latency, in milliseconds, for power-limit SETs across all nodes/processors.
edpp_get_latenc	`DurationStats`	Aggregate latency, in milliseconds, for EDPp percent reads across all nodes/processors.
wpps_get_latenc	`DurationStats`	Aggregate latency, in milliseconds, for WPPS endpoint reads across all nodes.
nodes_attempte	`int32`	Nodes the runner actually invoked a probe against, i.e. entries that reached the screening pass with a non-nil NodeReport. May be smaller than total_nodes if the aggregator dropped malformed entries.
nodes_unreachal	`int32`	Nodes for which probing failed before any sample was collected (BMC unreachable, auth, etc.).

2.5.6.1.1.40 DurationStats

DurationStats captures latency statistics, in milliseconds.

Field	Type	Description
count	int64	none
min_ms	int64	none
max_ms	int64	none
avg_ms	int64	none
p50_ms	int64	none
p95_ms	int64	none
p99_ms	int64	none

2.5.6.1.1.41 EDPpRead

EDPpRead captures the EDPp reading for a processor.

Field	Type	Description
processor	string	none
current_pct	int32	none
reference_pct	int32	none
reference_available	bool	none

2.5.6.1.1.42 EndpointStat

EndpointStat records request statistics for a single Redfish endpoint+method pair.

Field	Type	Description
endpoint	string	none
method	string	none
attempts	int64	none
successes	int64	none
failures	int64	none
duration	DurationStats	none

2.5.6.1.1.43 FirmwareEntry

FirmwareEntry describes a single firmware inventory entry from the BMC.

The node attribution is denormalized onto every entry (rather than only living on the enclosing NodeReport) so the firmware list is usable when flattened across the cluster – e.g. when correlating firmware versions against failing-node Issues, or when emitting a per-node firmware summary into structured logs without re-walking the parent report.

Field	Type	Description
id	`string`	Inventory ID as reported by the BMC (e.g. “BMC_0”, “HMC_0”, “ERoT_BMC_0”). Stable across probe runs.
version	`string`	Version string returned by the BMC. Empty when the inventory item existed but its version field could not be parsed.
node	`string`	Node name the firmware was read from. Always set by probeFirmware so consumers can filter the cluster’s firmware list by node without joining back to the parent NodeReport.

2.5.6.1.1.44 IBOOBDrift

IBOOBDrift captures the drift between in-band and out-of-band power values.

Field	Type	Description
component	`string`	none
set_point_w	`int64`	none
oneshot_w	`int64`	none
drift	`bool`	none

2.5.6.1.1.45 Issue

Issue describes a single problem detected during the health check.

Field	Type	Description
severity	`Severity`	none
code	`string`	none
node	`string`	none
resource	`string`	none
observed	`string`	none
threshold	`string`	none
message	`string`	none

2.5.6.1.1.46 NodeReport

NodeReport summarizes the BMC pre-requisite health check results for a single node.

Field	Type	Description
node	`string`	none
reachable	`bool`	none
error_msg	`string`	none
firmware	repeated `FirmwareEntry`	none
endpoint_stats	repeated `EndpointStat`	none
power_settings	repeated `PowerSettingRead`	none
power_writes	repeated `PowerWriteCycleStat`	none
ib_oob_drift	repeated `IBOOBDrift`	none
edpp	repeated `EDPpRead`	none
wpps	repeated `WPPSRead`	none
telemetry_samples	`int64`	Number of telemetry samples requested for this node. This is not the number completed. See the device.StdMetrics endpoint_stats entry for the actual attempt, success, and failure counts.
telemetry	repeated `TelemetryComponentStat`	none
telemetry_samples_comp	`int64`	Number of device.StdMetrics calls actually attempted. This includes successful and failed reads.
telemetry_completion_re	`TelemetrySamplingCompletionReason`	Why telemetry sampling stopped. Older agents leave this unspecified; the server infers it from device.StdMetrics endpoint statistics.

2.5.6.1.1.47 PRSLatencyCheck

PRSLatencyCheck configures the optional DO-007 PRS-to-BMC roundtrip latency observation run alongside a BMC health check.

Field	Type	Description
metrics_url	string	Prometheus /metrics URL on the PRS controller. When empty, the server derives it from the configured PRS controller host.
interval_override	google.protobuf.Duration	When set, overrides the PRS scheduler interval_seconds for the duration of the check and restores the original value on exit. Accelerates sample accumulation on low-traffic clusters.
window	google.protobuf.Duration	Observation window for PRS cluster_apply histogram samples. Zero uses the server default (5 minutes).
min_samples	int32	Minimum PRS cluster_apply histogram sample count required over the window. Zero uses the server default (1200).
actuation_p99_max	google.protobuf.Duration	Inclusive upper bound for PRS cluster_apply p99 actuation latency. Zero uses the server default (1s).
metrics_step	string	PRS controller operation histogram step. Empty uses “cluster_apply” (PRS SET□DPS response wall time). Use “pipeline” for the full iteration.

2.5.6.1.1.48 PowerSettingRead

PowerSettingRead captures the readable power setting state for a component.

Field	Type	Description
component	string	none
set_point_w	int64	none
allowable_max_w	int64	none
default_set_point_w	int64	none
power_w	double	none

2.5.6.1.1.49 PowerWriteCycleStat

PowerWriteCycleStat captures the results of an exercised power write/restore cycle.

Field	Type	Description
component	string	none
original_w	int64	none
decrement_cycles	int32	Number of decrement-by-1W iterations the probe attempted.
mismatches	int32	Total readback / set mismatches observed across all decrement and increment iterations plus the final restore readback.
restored	bool	none
restored_value_w	int64	none
restore_error	string	none
increment_cycles	int32	Number of increment-by-1W iterations the probe attempted as part of walking the SetPoint back up to the captured original.

2.5.6.1.1.50 TelemetryComponentStat

TelemetryComponentStat summarizes telemetry samples for a single component.

Field	Type	Description
component	string	none
power_w	ValueStats	none

2.5.6.1.1.51 ValueStats

ValueStats captures statistics over a numeric value (e.g. telemetry power).

Field	Type	Description
count	int64	none
min	double	none
max	double	none
mean	double	none
p50	double	none
p95	double	none
p99	double	none

2.5.6.1.1.52 WPPSRead

WPPSRead captures the workload power profile state for a processor.

Field	Type	Description
processor	`string`	none
accessible	`bool`	none
enforced_mask	`string`	none
requested_mask	`string`	none
supported_mask	`string`	none
available_profile_ids	repeated `int32`	none
error_msg	`string`	none
was_active	`bool`	True when EnforcedProfileMask was non-zero on probe – i.e. one or more workload power profiles were active on this processor at probe time. Lights-on validation requires no active profiles; the screen pass converts this into a WPPS_PROFILES_ACTIVE Issue.
active_profile_ids	repeated `int32`	The profile IDs decoded from the original EnforcedProfileMask, in ascending order. Empty when was_active is false. Captured before any remediation so operators always see exactly what was active at probe time, even if the reset succeeded.
reset_attempted	`bool`	True when the probe attempted to disable the active profiles. False when no profiles were active or when –skip-writes was set.
reset_succeeded	`bool`	True when the post-reset re-read confirmed EnforcedProfileMask is now zero. Meaningful only when reset_attempted is true.
reset_error	`string`	Diagnostic message when reset_attempted is true and reset_succeeded is false. Empty otherwise.

2.5.6.1.1.53 Enums

2.5.6.1.1.54 Severity

Severity classifies the impact level of an Issue raised during the health check.

Name	Number	Description
SEVERITY_UNSPECIFIED	0	none
SEVERITY_WARNING	1	none
SEVERITY_ERROR	2	none

2.5.6.1.1.55 TelemetrySamplingCompletionReason

TelemetrySamplingCompletionReason describes whether the requested telemetry sampling loop ran to completion. Failed reads do not make the loop incomplete when every requested device.StdMetrics call was attempted.

Name	Number	Description
TELEMETRY_SAMPLING_COMPLETION_REASON_UNSPECIFIED	0	none
TELEMETRY_SAMPLING_COMPLETION_REASON_COMPLETED	1	none
TELEMETRY_SAMPLING_COMPLETION_REASON_BEST_EFFORT	2	none
TELEMETRY_SAMPLING_COMPLETION_REASON_TIMED_OUT	3	none
TELEMETRY_SAMPLING_COMPLETION_REASON_CANCELED	4	none
TELEMETRY_SAMPLING_COMPLETION_REASON_INCOMPLETE	5	none

2.5.6.1.1.56 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.57 v1/common.proto

2.5.6.1.1.58 Messages

2.5.6.1.1.59 BestEffortCleanupWarning

BestEffortCleanupWarning records a cleanup error that was intentionally ignored so the primary operation could continue.

Field	Type	Description
kind	`string`	Cleanup warning category, for example “redfish_policy_cleanup”.
operation	`string`	Best-effort operation that produced the warning.
affected_object_type	`string`	Type of object affected by the cleanup warning.
affected_object_name	`string`	Name of the object affected by the cleanup warning.
affected_domain	`string`	Redfish/domain identifier affected by the cleanup warning.
affected_endpoint	`string`	Endpoint affected by the cleanup warning.
ignored_error	`string`	Error that was ignored to preserve the primary operation.

2.5.6.1.1.60 Status

Status denotes operation status

Field	Type	Description
ok	`bool`	Whether or not the operation was successful
diag_msg	`string`	Diagnostic msg, if any

2.5.6.1.1.61 ValidationError

ValidationError contains a single topology validation error

Field	Type	Description
object_name	`string`	The name of the object with the validation failure
object_type	`string`	The type of object with the validation failure
error	`string`	The type of error
message	`string`	The error message

2.5.6.1.1.62 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.63 v1/config.proto

2.5.6.1.1.64 Services

2.5.6.1.1.65 ConfigurationManagementService

ConfigurationManagementService handles global configuration settings

2.5.6.1.1.66 GetConfiguration

rpc GetConfiguration(`GetConfigurationRequest`) returns `GetConfigurationResponse`

2.5.6.1.1.67 UpsertConfiguration

rpc UpsertConfiguration(`UpsertConfigurationRequest`) returns `UpsertConfigurationResponse`

2.5.6.1.1.68 Messages

2.5.6.1.1.69 GetConfigurationRequest

GetConfigurationRequest is an empty request for getting configuration

2.5.6.1.1.70 GetConfigurationResponse

GetConfigurationResponse contains all global configuration settings

Field	Type	Description
status	`Status`	Operation Status
settings	map `GetConfigurationResponse.SettingsEntry`	Map of global configuration settings

2.5.6.1.1.71 GetConfigurationResponse.SettingsEntry

Field	Type	Description
key	`string`	none
value	`string`	none

2.5.6.1.1.72 UpsertConfigurationRequest

UpsertConfigurationRequest contains configuration settings to update

Field	Type	Description
settings	map `UpsertConfigurationRequest.SettingsEntry`	global settings to update

2.5.6.1.1.73 UpsertConfigurationRequest.SettingsEntry

Field	Type	Description
key	`string`	none
value	`string`	none

2.5.6.1.1.74 UpsertConfigurationResponse

UpsertConfigurationResponse contains the result of configuration update

Field	Type	Description
status	`Status`	Operation Status
updated_settings	map `UpsertConfigurationResponse.UpdatedSettingsEntry`	Updated settings

2.5.6.1.1.75 UpsertConfigurationResponse.UpdatedSettingsEntry

Field	Type	Description
key	`string`	none
value	`string`	none

2.5.6.1.1.76 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.77 v1/devicecontroller.proto

The device controller deals with communicating with the devices in the datacenter to control power

2.5.6.1.1.78 Services

2.5.6.1.1.79 DeviceController

DeviceController is the service interface implemented by device plugins. The topology server sends power management requests via SendDeviceRequest. The RPC returns Empty; device plugins deliver DeviceControllerResponse asynchronously via a callback to the topology server.

2.5.6.1.1.80 SendDeviceRequest

rpc SendDeviceRequest(`DeviceControllerRequest`) returns `.google.protobuf.Empty`

2.5.6.1.1.81 Messages

2.5.6.1.1.82 DeviceControllerRequest

DeviceControllerRequest is sent to device plugins for power management. This message contains the target device id as specified in the topology, and a request id. Once the request is handled by the plugin, a response with a matching requestId must be returned.

Field	Type	Description
device	string	The device ID in topology
request_id	string	Request ID is initialized by the topology server when sending the requests out to the devices. The device should respond with a DeviceControllerResponse with the same RequestId
timeout_seconds	uint64	Timeout for this device request in seconds. If not set, defaults to the controller's DeviceCallTimeout
device_state_json	bytes	Device state JSON that is returned by previous calls to this device
trace_id	string	Trace ID from the originating gRPC call (e.g. ActivateTopology). Propagated across the agent stream boundary so agent-side logs can be correlated with the server-side request that triggered the device operation.
oneof request.resource_grou	DeviceControllerRequest.AllocationRequest	none
oneof request.resource_grou	DeviceControllerRequest.AllocationRequest	none
oneof request.resource_grou	DeviceControllerRequest.AllocationRequest	none
oneof request.topology_activ	DeviceControllerRequest.PolicySetRequest	none
oneof request.topology_deac	DeviceControllerRequest.PolicySetRequest	none
oneof request.gpu_policies_u	DeviceControllerRequest.GPUPoliciesSetRequest	none
oneof request.enable_worklo	DeviceControllerRequest.PowerProfileRequest	none
oneof request.disable_workl	DeviceControllerRequest.PowerProfileRequest	none
oneof request.update_workl	DeviceControllerRequest.PowerProfileRequest	none
oneof request.ping	DeviceControllerRequest.PingRequest	none
oneof request.gpu_metrics	DeviceControllerRequest.GPUMetricsRequest	none
oneof request.std_metrics	DeviceControllerRequest.StdMetricsRequest	none
oneof request.gpu_identity	DeviceControllerRequest.GPUIdentityRequest	none
oneof request.modify_entity_	DeviceControllerRequest.EntityModificationReques	none
oneof request.bmc_health_c	DeviceControllerRequest.BMCHealthDeviceRequest	none

2.5.6.1.1.83 DeviceControllerRequest.AllocationRequest

AllocationRequest is sent for resource group activation, deactivation and update

Field	Type	Description
resource_group	`string`	Resource group name
oneof _policy.policy	optional `DeviceControllerRequest.PolicySetRequest`	The policy request. This can be null, meaning only GPU policies are being set
oneof _gpu_policies.gpu_pol	optional `DeviceControllerRequest.GPUPoliciesSetRequest`	The GPU policies request. This can be null, meaning no changes to GPU policies
oneof _workload_profiles.workloac	optional `DeviceControllerRequest.PowerProfileRequest`	The Workload Power Profiles to apply. This can be null, meaning no changes to WPPS.

2.5.6.1.1.84 DeviceControllerRequest.BMCHealthDeviceRequest

Per-device BMC pre-requisite health check request. Carries the per-node probe parameters; the server-side BMCHealthCheckTask fans this out to one request per entity. Plugins that implement device.BMCHealthSupport perform the deep probe (firmware, power write/read, telemetry, EDPp, WPPS, IB/OOB drift); plugins that do not implement it fall back to a ping + common endpoint surface check (firmware inventory, ServiceRoot, WPPS endpoint probe).

Field	Type	Description
samples_per_telemetry	`int3`	Number of telemetry samples to collect for each component on this node. Zero means use the agent-side default.
skip_writes	`bool`	When true, the agent must skip the power-limit write/read-back/MaxP/ restore cycle and only perform read-only probes.
telemetry_interval_ms	`int3`	Sleep between telemetry samples on this node, in milliseconds. Zero means no inter-sample delay (back-to-back samples).
write_resolution_time	`int3`	Per-PATCH readback window for the write-cycle probe, in milliseconds. Zero uses the agent-side default (1s).

2.5.6.1.1.85 DeviceControllerRequest.EntityModificationRequest

EntityModificationRequest is sent to device plugins as part of the two-phase entity modification cycle. When an entity is modified during operation, the plugins are notified by this message at the first phase (phase=0), before the entity modifications are committed to the in-memory topology. The plugin may request a second call after the first one. If the plugin requests a second call, this is done after the topology is updated with the new entity, and the same message is sent with phase=1

Field	Type	Description
phase	`int32`	The current phase of the operation, 0 or 1.
old_device_model	`bytes`	The previous device model
old_default_policy	`PolicyObject`	Deprecated legacy old topology-level node policy.
new_device_model	`bytes`	The new device model
new_default_policy	`PolicyObject`	Deprecated legacy new topology-level node policy.
resource_group	`string`	Resource group name, if the node is allocated for a resource group
old_default_policy_	`DeviceEffectivePolicies`	Complete effective snapshot of the old topology-level node policies.
new_default_policy	`DeviceEffectivePolicies`	Complete effective snapshot of the new topology-level node policies.

2.5.6.1.1.86 DeviceControllerRequest.GPUIdentityRequest

Read-only request for hardware identities of installed GPUs.

Field	Type	Description
gpu_ids	repeated `uint32`	none

2.5.6.1.1.87 DeviceControllerRequest.GPUMetricsRequest

Out-of-band GPU metrics request

Field	Type	Description
gpu_ids	repeated `uint32`	none

2.5.6.1.1.88 DeviceControllerRequest.GPUPoliciesSetRequest

GPUPoliciesSetRequest is sent to update GPU-specific power policies

Field	Type	Description
resource_grou	string	Resource group name
gpu_policy_a	DeviceControllerRequest.PolicyAction	Are we setting policies, or resetting them? If resetting, gpu_policies field is ignored
current_policy	PolicyObject	Deprecated legacy node policy for the node containing these GPUs.
gpu_policies	GPUPolicies	The GPU policies. This can be nil
current_policy_s	DeviceEffectivePolicies	Complete effective policy snapshot and resolved node set point.

2.5.6.1.1.89 DeviceControllerRequest.PingRequest

Out-of-band ping request

Field	Type	Description
retry_count	int32	Max number of times to retry pinging the target host

2.5.6.1.1.90 DeviceControllerRequest.PolicySetRequest

PolicySetRequest is used to set node policies during topology activation/deactivation and resource group operations

Field	Type	Description
policy_action	DeviceControllerRequest.PolicyAction	Set or unset node policy
policy	PolicyObject	Deprecated legacy policy to set.
policy_set	DeviceEffectivePolicies	Complete effective policy snapshot and resolved node set point to set.

2.5.6.1.1.91 DeviceControllerRequest.PowerProfileRequest

Field	Type	Description
enable_profile_ids	repeated int32	none
disable_profile_ids	repeated int32	none
disable_async_verification	bool	If true, profile asynchronously verification(s) is disabled

2.5.6.1.1.92 DeviceControllerRequest.StdMetricsRequest

Field	Type	Description
effective	`bool`	Read power-limit fields from BMC EnvironmentMetrics.

2.5.6.1.1.93 DeviceControllerResponse

DeviceControllerResponse is returned by the device once it finishes handling the request. It contains the device ID, the request ID, and the response payload. The response payload type matches the request payload type.

Field	Type	Description
device	`string`	The device ID as specified in the topology
request_id	`string`	The request_id must match the id of the request that initiated this response.
device_state_json	`bytes`	The device plugin state data, in JSON format, that should be stored and passed in subsequent requests If nil, the existing plugin state data should not be modified
device_ping_time	`google.protobuf.Timestamp`	The timestamp when attempt was made to access the device
device_ping_error	`string`	The error message when attempt was made to access the device and it failed
oneof response.resource_gro	`DeviceControllerResponse AllocationResponse`	none
oneof response.resource_gro	`DeviceControllerResponse AllocationResponse`	none
oneof response.resource_gro	`DeviceControllerResponse AllocationResponse`	none
oneof response.topology_act	`DeviceControllerResponse AllocationResponse`	none
oneof response.topology_dea	`DeviceControllerResponse AllocationResponse`	none
oneof response.gpu_policies_	`DeviceControllerResponse GPUPoliciesUpdateRespons`	none
oneof response.enable_workl	`DeviceControllerResponse PowerProfileResponse`	none
oneof response.disable_work	`DeviceControllerResponse PowerProfileResponse`	none
oneof response.update_work	`DeviceControllerResponse PowerProfileResponse`	none
oneof response.ping	`DeviceControllerResponse PingResponse`	none
oneof response.gpu_metrics	`DeviceControllerResponse GPUMetricsResponse`	none
oneof response.std_metrics	`DeviceControllerResponse StdMetricsResponse`	none
oneof response.gpu_identity	`DeviceControllerResponse GPUIdentityResponse`	none
oneof response.modify_entit	`DeviceControllerResponse EntityModificationRespon`	none
oneof response.bmc_health_	`DeviceControllerResponse BMCHealthDeviceResponse`	none

2.5.6.1.1.94 DeviceControllerResponse.AllocationResponse

AllocationResponse is the device response to an AllocationRequest

Field	Type	Description
success	`bool`	If not true, then error field provides the diagnostic message
error	`string`	If the call is not successful, the diagnostic message
policy	`PolicyObject`	Deprecated legacy policy applied on the node. Can be nil.
power_cap_watt	`double`	The power cap
allocated_gpu_polic	`DeviceControllerResponse.GPUPoliciesUpdateResponse`	The GPU policies applied on the node. Can be nil.
workload_profile_res	`DeviceControllerResponse.PowerProfileResponse`	The result of applying workload power profiles. Can be nil.
resource_group	`string`	The resource group name associated with this allocation
allocated_node_pol	`DeviceControllerResponse.PolicyComponentResult`	The result of applying the node policy. Can be nil.
allocated_cpu_polic	`DeviceControllerResponse.PolicyComponentResult`	The result of applying the CPU policy. Can be nil.
allocated_memory_	`DeviceControllerResponse.PolicyComponentResult`	The result of applying the memory policy. Can be nil.
policy_set	`DeviceEffectivePolicies`	Complete effective policy snapshot requested for the node. Can be nil.

2.5.6.1.1.95 DeviceControllerResponse.BMCHealthDeviceResponse

Per-device BMC pre-requisite health check response. Carries the per-node NodeReport plus any node-scoped Issues raised during the probe. Cluster- level aggregation (ClusterSummary, threshold-based Issues that span more than one node) is the BMCHealthCheckTask's job on the server side.

Field	Type	Description
success	`bool`	True if the agent was able to reach the BMC and produce at least a partial NodeReport; false on hard reachability or auth failure.
error	`string`	Diagnostic message when success is false. May be set on success too if a non-fatal probe step failed (the matching Issue is also emitted).
report	`NodeReport`	Per-node report. Always populated, even on partial failure (with the bits that did succeed).
issues	repeated `Issue`	Node-scoped Issues raised during the probe (e.g. EDPP_NO_REFERENCE, POWER_LATENCY_HIGH localized to this node).
deep_probe	`bool`	True when the agent dispatched the deep probe path (plugin implements BMCHealthSupport). False when the ping+common fallback was used.
endpoint_durations_u	repeated `int64`	Raw per-endpoint request durations in microseconds. Populated by the deep-probe path; left empty by the fallback path. The server-side BMCHealthCheckTask uses these to compute exact cross-node ClusterSummary percentiles via stats.ComputeDurations - the per-node EndpointStat already carries summary statistics, but the raw samples are needed for cross-node re-percentiling.
telemetry_durations_us	repeated `int64`	Raw per-telemetry-fetch durations in microseconds.
power_get_durati	repeated `int64`	Raw per-power-limit-GET durations in microseconds.
power_set_durati	repeated `int64`	Raw per-power-limit-SET (PATCH) durations in microseconds.
edpp_get_duratio	repeated `int64`	Raw per-EDPp-percent-GET durations in microseconds.
wpps_get_duratic	repeated `int64`	Raw per-WPPS-GET durations in microseconds.

2.5.6.1.1.96 DeviceControllerResponse.EntityModificationResponse

Field	Type	Description
response	`DeviceControllerResponse.EntityModificationResponse.Response`	none
cleanup_warning	repeated `BestEffortCleanupWarning`	none

2.5.6.1.1.97 DeviceControllerResponse.GPUIdentityResponse

Read-only hardware identity response for installed GPUs.

Field	Type	Description
identities	repeated DeviceControllerResponse.GPUIdentityResponse.GPUIdentity	none
success	bool	none
error	string	none

2.5.6.1.1.98 DeviceControllerResponse.GPUIdentityResponse.GPUIdentity

Field	Type	Description
gpu_id	uint32	none
uuid	string	none
serial_number	string	none
processor_uri	string	none
success	bool	none
error	string	none

2.5.6.1.1.99 DeviceControllerResponse.GPUMetricsResponse

Out-of-band GPU metrics response

Field	Type	Description
metrics	repeated DeviceControllerResponse.GPUMetricsResponse.GPUMetrics	Metrics for each GPU
success	bool	none
error	string	none

2.5.6.1.1.100 DeviceControllerResponse.GPUMetricsResponse.GPUMetrics

Field	Type	Description
oneof _gpu_id.gpu_id	optional uint32	GPU ID
usage	double	GPU power usage

2.5.6.1.1.101 DeviceControllerResponse.GPUPoliciesUpdateResponse

GPUPoliciesUpdateResponse is returned for GPU policies update call

Field	Type	Description
results	repeated `DeviceControllerResponse.GPUPolicyResult`	GPU-specific results
resource_grou	`string`	The resource group name associated with this Update

2.5.6.1.1.102 DeviceControllerResponse.GPUPolicyResult

GPUPolicyResult is the results of applying a power cap to a single GPU

Field	Type	Description
oneof _gpu_id.gpu_id	optional `uint32`	The GPU id within the node
ok	`bool`	Whether or not the operation was successful
power_cap_watts	`double`	The actual limit set
diag_msg	`string`	Diagnostic msg, if any

2.5.6.1.1.103 DeviceControllerResponse.GPUProfileStatus

Field	Type	Description
gpu_id	`string`	none
success	`bool`	none
enforced_workload_profile_ids	repeated `int32`	none
diag_msg	`string`	none

2.5.6.1.1.104 DeviceControllerResponse.PingResponse

PingResponse contains success, and if success is false, the error msg.

Field	Type	Description
success	`bool`	none
error	`string`	none

2.5.6.1.1.105 DeviceControllerResponse.PolicyComponentResult

PolicyComponentResult represents the result of applying a policy to a specific component

Field	Type	Description
ok	`bool`	Whether or not the operation was successful
power_cap_watts	`double`	The actual limit set in watts
diag_msg	`string`	Diagnostic message, if any

2.5.6.1.1.106 DeviceControllerResponse.PowerProfileResponse

Field	Type	Description
success	`bool`	none
error	`string`	none
gpu_status	repeated `DeviceControllerResponse.GPUProfileStatus`	none
verification_id	`string`	Optional: verification ID for async verification tracking

2.5.6.1.1.107 DeviceControllerResponse.StdMetricsResponse

StdMetricsResponse

Field	Type	Description
metrics	`NodeMetricsResponse`	none
success	`bool`	none
error	`string`	none

2.5.6.1.1.108 DeviceEffectivePolicies

DeviceEffectivePolicies carries the complete effective policy snapshot and the authoritative node power set point resolved by the topology layer.

Field	Type	Description
policies	map `DeviceEffectivePolicies.PoliciesEntry`	Effective policies keyed by their canonical policy type.
oneof _set_point_watts.set_p	optional `double`	Total node power cap to apply. When present, it must be finite and positive.

2.5.6.1.1.109 DeviceEffectivePolicies.PoliciesEntry

Field	Type	Description
key	`string`	none
value	`Policy`	none

2.5.6.1.1.110 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.111 v1/devices.proto

These are the power distribution network, node, GPU, CPU, etc. device abstractions.

2.5.6.1.1.112 Messages

2.5.6.1.1.113 CPUSpec

CPU device specification for a generic CPU. If you encounter a CPU with different power capabilities, do not use this model. Instead, create a new CPU specification for that CPU and register it as a new device.

Field	Type	Description
min_load_watts	`double`	Minimum power value that can be set (the minimum amount of power device consumes). Must be finite and greater than or equal to 0.
max_load_watts	`double`	Maximum power value that can be set (the maximum amount of power device consumes). Must be finite and greater than 0.
power_policy_plugin	`strin`	The name of the power policy plugin that will be used to set CPU specific policies
redfish_environment_metric	`strin`	Partial Redfish URL path to the CPU device that contains EnvironmentMetrics, example: "Systems/HGX_Baseboard_0/Processors/CPU_%d"
minimum_power_increment	`double`	Minimum meaningful power increment in watts. Must be finite and greater than or equal to 0. Zero (the proto3 default) means use the device default.

2.5.6.1.1.114 ComputerSystemComponent

Field	Type	Description
type	`string`	Type of the device
model	`string`	Model name of the device
count	`uint32`	Number of devices of the given type

2.5.6.1.1.115 ComputerSystemSpec

ComputerSystem device specification

Field	Type	Description
devices	repeated ComputerSystemComponent	Contained devices in this node
power_policy_plugi	string	The name of the power policy plugin that will be used to set power policies. This looks like “redfish”, (or something like “redfish_gb200.v1” if there are different flavors of it), and identifies one of the existing implementations of power policy plugins.
min_load_watts	double	Minimum power value that can be set (the minimum amount of power device consumes). Must be finite and greater than or equal to 0.
max_load_watts	double	Maximum power value that can be set (the maximum amount of power device consumes). Must be finite and greater than 0.
redfish_environment_r	string	Partial Redfish URL path to the device that contains EnvironmentMetrics, example: “Chassis/HGX_ProcessorModule_%d”
processor_modules_count	uint32	Number of processor modules ComputerSystem contains
idle_policy	map ComputerSystemSpec.IdlePolicyEntry	The policy that should be used to cap power for idle nodes. If omitted, idle nodes will not be capped. The key must be one of: Node, GPU, CPU, Memory The limit must contain either the watt limit, or percentage limit
disable_firmware_chec	bool	Skip firmware-version validation for this device.
static_load_watts	double	Total power consumed by the computer system while idle, including idle GPU and CPU consumption and fixed infrastructure overhead. A topology entity’s StaticLoad, when present, overrides this value. Must not exceed max_load_watts.
minimum_power_incren	double	Minimum meaningful power increment in watts. Must be finite and greater than or equal to 0. Zero (the proto3 default) means use the device default
metrics_source	ComputerSystemSpec.MetricsSource	none

2.5.6.1.1.116 ComputerSystemSpec.IdlePolicyEntry

Field	Type	Description
key	string	none
value	ComputerSystemSpec.Limit	none

2.5.6.1.1.117 ComputerSystemSpec.Limit

Policy limit for a component, given as watts or percentage

Field	Type	Description
oneof _watts.watts	optional `double`	Power limit in watts. Must be finite and greater than 0.
oneof _percentage.percentage	optional `double`	Power limit as a percentage. Must be finite and greater than 0 and no more than 100.

2.5.6.1.1.118 Device

A device describes the power-related attributes of a power distribution network device, gpu, node, etc.

Field	Type	Description
type	`string`	none
model	`string`	none
description	`string`	none
oneof Spec.gpu	`GPUSpec`	none
oneof Spec.computerSystem	`ComputerSystemSpec`	none
oneof Spec.powerSupply	`PDNComponentSpec`	none
oneof Spec.powerDistribution	`PDNComponentSpec`	none
oneof Spec.powerDomain	`PDNComponentSpec`	none
oneof Spec.cpu	`CPUSpec`	none
oneof Spec.memory	`MemorySpec`	none

2.5.6.1.1.119 Devices

Devices is a list of devices

Field	Type	Description
devices	repeated `Device`	none

2.5.6.1.1.120 GPUSpec

GPU device specification for a generic GPU. If you encounter a GPU with different power capabilities, do not use this model. Instead, create a new GPU specification for that GPU and register it as a new device.

Field	Type	Description
min_load_watts	`double`	Minimum power value that can be set (the minimum amount of power device consumes). Must be finite and greater than or equal to 0.
max_load_watts	`double`	Maximum power value that can be set (the maximum amount of power device consumes). Must be finite and greater than 0.
wpps_support	`bool`	Whether the GPU supports workload power profiles
power_policy_plugin	`string`	The name of the power policy plugin that will be used to set GPU specific policies
redfish_environment_met	`string`	Partial Redfish URL path to the GPU device that contains EnvironmentMetrics, example: "Systems/HGX_Baseboard_0/Processors/GPU_%d"
index_base	`uint32`	Starting Redfish index used when formatting indexed GPU endpoint paths.
edpp	`GPUSpec.EDPpSpec`	EDPp capability metadata. If omitted, EDPp is unsupported for this GPU.
minimum_power_incremen	`double`	Minimum meaningful power increment in watts. Must be finite and greater than or equal to 0. Zero (the proto3 default) means use the device default.

2.5.6.1.1.121 GPUSpec.EDPpSpec

EDPp (Electrical Design Point percentage) capability metadata for a GPU. If omitted, EDPp is unsupported for this GPU.

Field	Type	Description
min_percenta	`double`	Minimum EDPp percentage supported by this GPU. Must be set and greater than 0.
max_percenta	`double`	Maximum EDPp percentage supported by this GPU. Must be set and greater than 0.
multiplier	`double`	Multiplier used to convert between EDPp percentage and power. Must be set, finite, and greater than 0.

2.5.6.1.1.122 MemorySpec

Memory device specification for a generic memory unit.

Field	Type	Description
min_load_watts	double	Minimum power value that can be set (the minimum amount of power device consumes). Must be finite and greater than or equal to 0.
max_load_watts	double	Maximum power value that can be set (the maximum amount of power device consumes). Must be finite and greater than 0.
minimum_power_increm	double	Minimum meaningful power increment in watts. Must be finite and greater than or equal to 0. Zero (the proto3 default) means use the device default.

2.5.6.1.1.123 PDNComponentSpec

Generic power distribution network device specification

Field	Type	Description
min_load_watts	double	Minimum power value that can be set (the minimum amount of power device consumes). Must be finite and greater than or equal to 0.
max_load_watts	double	Maximum power value that can be set (the maximum amount of power device consumes). Must be finite and greater than 0.
efficiency_factor	double	A factor of efficiency of the device, affects only contained devices. Total load on the device should not exceed total available power * efficiency_factor. Must be finite and greater than 0 and no more than 1.
power_limit_wa	double	The power limit that cannot be exceeded (this_device_consumption + (downstream_devices_consumption * efficiency_factor)). Must be finite and greater than or equal to 0.
power_factor	double	AC power factor. Must be finite and between 0 and 1, inclusive.
power_limit_va	double	Apparent power limit in volt-amperes. Must be finite and greater than or equal to 0.
current_limit_amps	double	Current limit in amperes. Must be finite and greater than or equal to 0.
v_min	double	Minimum voltage. Must be finite and greater than or equal to 0.
v_max	double	Maximum voltage. Must be finite and greater than or equal to 0.
static_load_wat	double	Power consumed by this component independent of its downstream load. A topology entity's StaticLoad, when present, overrides this value. During PDN aggregation, downstream load is adjusted by efficiency_factor before this component's static load is added. Must not exceed max_load_watts.

2.5.6.1.1.124 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.125 v1/dpsagent.proto

DPS Agent service for remote device management.

The agent connects to the DPS server via a bidirectional gRPC stream, receives device control requests, and dispatches them locally via SimpleController + Redfish plugins.

Protocol versioning: nvidia.dcpower.v1 package. Additive-only changes within v1 (new oneof fields, new RPCs). Breaking changes require v2 package.

2.5.6.1.1.126 Services

2.5.6.1.1.127 AgentManagementService

AgentManagementService exposes read-only agent discovery to operators.

2.5.6.1.1.128 ListAgents

rpc ListAgents(`ListAgentsRequest`) returns `ListAgentsResponse`

ListAgents returns the current in-memory agent inventory.

2.5.6.1.1.129 DPSAgentService

DPSAgentService is exposed by the DPS server on a separate gRPC port (mTLS). The agent opens Connect(), sends Handshake as the first AgentResponse, then loops receiving AgentRequests and sending AgentResponses.

2.5.6.1.1.130 Connect

rpc Connect(`AgentResponse`) returns `AgentRequest`

2.5.6.1.1.131 Messages

2.5.6.1.1.132 AgentAlerts

AgentAlerts is the batch payload for agent-originated alert state updates.

Field	Type	Description
alerts	repeated `Alert`	none

2.5.6.1.1.133 AgentError

AgentError carries structured error information for device-level errors.

Field	Type	Description
code	`string`	Error code: “unknown_device”, “unknown_agent”, “invalid_handshake”, “queue_full”
message	`string`	none
device	`string`	Device name, if applicable.

2.5.6.1.1.134 AgentInfo

AgentInfo describes topology configuration and live sessions for one agent ID.

Field	Type	Description
agent_id	`string`	Agent identifier from topology assignments or live handshakes.
configured	`bool`	Configured is true when the active topology assigns devices to the agent.
connected	`bool`	Connected is true when the agent has at least one active stream.
assigned_devices	repeated `string`	Device instance IDs assigned to the agent by the active topology.
sessions	repeated `AgentSessionInfo`	Active connections for this agent ID.

2.5.6.1.1.135 AgentRequest

AgentRequest is the envelope for core-to-agent messages. The oneof acts as the capability router - each message type maps to a capability declared in the agent's Handshake. The agent dispatches based on the concrete type.

Field	Type	Description
traceparent	`string`	W3C Trace Context parent for the message carried by this envelope.
tracestate	`string`	W3C vendor-specific trace state associated with traceparent.
oneof msg.pong	`Pong`	Core agent lifecycle (not capability-specific)
oneof msg.remote_config	`RemoteConfig`	none
oneof msg.device_request	`DeviceControllerRequest`	device_control capability

2.5.6.1.1.136 AgentResponse

AgentResponse is the envelope for agent-to-core messages. First message must be a Handshake.

Field	Type	Description
oneof msg.handshake	`Handshake`	Core agent lifecycle
oneof msg.ping	`Ping`	none
oneof msg.device_response	`DeviceControllerResponse`	device_control capability
oneof msg.alerts	`AgentAlerts`	agent alert stream

2.5.6.1.1.137 AgentSessionInfo

AgentSessionInfo contains metadata reported by one active agent connection.

Field	Type	Description
version	`string`	Agent build version reported in the handshake.
capabilities	repeated `string`	Capabilities reported in the handshake.
reported_devices	repeated `string`	Device instance IDs reported in the handshake.

2.5.6.1.1.138 Alert

Alert is one typed alert inside an AgentAlerts payload.

Field	Type	Description
oneof payload.node_compliance	`NodeComplianceAlert`	none

2.5.6.1.1.139 CPUComplianceResource

Field	Type	Description
index	`uint32`	none

2.5.6.1.1.140 ComplianceResource

ComplianceResource identifies the node scope that the compliance status applies to.

Field	Type	Description
oneof scope.node	`NodeComplianceResource`	none
oneof scope.gpu	`GPUComplianceResource`	none
oneof scope.cpu	`CPUComplianceResource`	none
oneof scope.memory	`MemoryComplianceResource`	none

2.5.6.1.1.141 GPUComplianceResource

Field	Type	Description
index	`uint32`	none

2.5.6.1.1.142 Handshake

Handshake is the first message an agent sends after connecting. Core verifies cert CN matches agent_id when identity verification is enabled.

Field	Type	Description
agent_id	`string`	Agent identifier. Must match topology AgentId assignments.
version	`string`	Agent build version (diagnostic, not used for negotiation in v1).
devices	repeated `string`	Topology device instance IDs this agent manages (e.g. “eos0205”, “b20001”). Populated from persisted RemoteConfig received from the server. Core uses this for device request routing and mismatch detection.
capabilities	repeated `string`	Enabled capabilities, e.g., [“device_control”, “metrics”]. Diagnostic in v1; request routing is based on the concrete AgentRequest type.

2.5.6.1.1.143 ListAgentsRequest

ListAgentsRequest requests the current configured and connected agent inventory.

2.5.6.1.1.144 ListAgentsResponse

ListAgentsResponse contains one record per configured or connected agent ID.

Field	Type	Description
agents	repeated `AgentInfo`	Agents is ordered lexically by agent_id.

2.5.6.1.1.145 MemoryComplianceResource

Field	Type	Description
index	`uint32`	none

2.5.6.1.1.146 NodeComplianceAlert

NodeComplianceAlert reports whether observed node or component power is compliant with the desired DPS set point.

Field	Type	Description
device	`string`	none
resource	`ComplianceResource`	none
compliance_status	`NodeComplianceAlert.ComplianceStatus`	none
desired_set_point_w	`double`	Desired DPS target in watts.
current_reading_watt	`double`	Agent-observed power reading in watts.
observed_at	`google.protobuf.Timestamp`	Agent-observed telemetry timestamp. Unset means unavailable.
ends_at	`google.protobuf.Timestamp`	Alert expiration time. Unset means unavailable.

2.5.6.1.1.147 NodeComplianceResource

2.5.6.1.1.148 Ping

Ping is sent by the agent to keep the stream alive through ingresses.

2.5.6.1.1.149 Pong

Pong is the server's response to a Ping.

2.5.6.1.1.150 PushDeviceConfig

PushDeviceConfig carries per-device-instance agent configuration.

Field	Type	Description
instance_id	`string`	Topology device instance ID (for example, "eos0205").
telemetry_id	`string`	Telemetry stream identifier (for example, Zapp sensor UUID).
device_model_json	`bytes`	Device model JSON from topology, including Redfish endpoint and credential reference.

2.5.6.1.1.151 RemoteConfig

RemoteConfig is the device-management state pushed from server to agent. Sent after handshake (bootstrap) and whenever topology or device assignments change. The agent applies it atomically - it always has a consistent view of catalog + assignments + firmware validation policy.

Follows the OpAMP RemoteConfig pattern for topology and routing state: the server owns these inputs, and the agent reconciles to match.

Field	Type	Description
device_catalog_data	`bytes`	Device type definitions (YAML-encoded). Agent rebuilds its device registry from this. SHA-256 hash allows the agent to skip rebuilds when unchanged.
device_catalog_hash	`bytes`	none
assigned_devices	repeated `string`	Topology device instance IDs assigned to this agent. Empty means no devices assigned (e.g. fresh deploy before first topology activation).
firmware_validatio	`bool`	When true, firmware validation blocks topology activation and device operations. When false, validation only logs warnings.
device_configs	repeated `PushDeviceConfig`	Per-device instance configuration for this agent (telemetry and future fields).

2.5.6.1.1.152 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.153 v1/events.proto

2.5.6.1.1.154 Messages

2.5.6.1.1.155 DPSEvent

DPSEvent contains the event payload.

Field	Type	Description
when	`google.protobuf.Timestamp`	none
event_name	`string`	none
correlation_id	`string`	none
oneof event.ping	`google.protobuf.Empty`	none
oneof event.power_state	`PowerStateEvent`	none

2.5.6.1.1.156 EventSubscriptionRequest

Subscribes to one or more events. The subscriber id identifies the receiver, and must be unique among subscribers.

Field	Type	Description
event_nam	repeated `string`	Names of the events to subscribe. If multiple, the returned stream will have all these subscribed events. Each event name must not be empty.
sub-scriber_id	`string`	The unique id of the subscriber. Messages will be queued by subscriber. Must not be empty.
persis-tent	`bool`	If persistent, the subscription is a persistent subscription. An event queue will be kept for the subscriber and events will be queued when the subscriber is not actively connected. If a subscriber is not persistent, events will be delivered only when the subscriber is connected. Undeliv-ered events will not be queued.

2.5.6.1.1.157 PowerStateEvent

PowerStateEvent is used by nvgrid APIs to notify power state changes

Field	Type	Description
feeds	map `PowerStateEvent.FeedsEntry`	none

2.5.6.1.1.158 PowerStateEvent.FeedState

Field	Type	Description
metadata	FeedMetadata.Entry	Feed limits and constraints
targets	repeated PowerStateEvent.LoadTarget	Load targets currently active and future targets
calculated_load	LoadValue	Calculated current load
in_flight	bool	Is this load target in flight?
power_event_start_ti	google.protobuf.Timestamp	Event start time
compliant	bool	Is the data center compliant?
power_event	PowerStateEvent.EventType	none

2.5.6.1.1.159 PowerStateEvent.FeedsEntry

Field	Type	Description
key	string	none
value	PowerStateEvent.FeedState	none

2.5.6.1.1.160 PowerStateEvent.LoadTarget

LoadTarget is a copy of the nvgrid load target message, with the added 'active' field

Field	Type	Description
interval	Interval	The start_time is the the time in the future when the constraint should be activated. If omitted, the current local time is assumed. The end_time is the end time for this load target. If there are no overlapping load targets and if the end time is specified, the power will revert back to the default level at this time. If there is an overlapping target, the most recent overlapping target is used. If omitted, there is no end time.
load_constrain	LoadValue	The load constraint. If nil, it removes any constraints at the given time, which results in the system falling back to its default constraint.
load_target_st	LoadTarget-Strategy	The strategy specifies what strategy will be used to achieve the desired load_constraint.
active	bool	Whether this load target is active
correlation_id	string	Optional string that can be used by the caller to identify load targets. The correlation id can be at most 36 bytes, and can contain ascii letters/digits and "-". It must be a unique id. If not provided, a UUID will be generated.

2.5.6.1.1.161 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.162 v1/metrics.proto

The Metrics API contains metrics & telemetry data queries & messages.

2.5.6.1.1.163 Services

2.5.6.1.1.164 MetricsManagementService

MetricsManagementService is responsible for retrieving current policy state, power usage, and other telemetry data from the system.

Currently, all metrics are retrieved via Redfish. Please see the supplemental Metrics API documentation for additional information regarding challenges & limitations of this service.

For `dpsctl` usage for the MetricsManagementService, see the *dpsctl metrics CLI guide*.

2.5.6.1.1.165 GPUMetricsQuery

rpc GPUMetricsQuery(`GPUMetricsQueryRequest`) returns `GPUMetricsQueryResponse`

GPUMetricsQuery is used to query GPU power usage metrics from a list of node-GPU pairs.

2.5.6.1.1.166 MetricsQuery

rpc MetricsQuery(`MetricsQueryRequest`) returns `MetricsQueryResponse`

MetricsQuery is used to retrieve standardized GPU, CPU, and memory usage metrics from a list of requested nodes.

2.5.6.1.1.167 Messages

2.5.6.1.1.168 GPUMetricsQueryRequest

GPUMetricsQueryRequest is used to query GPU power usage metrics from a given list of node-GPU pairs.

Each pair must specify the node name & GPU ID.

If you are unsure of the GPU IDs available on the node, you may retrieve the number of available GPUs using the *`MetricsQuery`* API. The available number of GPUs is listed under the `num_gpus` property, with IDs starting at zero and ranging to `num_gpus - 1`.

Example:

```
{
    "gpus": [
        {
            "node": "viking592",
            "gpu": 0
        },
        {
            "node": "viking592",
            "gpu": 1
        },
        {
            "node": "viking592",
            "gpu": 2
        },
        {
            "node": "viking593",
            "gpu": 0
        }
    ]
}
```

Field	Type	Description
gpus	repeated `GPUMetricsQueryRequest.GpuMetricsRequest`	List of node-GPU pairs to query. Must contain at least one item.

2.5.6.1.1.169 GPUMetricsQueryRequest.GpuMetricsRequest

Field	Type	Description
node	`string`	Name of the node entity within the configured Topology. Must not be empty.
gpu	`uint32`	The GPU ID on the node. IDs typically range from 0 to 7

2.5.6.1.1.170 GPUMetricsQueryResponse

```
{
  "metrics:" [
    {
      "node": "viking592",
      "gpu": 0,
      "usage": 700.0
    },
    {
      "node": "viking592",
      "gpu": 1,
      "usage": 698.2
    },
    {
      "node": "viking592",
      "gpu": 2,
      "usage": 700.0
    },
    {
      "node": "viking593",
      "gpu": 0,
      "usage": 374.5
    },
  ]
}
```

Field	Type	Description
metrics	repeated `GPUMetricsQueryResponse.GpuMetrics`	A list of node-GPU pairs with the corresponding GPU power usage

2.5.6.1.1.171 GPUMetricsQueryResponse.GpuMetrics

Field	Type	Description
node	`string`	Name of the node entity
gpu	`uint32`	GPU ID
usage	`double`	Power usage (in watts)
oneof _gpu_uuid.gpu_uuid	optional `string`	Hardware-backed Redfish Processor UUID.

2.5.6.1.1.172 MetricsQueryRequest

MetricsQueryRequest is used to query standardized usage metrics from the requested list of nodes.

Example:

```
{
    "nodes": [
        "node001",
        "node002",
        "node003",
    ]
}
```

Field	Type	Description
nodes	repeated `string`	nodes is the list of nodes to retrieve metrics from. Each node name must not be empty.
effective	`bool`	Read power-limit fields from BMC EnvironmentMetrics.

2.5.6.1.1.173 MetricsQueryResponse

MetricsQueryResponse contains standardized usage metrics for GPU, CPU, and memory from the requested nodes.

Field	Type	Description
nodes	repeated `NodeMetricsResponse`	List of node metrics responses, one for each requested node

2.5.6.1.1.174 NodeMetricsResponse

NodeMetricsResponse contains aggregated metrics for all components on a single node

Field	Type	Description
name	`string`	Name of the node
num_gpus	`uint32`	Number of GPUs present on the node
num_cpus	`uint32`	Number of CPUs present on the node
num_memory_	`uint32`	Number of memory units (DIMMs) present on the node
power_usage	`double`	none
gpus	repeated `NodeMetricsResponse.ComponentUsage`	List of GPU usage metrics for each GPU on the node
cpus	repeated `NodeMetricsResponse.ComponentUsage`	List of CPU usage metrics for each CPU on the node
memory	repeated `NodeMetricsResponse.ComponentUsage`	List of memory usage metrics for each memory unit on the node

2.5.6.1.1.175 NodeMetricsResponse.ComponentUsage

ComponentUsage contains power and thermal metrics for a single component (GPU, CPU, or memory). Where not available, metrics will be zero.

Field	Type	Description
id	`string`	Unique identifier for the component on the node (e.g., GPU ID typically 0-based)
power_usage	`double`	Current power consumption in watts
energy_usage	`double`	Cumulative energy consumption since last reset
energy_usage_unit	`string`	Unit for energy consumption (e.g., “Joules”)
set_limit	`double`	Currently set power limit in watts
max_limit	`double`	Maximum allowed power limit in watts
min_limit	`double`	Minimum allowed power limit in watts
temperature_celsius	`double`	Current temperature in Celsius
requested_oneshot_watts	`double`	Last oneshot OOB power-limit request observed from the BMC.
requested_persistent_watts	`double`	Last persistent OOB power-limit request observed from the BMC.
oneof _power_limit_persistency.pow	optional `bool`	Persistency flag for the latest OOB power-limit request.
oneof _gpu_uuid.gpu_uuid	optional `string`	Hardware-backed Redfish Processor UUID. Set only for GPU components.

2.5.6.1.1.176 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.177 v1/nvgrid.proto

2.5.6.1.1.178 Services

2.5.6.1.1.179 NvGrid

NvGrid service provides a mechanism for bridging the gap between grid solutions that may implement OpenADR, Smart Grid, or other power utility integrations to actuate power utilization changes in datacenters operated by DPS.

2.5.6.1.1.180 SetLoadTarget

rpc SetLoadTarget(`LoadTargetRequest`) returns `LoadTargetResponse`

SetLoadTarget updates the future load target schedule using the given list of LoadTarget objects. Each LoadTarget defines the starting time, the optional ending time, and the target load. If the target load is null, the load target is reset to the data center default. If the ending time is omitted, the LoadTarget will remain in effect forever until another load target is scheduled. For a given point in time, the latest LoadTarget becomes the effective one. Because of this, updating a schedule can be achieved by setting a new schedule for the same duration with a different target, or setting the target to null to delete.

2.5.6.1.1.181 GetLoadSchedule

rpc GetLoadSchedule(`GetLoadScheduleRequest`) returns `LoadSchedule`

Returns the load schedules for some or all of the power feeds in a given interval

2.5.6.1.1.182 GetCurrentLoadTarget

rpc GetCurrentLoadTarget(`.google.protobuf.Empty`) returns `GetCurrentLoadTargetResponse`

Returns the current load target for each feed

2.5.6.1.1.183 GetMetadata

rpc GetMetadata(`.google.protobuf.Empty`) returns `FeedMetadata`

Returns metadata for each feed. This API can be used to discover all feeds defined in the data center and their default load caps.

2.5.6.1.1.184 RegisterWebhook

rpc RegisterWebhook(`RegisterWebhookRequest`) returns `LoadTargetResponse`

Registers a webhook that will receive notifications when power events happen.

2.5.6.1.1.185 UnregisterWebhook

rpc UnregisterWebhook(`UnregisterWebhookRequest`) returns `LoadTargetResponse`

Unregisters a previously registered webhook

2.5.6.1.1.186 GetStatus

rpc GetStatus(`.google.protobuf.Empty`) returns `LoadTargetStatus`

Returns the current status. This includes the current load targets and achieved load values for each power feed.

2.5.6.1.1.187 Messages

2.5.6.1.1.188 FeedMetadata

FeedMetadata object used to expose power thresholds and default constraints for all power feeds.

Field	Type	Description
metadata	map FeedMetadata.MetadataEntry	none

2.5.6.1.1.189 FeedMetadata.Entry

Metadata entry object used to expose power thresholds and default constraints for each power feed.

Field	Type	Description
feed_tag	string	none
power_minimum	LoadValue	none
power_maximum	LoadValue	none
default_constraint	LoadValue	none

2.5.6.1.1.190 FeedMetadata.MetadataEntry

Field	Type	Description
key	string	none
value	FeedMetadata.Entry	none

2.5.6.1.1.191 GetCurrentLoadTargetResponse

Response for GetCurrentLoadTarget containing current targets per feed

Field	Type	Description
load_targe	map GetCurrentLoadTargetResponse.LoadTargetsEntry	Map of feed tag to current load target for that feed

2.5.6.1.1.192 GetCurrentLoadTargetResponse.LoadTargetsEntry

Field	Type	Description
key	string	none
value	LoadTarget	none

2.5.6.1.1.193 GetLoadScheduleRequest

Specifies which load schedules to return in a given interval. The returned schedule starts with the LoadTarget that is active at the beginning of the interval and all future LoadTargets that will start before the interval's end time.

Field	Type	Description
feed_tag	repeated `string`	The power feed tags for which to return load schedule. If empty, schedule for all feed tags will be returned
interval	`Interval`	The time interval for which to return the load schedule

2.5.6.1.1.194 Interval

Interval is a time interval. A missing element in the message implies an open interval.

Field	Type	Description
start_time	`google.protobuf.Timestamp`	none
end_time	`google.protobuf.Timestamp`	none

2.5.6.1.1.195 LoadSchedule

Load Schedule is an ordered-list of scheduled power constraint changes (LoadTargets).

Field	Type	Description
targets	repeated `LoadTarget`	none

2.5.6.1.1.196 LoadTarget

Describes how and when power targets should be applied.

Field	Type	Description
interval	`Interval`	The start_time is the the time in the future when the constraint should be activated. If omitted, the current local time is assumed. The end_time is the end time for this load target. If there are no overlapping load targets and if the end time is specified, the power will revert back to the default level at this time. If there is an overlapping target, the most recent overlapping target is used. If omitted, there is no end time.
load_constrain	`LoadValue`	The load constraint. If nil, it removes any constraints at the given time, which results in the system falling back to its default constraint.
load_target_st	`LoadTarget-Strategy`	The strategy specifies what strategy will be used to achieve the desired load_constraint.
feed_tags	repeated `string`	Optional field to specify which power feeds the load target applies to. If none is given and if the datacenter has multiple power feeds, the load target will be applied to all power feeds.
correlation_id	`string`	Optional string that can be used by the caller to identify load targets. The correlation id can be at most 36 bytes, and can contain ascii letters/digits and "-". It must be a unique id. If not provided, a UUID will be generated.

2.5.6.1.1.197 LoadTargetRequest

LoadTargetRequest contains one or more load targets in the future

Field	Type	Description
targets	repeated `LoadTarget`	The load targets

2.5.6.1.1.198 LoadTargetResponse

Response object that returns a status with a diagnostic message and a list of detailed messages.

Field	Type	Description
code	`LoadTargetResponse.StatusCode`	none
diag_msg	`string`	none
details	repeated `LoadTargetResponse.Detail`	none

2.5.6.1.1.199 LoadTargetResponse.Detail

Field	Type	Description
code	`LoadTargetResponse.StatusCode`	none
details	`google.protobuf.Any`	none
correlation_id	`string`	none

2.5.6.1.1.200 LoadTargetStatus

Current status of the system

Field	Type	Description
statuses	map `LoadTargetStatus.StatusesEntry`	none

2.5.6.1.1.201 LoadTargetStatus.FeedStatus

Field	Type	Description
current_load_tar	`LoadValue`	Current power target set by a LoadTarget message
current_calculat	`LoadValue`	The current load as calculated by DPS, which may be different from current_load_target and the current actual in the data center
in_flight	`bool`	If true, DPS is still transitioning from one power target level to another
power_event_	`google.protobuf.Timestamp`	The scheduled start time of the current power event in effect
correlation_id	`string`	The correlation id of the current active load target, if any.
unmet_budget_	repeated `LoadTargetStatus.FeedStatus.UnmetBudgetReason`	Stable typed causes for a terminal unmet target. Multiple values may be present when different resources fail for different reasons. Empty for in-flight and met targets.

2.5.6.1.1.202 LoadTargetStatus.StatusesEntry

Field	Type	Description
key	`string`	none
value	`LoadTargetStatus.FeedStatus`	none

2.5.6.1.1.203 LoadTargetStrategy

Describes the strategy used to achieve the desired load constraint.

Field	Type	Description
oneof strategy.best_effort	`bool`	If power constraints require shedding load, DPS will steal power from DPM-enabled jobs until the constraint is satisfied. If the constraint cannot be satisfied, DPS will steal as much power as it can, but leave the topology in a non-compliant state until jobs terminate. If power constraints are relaxed and if there are DPM-enabled jobs from which power was stolen, DPS will attempt to give more power to these jobs

2.5.6.1.1.204 LoadValue

LoadValue specifies the load with a unit.

Field	Type	Description
value	`double`	The value of the load. Must be finite and greater than or equal to 0.
unit	`LoadValue.Unit`	The unit of the value. Must be a defined enum value.

2.5.6.1.1.205 RegisterWebhookRequest

Registers a webhook for events. This webhook will receive a Status message via an HTTP POST request.

Field	Type	Description
for_eve	repeated `RegisterWebhookRequest.PowerEvent`	The list of events to subscribe to. If unspecified, then you will be subscribed to all power events. Each event must be a defined enum value.
url	`string`	The url to register for notifications. Must not be empty.

2.5.6.1.1.206 UnregisterWebhookRequest

Unregister a webhook.

Field	Type	Description
url	`string`	The url to unregister. Must not be empty.

2.5.6.1.1.207 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.208 v1/policy.proto

Policy specifies the power policies for devices. Policies are defined within a topology with unique names describing the power min-max values.

2.5.6.1.1.209 Services

2.5.6.1.1.210 PolicyService

PolicyService manages policy bundles and their bindings.

2.5.6.1.1.211 GetBundle

rpc GetBundle(`PolicyBundleGetRequest`) returns `PolicyBundle`

GetBundle retrieves one named policy bundle.

2.5.6.1.1.212 Upsert

rpc Upsert(`PolicyUpsertRequest`) returns `Status`

Upsert creates a policy bundle or replaces its complete policy set.

2.5.6.1.1.213 Delete

rpc Delete(`PolicyBundleDeleteRequest`) returns `Status`

Delete deletes a policy bundle and all of its bindings.

2.5.6.1.1.214 CreateBinding

rpc CreateBinding(`PolicyBindingCreateRequest`) returns `PolicyBindingCreateResponse`

CreateBinding creates a user-owned policy bundle binding.

2.5.6.1.1.215 GetBinding

rpc GetBinding(`PolicyBindingGetRequest`) returns `PolicyBinding`

GetBinding retrieves one policy binding by UUID.

2.5.6.1.1.216 UpdateBinding

rpc UpdateBinding(`PolicyBindingUpdateRequest`) returns `Status`

UpdateBinding replaces the scope and level of a user-owned policy binding.

2.5.6.1.1.217 DeleteBinding

rpc DeleteBinding(`PolicyBindingDeleteRequest`) returns `Status`

DeleteBinding deletes one user-owned policy binding.

2.5.6.1.1.218 DeleteAllBindings

rpc DeleteAllBindings(`PolicyBindingsDeleteAllRequest`) returns `Status`

DeleteAllBindings deletes all user-owned bindings for a policy bundle.

2.5.6.1.1.219 ListBundles

rpc ListBundles(`PolicyBundleListRequest`) returns `PolicyBundleListResponse`

ListBundles returns one bounded page of policy bundles.

2.5.6.1.1.220 ListBindings

rpc ListBindings(`PolicyBindingListRequest`) returns `PolicyBindingListResponse`

ListBindings returns one bounded page of policy bindings.

2.5.6.1.1.221 Messages

2.5.6.1.1.222 APPMPolicy

An APPM.v1 policy type defines APPM settings.

Field	Type	Description
scope	`PolicyScope`	Defines which entities/resource groups the policy applies to.
spec	`APPMSpec`	Specifies the policy spec.

2.5.6.1.1.223 APPMSpec

APPMSpec specifies the APPM policy values

Field	Type	Description
value	`APPMSpec.Enabled`	none

2.5.6.1.1.224 AggregatePowerLimitsPolicy

An AggregatePowerLimits.v1 policy participates in per-entity effective-policy composition. Runtime validation groups complete compute nodes that carry the same composed effective policy and limits the sum of their SetPointWatts.

Field	Type	Description
scope	`PolicyScope`	Defines which entities/resource groups the policy applies to.
spec	`AggregatePowerLimitsSpec`	Specifies the policy spec.

2.5.6.1.1.225 AggregatePowerLimitsSpec

AggregatePowerLimitsSpec specifies aggregate power limits for complete compute nodes that carry the same composed effective AggregatePowerLimits policy. Runtime validation sums the SetPointWatts of each node once within the effective policy group; TargetWatts and LoadWatts are not included. A proposed allocation is rejected when any group violates its floor or ceiling. Partial-node allocation is out of scope.

Field	Type	Description
ceiling_watts	double	Max power limit for the aggregate scope. Must be finite and greater than or equal to 0.
floor_watts	double	Min power limit for the aggregate scope. Must be finite and greater than or equal to 0.

2.5.6.1.1.226 ComputerSystemLimitsPolicy

A ComputerSystemLimit.v1 policy type can be bound to any ComputerSystem type device or entity, as designated by the Scope field. Some ComputerSystem devices may be suited for a more specific policy (e.g. VeraRubinLimits.v1).

Field	Type	Description
scope	PolicyScope	Defines which entities/resource groups the policy applies to
spec	ComputerSystemLimitsSpec	Specifies the policy spec.

2.5.6.1.1.227 ComputerSystemLimitsSpec

ComputerSystemLimitsSpec specifies the computer system operating range. Node, GPU, CPU, and memory ranges are independently optional, but at least one range value must be set.

Field	Type	Description
oneof range_spec.operating_	ComputerSystemLimits OperatingRange	Node operating range in watts
oneof range_spec.operating_	ComputerSystemLimits OperatingRange	Node operating range in percentages of device max
gpu_target_percent	double	Relative GPU target, as a percentage. This value represents a % of the node ceiling. Zero means no GPU target is set. Must be finite and between 0 and 100, inclusive.
cpu_target_percent	double	Relative CPU target, as a percentage. This value represents a % of the node ceiling. Zero means no CPU target is set. Must be finite and between 0 and 100, inclusive.
memory_target_percent	double	Relative Memory target, as a percentage. This value represents a % of the node ceiling. Zero means no memory target is set. Must be finite and between 0 and 100, inclusive.
gpu_ceiling_percent	double	Relative GPU ceiling, as a percentage. This value represents a % of the node ceiling. Zero means no GPU ceiling is set. Must be finite and between 0 and 100, inclusive.
gpu_floor_percent	double	Relative GPU floor, as a percentage. This value represents a % of the node ceiling. Zero means no GPU floor is set. Must be finite and between 0 and 100, inclusive.
cpu_ceiling_percent	double	Relative CPU ceiling, as a percentage. This value represents a % of the node ceiling. Zero means no CPU ceiling is set. Must be finite and between 0 and 100, inclusive.
cpu_floor_percent	double	Relative CPU floor, as a percentage. This value represents a % of the node ceiling. Zero means no CPU floor is set. Must be finite and between 0 and 100, inclusive.
memory_ceiling_percent	double	Relative memory ceiling, as a percentage. This value represents a % of the node ceiling. Zero means no memory ceiling is set. Must be finite and between 0 and 100, inclusive.
memory_floor_percent	double	Relative memory floor, as a percentage. This value represents a % of the node ceiling. Zero means no memory floor is set. Must be finite and between 0 and 100, inclusive.
gpu_target_watts	double	GPU target in watts. Zero means no GPU target is set. Must be finite and greater than or equal to 0.
cpu_target_watts	double	CPU target in watts. Zero means no CPU target is set. Must be finite and greater than or equal to 0.
memory_target_watts	double	Memory target in watts. Zero means no memory target is set. Must be finite and greater than or equal to 0.
gpu_ceiling_watts	double	GPU ceiling in watts. Zero means no GPU ceiling

2.5.6.1.1.228 ComputerSystemLimitsSpec.OperatingRange

Field	Type	Description
ceiling	double	Operating range ceiling. Zero means no upper bound. Must be finite and greater than or equal to 0.
target	double	Operating range target. Zero means unspecified. Must be finite and greater than or equal to 0.
floor	double	Operating range floor. Zero means no lower bound. Must be finite and greater than or equal to 0.

2.5.6.1.1.229 DPMPolicy

A DPM.v1 policy type defines dynamic power management settings.

Field	Type	Description
scope	PolicyScope	Defines which entities/resource groups the policy applies to.
spec	DPMSpec	Specifies the policy spec.

2.5.6.1.1.230 DPMSpec

DPMSpec specifies dynamic power management policy settings.

Field	Type	Description
oneof _strategy.strategy	optional DPMSpec.Strategy	Strategy selects the dynamic power management optimization strategy.
oneof _power_shift.power_shift	optional DPMSpec.Shift	PowerShift selects the power-shift behavior.
oneof _grid_shift.grid_shift	optional DPMSpec.Shift	GridShift selects the grid-shift behavior.
oneof _real_time_shift.real_time_	optional DPMSpec.Shift	RealTimeShift selects the real-time-shift behavior.

2.5.6.1.1.231 EPDPolicy

An EPD.v1 policy type defines EPD settings.

Field	Type	Description
scope	PolicyScope	Defines which entities/resource groups the policy applies to.
spec	EPDSpec	Specifies the policy spec.

2.5.6.1.1.232 EPDSpec

EPDSpec specifies the EPD policy values

Field	Type	Description
value	`EPDSpec.Enabled`	none

2.5.6.1.1.233 GPUPolicies

GPUPolicies is a list of GPUPolicy objects. Each policy in the list should address a unique GPU.

Field	Type	Description
policies	repeated `GPUPolicy`	The GPU policies. Each policy should address a distinct GPU

2.5.6.1.1.234 GPUPolicy

GPUPolicy specifies the power policy for a single GPU in a node.

Field	Type	Description
oneof _gpu_id.gpu_id	optional `uint32`	The zero-based GPU index within the node. The gpu_id name is retained for API compatibility. UpdateGPUPolicies requests must provide gpu_id or gpu_uuid; when both are provided, they must identify the same GPU.
oneof _power_limit.power_	optional `PowerLimit`	The power limit. Null value means the GPU does not have a set policy. Use null to remove GPU-specific policy assignment
oneof _gpu_uuid.gpu_uuid	optional `string`	Hardware-backed Redfish Processor UUID. UpdateGPUPolicies requests may use this as the GPU selector instead of gpu_id. Other request paths do not use UUID selection. Responses include both identifiers when known.

2.5.6.1.1.235 PRSPolicy

A PRS.v1 policy type defines PRS settings.

Field	Type	Description
scope	`PolicyScope`	Defines which entities/resource groups the policy applies to.
spec	PRSSpec	Specifies the policy spec.

2.5.6.1.1.236 PRSSpec

PRSSpec specifies the PRS policy values

Field	Type	Description
value	`PRSSpec.Enabled`	none

2.5.6.1.1.237 Policy

Policy combines all known policy types

Field	Type	Description
oneof Policy.computer_system_limits	`ComputerSystemLimitsPolicy`	none
oneof Policy.dpm	`DPMPolicy`	none
oneof Policy.prs	`PRSPolicy`	none
oneof Policy.appm	`APPMPolicy`	none
oneof Policy.epd	`EPDPolicy`	none
oneof Policy.aggregate_power_limits	`AggregatePowerLimitsPolicy`	none
oneof Policy.wpps	`WPPSPolicy`	none

2.5.6.1.1.238 PolicyApplyStatus

Policy application status including applied policy details

Field	Type	Description
policy	`PolicyObject`	The policy that was applied
status	`Status`	Operation status

2.5.6.1.1.239 PolicyBinding

PolicyBinding describes one persisted policy bundle binding.

Field	Type	Description
binding_id	`string`	UUID identifier of the binding.
policy_bundle	`string`	Name of the bound policy bundle.
scope	`PolicyScope`	Scope to which the policy bundle is bound.
created_by	`PolicyBindingCreatedBy`	Subsystem that owns the binding. Only user-created bindings may be modified through PolicyService binding mutation APIs.
policy_level	`PolicyLevel`	Composition precedence for this binding. Policy level does not constrain the scope.

2.5.6.1.1.240 PolicyBindingCreateRequest

PolicyBindingCreateRequest creates a user-owned policy binding.

Field	Type	Description
policy_bundle	`string`	Unique policy bundle name.
scope	`PolicyScope`	Scope to which the policy bundle is bound.
policy_level	`PolicyLevel`	Composition precedence for the binding. Policy level does not constrain the scope.

2.5.6.1.1.241 PolicyBindingCreateResponse

PolicyBindingCreateResponse identifies the newly created binding.

Field	Type	Description
status	`Status`	Operation status.
binding_id	`string`	UUID identifier for the new binding.

2.5.6.1.1.242 PolicyBindingDeleteRequest

PolicyBindingDeleteRequest identifies a user-owned binding to delete.

Field	Type	Description
binding_id	`string`	UUID identifier of the binding.

2.5.6.1.1.243 PolicyBindingGetRequest

PolicyBindingGetRequest identifies a policy binding to retrieve.

Field	Type	Description
binding_id	`string`	UUID identifier of the binding.

2.5.6.1.1.244 PolicyBindingListRequest

PolicyBindingListRequest requests one page of policy bindings.

Field	Type	Description
page_size	`int32`	Maximum number of bindings to return. Zero uses the server default of 50; values greater than 100 are reduced to 100.
page_toke	`string`	Opaque token returned by a previous ListBindings call.

2.5.6.1.1.245 PolicyBindingListResponse

PolicyBindingListResponse contains one page of policy bindings.

Field	Type	Description
policy_bindings	repeated `PolicyBinding`	Policy bindings in ascending binding ID order.
next_page_tok	`string`	Opaque token for the next page. Empty when no more bindings remain.

2.5.6.1.1.246 PolicyBindingUpdateRequest

PolicyBindingUpdateRequest replaces a user-owned binding's scope and level.

Field	Type	Description
binding_id	`string`	UUID identifier of the binding.
scope	`PolicyScope`	Replacement binding scope.
policy_level	`PolicyLevel`	Replacement composition precedence. Policy level does not constrain the scope.

2.5.6.1.1.247 PolicyBindingsDeleteAllRequest

PolicyBindingsDeleteAllRequest removes every user-owned binding for a bundle.

Field	Type	Description
policy_bundle	`string`	Unique policy bundle name.

2.5.6.1.1.248 PolicyBundle

A PolicyBundle groups policies

Field	Type	Description
name	`string`	Optional bundle name. A bundle name is required if the PolicyBundle is used as an input from the end-user or when it is stored in to a db, but it does not need a name when the policy bundle is used simply as a set of policies.
policies	repeated `Policy`	Polymorphic policy list contained in the bundle
updated_at	`google.protobuf.Timestamp`	The timestamp policy bundle was last updated at

2.5.6.1.1.249 PolicyBundleDeleteRequest

PolicyBundleDeleteRequest identifies a policy bundle to delete.

Field	Type	Description
policy_bundle	`string`	Unique policy bundle name.

2.5.6.1.1.250 PolicyBundleGetRequest

PolicyBundleGetRequest identifies a policy bundle to retrieve.

Field	Type	Description
policy_bundle	`string`	Unique policy bundle name.

2.5.6.1.1.251 PolicyBundleListRequest

PolicyBundleListRequest requests one page of policy bundles.

Field	Type	Description
page_size	int32	Maximum number of bundles to return. Zero uses the server default of 50; values greater than 100 are reduced to 100.
page_toke	string	Opaque token returned by a previous ListBundles call.

2.5.6.1.1.252 PolicyBundleListResponse

PolicyBundleListResponse contains one page of policy bundles.

Field	Type	Description
policy_bundles	repeated PolicyBundle	Policy bundles in ascending name order.
next_page_toke	string	Opaque token for the next page. Empty when no more bundles remain.

2.5.6.1.1.253 PolicyLimit

PolicyLimit specifies a power limit for a specific entity type.

Field	Type	Description
element_type	PolicyElementType	The entity type to which the power limit applies. A valid PolicyObject must include at least one limit with POLICY_ELEMENT_TYPE_NODE (Node). Must be a defined enum value.
power_limi	PowerLimit	The power limit value

2.5.6.1.1.254 PolicyObject

PolicyObject specifies a topology or resource group entity power policy.

Field	Type	Description
name	string	Name of the policy. The name must be unique within a topology.
limits	repeated PolicyLimit	Limits for the policy. Must contain at least one item and include at least one limit with element_type NODE.
properties	google.protobuf.Struct	Properties for the policy

2.5.6.1.1.255 PolicyScope

PolicyScope selects the entities to which a policy applies. It is the sole binding selector; PolicyLevel controls precedence only. Entities are selected by:

- entity name, or other entity properties
- entity device properties
- entity membership in a resource group
- go-expr expression that selects entities

Structured resource-group scopes also participate in parent-to-child policy inheritance when used by the effective-policy resolver.

Field	Type	Description
oneof scope.entities	`PolicyScope.Entities`	none
oneof scope.devices	`PolicyScope.Devices`	none
oneof scope.resource_groups	`PolicyScope.ResourceGroups`	none
oneof scope.expr	`string`	none

2.5.6.1.1.256 PolicyScope.Device

Device message type contains the device type and model. Both fields are optional, but at least one must be present

Field	Type	Description
type	`string`	none
model	`string`	none

2.5.6.1.1.257 PolicyScope.Devices

Devices message type contains a list of entity devices

Field	Type	Description
devices	repeated `PolicyScope.Device`	Entity devices in scope. Must contain at least one item.

2.5.6.1.1.258 PolicyScope.Entities

Entities message type contains entity names

Field	Type	Description
entities	repeated `string`	Entity names in scope. Each entity name must not be empty.

2.5.6.1.1.259 PolicyScope.ResourceGroups

ResourceGroups message type contains resource groups by id

Field	Type	Description
resource_groups	repeated `string`	Resource group ids in scope. Each resource group id must not be empty.

2.5.6.1.1.260 PolicyUpsertRequest

PolicyUpsertRequest creates or replaces a named policy bundle.

Field	Type	Description
policy_bundle	`PolicyBundle`	Policy bundle to create or replace.

2.5.6.1.1.261 PowerLimit

PowerLimit specifies a power limit for a specific entity

Field	Type	Description
oneof power_limit.watts	`double`	Power limit in watts. Must be finite and greater than 0.
oneof power_limit.percentage	`double`	Power limit as a percentage. Must be finite and greater than 0 and no more than 100.

2.5.6.1.1.262 WPPSPolicy

A WPPS.v1 policy type defines workload profile settings.

Field	Type	Description
scope	`PolicyScope`	Defines which entities/resource groups the policy applies to.
spec	`WPPSSpec`	Specifies the complete desired workload profile set. Omitted and empty repeated values both mean an empty desired set and explicitly clear all workload profiles.

2.5.6.1.1.263 WPPSSpec

WPPSSpec specifies the complete desired workload profile set.

Field	Type	Description
profile_ids	repeated `int32`	Profile IDs are the complete desired set. Omitted and empty repeated values both mean an empty desired set and explicitly clear all workload profiles.

2.5.6.1.1.264 Enums

2.5.6.1.1.265 APPMSpec.Enabled

Name	Number	Description
UNSPECIFIED	0	none
DISABLED	1	none
ENABLED	2	none

2.5.6.1.1.266 DPMSpec.Shift

Shift controls an individual DPM adjustment path, such as power, grid, or real-time shifting. Disabled/enabled values express normal policy intent; force values make that intent restrictive for lower-priority policies.

Name	Number	Description
SHIFT_UNSPECIFIED	0	none
SHIFT_DISABLED	1	none
SHIFT_ENABLED	2	none
SHIFT_FORCE_DISABLED	3	none
SHIFT_FORCE_ENABLED	4	none

2.5.6.1.1.267 DPMSpec.Strategy

Strategy selects the dynamic power management optimization strategy.

Name	Number	Description
STRATEGY_UNSPECIFIED	0	none
STRATEGY_UNIFORM	1	DPS allocates power uniformly across the specified policy scope

Example: Nodes in a resource group. * Scope: Nodes in an RG * Previous Node Policies: 5600W, 4300W, 3200W, 5600W * New Node Policies (stealing 800W total power, 200W per node): 5400W, 4100W, 3000W, 5400W | | STRATEGY_PROPORTIONAL | 2 | none | | STRATEGY_GREEDY | 3 | DPS allocates power according to highest & lowest policy. Here, the highest power jobs will be stolen from first, and lowest power jobs given back to first. This strategy should be used to decrease the number of policy allocations.

Example: High power job * Scope: Topology * Previous Job Policies Job 1: 10kW, Job 2: 2kW, Job 3: 8kW * New Job Policies (stealing 1.5kW of power) Job 1: 8.5kw, Job 2: 2kW, Job 3: 8kW |

2.5.6.1.1.268 EPDSpec.Enabled

Name	Number	Description
UNSPECIFIED	0	none
DISABLED	1	none
ENABLED	2	none

2.5.6.1.1.269 PRSSpec.Enabled

Name	Number	Description
UNSPECIFIED	0	none
DISABLED	1	none
ENABLED	2	none
FORCE_DISABLED	3	none
FORCE_ENABLED	4	none

2.5.6.1.1.270 PolicyBindingCreatedBy

PolicyBindingCreatedBy identifies the subsystem that owns a binding.

Name	Number	Description
POLICY_BINDING_CREATED_BY_UNSPECIFIED	0	none
POLICY_BINDING_CREATED_BY_USER	1	none
POLICY_BINDING_CREATED_BY_TOPOLOGY	2	none
POLICY_BINDING_CREATED_BY_RESOURCE_GROUP	3	none

2.5.6.1.1.271 PolicyElementType

PolicyElementType specifies the type of entity to which the power limit applies

Name	Number	Description
POLICY_ELEMENT_TYPE_NODE	0	none
POLICY_ELEMENT_TYPE_GPU	1	none
POLICY_ELEMENT_TYPE_CPU	2	none
POLICY_ELEMENT_TYPE_MEMORY	3	none

2.5.6.1.1.272 PolicyLevel

PolicyLevel determines policy precedence during effective-policy composition. A binding's scope selects which entities receive that binding; individual policy types may further scope their own applicability. This enum does not constrain where a binding applies. Precedence increases from global through entity.

Name	Number	Description
POLICY_LEVEL_UNSPECIFIED	0	none
POLICY_LEVEL_GLOBAL	1	none
POLICY_LEVEL_TOPOLOGY	2	none
POLICY_LEVEL_RESOURCE_GROUP	3	none
POLICY_LEVEL_ENTITY	4	none

2.5.6.1.1.273 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.274 v1/resourcegroup.proto

Resource Groups manage power policies for temporary workload allocations on datacenter hardware. They provide ephemeral power management that overrides topology defaults during job execution.

A resource group is a collection of compute resources (nodes) allocated for a specific workload, such as a SLURM job or machine learning training run. Resource groups temporarily override the topology-specified power policies of their entities during workload execution, then restore topology defaults when the workload completes.

Resource groups follow a 5-step lifecycle:

1. **CREATE** - Create an empty, inactive resource group with optional default power policy
2. **ADD/REMOVE** - Add or remove compute resources (nodes); for active groups, policies are applied

or reverted immediately and power is reallocated

3. **ACTIVATE** - Apply power policies to hardware and mark the group as active
4. **UPDATE** (Optional) - Dynamically adjust power policies during workload execution
5. **DELETE** - Deactivate and cleanup, restoring topology defaults

2.5.6.1.1.275 Power Policy Hierarchy

Resource groups use a 3-level policy hierarchy to determine effective power settings:

1. **Entity-level policy** - Specific policy for individual nodes (highest priority)
2. **Resource group policy** - Default policy for all nodes in the group
3. **Topology policy** - Baseline policy from the datacenter topology (lowest priority)

2.5.6.1.1.276 Dynamic Power Management

Resource groups support dynamic power management (DPM) through:

- **Power Reservation Steering (PRS)** - Automatic power redistribution based on telemetry
- **Policy Updates** - Runtime policy adjustments for optimization
- **GPU Workload Profiles** - Hardware-specific power optimization for GPU workloads

2.5.6.1.1.277 Integration with Workload Schedulers

Resource groups are designed to integrate with workload schedulers like SLURM:

- Use external IDs to map to scheduler job IDs (e.g., SLURM_JOB_ID)
- Follow scheduler lifecycle events (job start/end)
- Support scheduler-driven power policy updates

Power policies are defined in *policy.proto*, and topology entities are defined in *topology.proto*. Workload-specific optimizations use telemetry data structures from *common.proto*.

2.5.6.1.1.278 Services

2.5.6.1.1.279 ResourceGroupManagementService

ResourceGroupManagementService manages ephemeral power policy allocations for workloads

This service provides APIs to create, manage, and monitor resource groups - collections of compute resources with customized power policies for specific workloads. Resource groups temporarily override topology-specified power policies during workload execution, enabling dynamic power management optimized for specific computational tasks.

The service integrates with workload schedulers like SLURM to provide power management throughout the workload lifecycle. It supports both static policy assignment and dynamic policy updates based on real-time telemetry data.

Resource groups must be created from an active topology. The topology defines the available entities and baseline power policies that resource groups can override.

2.5.6.1.1.280 ResourceGroupCreate

rpc ResourceGroupCreate(`ResourceGroupCreateRequest`) returns `ResourceGroupCreateResponse`

Create a new empty and inactive resource group.

This is the first step in the resource group lifecycle. The created resource group is initially empty (no resources) and inactive (no policies applied to hardware). Resources must be added and the group activated before power policies take effect.

2.5.6.1.1.281 ResourceGroupDelete

rpc ResourceGroupDelete(`ResourceGroupDeleteRequest`) returns `ResourceGroupDeleteResponse`

Deactivate and delete a given resource group

This permanently removes the resource group and restores all associated hardware resources to their topology-specified power policies. If the resource group is active, it is automatically deactivated before deletion. This is typically called when a workload completes.

2.5.6.1.1.282 ResourceGroupList

rpc ResourceGroupList(`ResourceGroupListAllRequest`) returns `ResourceGroupListAllResponse`

List all available resource groups

Returns comprehensive information about all resource groups in the system, including their activation status, assigned resources, and power policies. This is used for monitoring, administration, and troubleshooting resource group state.

2.5.6.1.1.283 ResourceGroupAddResources

rpc ResourceGroupAddResources(`ResourceGroupAddResourcesRequest`) returns `ResourceGroupAddResourcesResponse`

Add resources to a given resource group.

This assigns compute resources (nodes) to the resource group. Resources use the resource-group power policy. If the resource group is active, policies are applied immediately and power is reallocated; if inactive, the assignment is stored for activation.

2.5.6.1.1.284 ResourceGroupRemoveResources

rpc ResourceGroupRemoveResources(`ResourceGroupRemoveResourcesRequest`) returns `ResourceGroupRemoveResourcesResponse`

Remove resources from a given resource group.

This removes compute resources from the resource group. If active, policies are reverted to topology defaults and power is reallocated; if inactive, resources are simply removed.

2.5.6.1.1.285 ActivateResourceGroup

rpc ActivateResourceGroup(`ActivateResourceGroupRequest`) returns `ActivateResourceGroupResponse`

Activate a given resource group. The resource group must be inactive.

This applies the resource group's power policies to the assigned hardware resources and marks the group as active. The activation process validates power allocation constraints and applies policies to the hardware. Once active, the resource group manages power policies for its assigned resources until deactivation.

2.5.6.1.1.286 ResourceGroupUpdate

rpc ResourceGroupUpdate(`ResourceGroupUpdateRequest`) returns `ResourceGroupUpdateResponse`

Update a resource group's policy bundle or other mutable settings. If the resource group is active, the update is validated before it is persisted.

Policy bundles are scoped to the complete resource group, not individual resources.

2.5.6.1.1.287 UpdateGPUPolicies

rpc UpdateGPUPolicies(`UpdateGPUPoliciesRequest`) returns `UpdateGPUPoliciesResponse`

UpdateGPUPolicies allows updating GPU policies without specifying the resource group. Because of this, a single call may end up updating multiple resource groups. If the resource groups are active, the changes are applied immediately.

This provides direct GPU power management without requiring knowledge of resource group assignments. It's designed for telemetry-driven power optimization where external monitoring systems can adjust GPU power limits based on real-time performance data. A single call may affect multiple resource groups if GPUs span multiple groups.

2.5.6.1.1.288 StreamUpdateGPUPolicies

rpc StreamUpdateGPUPolicies(`UpdateGPUPoliciesRequest`) returns `StreamUpdateGPUPoliciesResponse`

StreamUpdateGPUPolicies streams per-node results and aggregate progress while updating GPU policies without specifying the resource group.

2.5.6.1.1.289 AsyncOperationStatus

rpc AsyncOperationStatus(`ResourceGroupAsyncOperationStatusRequest`) returns `ResourceGroupAsyncOperationStatus`

Queries the server about the status of an asynchronous operation started by one of the service APIs

Resource group operations (especially activation) can be performed asynchronously for better scalability. This API allows clients to check the progress and results of these asynchronous operations. The operation results are available until explicitly queried and forgotten.

2.5.6.1.1.290 Messages

2.5.6.1.1.291 ActivateResourceGroupRequest

ActivateResourceGroupRequest is used by ResourceGroupManagementService.ActivateResourceGroup.

Validates and applies power policies to all hardware resources in the resource group. Resource group must be inactive with resources added. After activation, workload can start.

Examples: dpsctl resource-group activate –resource-group “rg_$SLURM_JOB_ID” –sync # Synchronous activation dpsctl resource-group activate –resource-group “rg_$SLURM_JOB_ID” # Asynchronous activation dpsctl resource-group activate –resource-group “rg_$SLURM_JOB_ID” –partial-n-hosts 8 # Succeed if 8 hosts activate

Field	Type	Description
group_nam	`string`	Name of resource group to activate. Must not be empty.
strict	`bool`	If true, and if the given resource group cannot be activated because of power limits, do not try reducing the resource group policy. If false, the server will try a lower power policy if the given one fails.
async	`ResourceGroupAsyncStrategy`	Asynchronous activation strategy. If set, activation is asynchronous. If not set, activation is synchronous.
partial_activati	`ResourceGroupPartialActivation`	Options for partial activation with failure tolerance. Allows the resource group to be deemed active even if some hosts fail, as long as at least the specified number/fraction of hosts can be activated. When omitted, activation defaults to at least 80% of hosts, rounded up.
allow_reprovi	`bool`	When activating a resource and this flag is set to true, a resource group that doesn’t have sufficient power to activate will lower the power policy of other resource groups in order to be able to activate. Setting this flag to false will result in failure to activate if there isn’t sufficient power to activate the resource group.

2.5.6.1.1.292 ActivateResourceGroupResponse

ActivateResourceGroupResponse is returned if resource group activation was successful.

Field	Type	Description
node_sta	map `ActivateResourceGroupResNodeStatusesEntry`	Node statuses is a map where key is entity name and value is a struct with policy apply status and workload profile results
status	`Status`	Operation status
operation_id	`string`	The operation id for asynchronous operations, if the status is “async”

2.5.6.1.1.293 ActivateResourceGroupResponse.NodeStatusesEntry

Field	Type	Description
key	`string`	none
value	`NodeStatusResponse`	none

2.5.6.1.1.294 ComponentPolicyApplyResult

ComponentPolicyApplyResult describes the result of applying an effective power policy to a node, CPU, or memory component.

Field	Type	Description
ok	`bool`	Whether or not the operation was successful
power_cap_watts	`double`	The actual limit set in watts
diag_msg	`string`	Diagnostic message, if any

2.5.6.1.1.295 DeactivateResourceGroupRequest

DeactivateResourceGroupRequest is used to deactivate a resource group and remove applied policies from physical entities.

Removes power policies from hardware resources and returns them to topology defaults. Typically called automatically during resource group deletion. Resource group must be active. After deactivation, resources return to topology defaults.

Field	Type	Description
group_name	`strin`	Name of resource group to deactivate. Must not be empty.
wpps_disable_async_ve	`bool`	If true, workload power profile service (WPPS) operations will use synchronous verification instead of asynchronous. Default is false (asynchronous verification enabled).

2.5.6.1.1.296 DeactivateResourceGroupResponse

DeactivateResourceGroupResponse contains the resource group deactivation result.

Field	Type	Description
status	`Status`	Operation status

2.5.6.1.1.297 GPUPowerPolicyResponse

GPUPowerPolicyResponse describes the result of applying a power policy to one GPU.

Field	Type	Description
oneof _gpu_id.gpu_id	optional `uint32`	The GPU id within the node (numeric index)
ok	`bool`	Whether or not the operation was successful
set_limit	`double`	The actual power limit set, in watts (may differ from requested due to clamping)
diag_msg	`string`	Diagnostic msg, if any
oneof _gpu_uuid.gpu_uuid	optional `string`	Hardware-backed Redfish Processor UUID.

2.5.6.1.1.298 GPUPowerPolicyResponses

GPUPowerPolicyResponses wraps per-GPU power policy results with an overall status.

Field	Type	Description
gpu_power_policy_result	repeated `GPUPowerPolicyResponse`	none
status	`Status`	Operation status

2.5.6.1.1.299 GPUWorkloadProfileResponse

GPUWorkloadProfileResponse describes the result of setting workload profile on one GPU

Field	Type	Description
gpu_id	`string`	The GPU id within the node
ok	`bool`	Whether or not the operation was successful
enforced_workload_profile_ids	repeated `int32`	The actual workload profiles set
diag_msg	`string`	Diagnostic msg, if any
oneof _gpu_uuid.gpu_uuid	optional `string`	Hardware-backed Redfish Processor UUID.

2.5.6.1.1.300 GPUWorkloadProfileResponses

GPUWorkloadProfileResponses describes workload profile update results across GPUs.

Field	Type	Description
workload_profile_result	repeated GPUWorkloadProfileResponse	Operation status
status	Status	Operation status

2.5.6.1.1.301 NodeStatusResponse

NodeStatusResponse combines the node status, workload profile results, and GPU power policy results

Field	Type	Description
policy_apply_status	PolicyApplyStatus	Struct with an activation status and an actual policy
workload_profile_resu	GPUWorkloadProfileResponses	Result of the per-GPU workload profile operation
gpu_power_polic	GPUPowerPolicyResponses	Result of the per-GPU power policy operation
node_policy_resu	ComponentPolicyApplyResult	Per-component effective-policy apply results (steal / activation). Omitted when that lever was not attempted.
cpu_policy_resul	ComponentPolicyApplyResult	none
memory_policy_result	ComponentPolicyApplyResult	none

2.5.6.1.1.302 ResourceGroupAddResourcesRequest

ResourceGroupAddResourcesRequest is used by ResourceGroupManagementService.ResourceGroupAddResources.

Adds compute resources (nodes) to an existing resource group. For active resource groups, policies are applied immediately and power is reallocated. For inactive resource groups, this is a database-only operation.

Examples: dpsctl resource-group add –resource-group “rg_$SLURM_JOB_ID” –entities “node001,node002”

Field	Type	Description
group_name	`string`	Name of resource group to add resource entities to. Must not be empty.
resource_name	repeated `string`	Names of resources to add to the resource group. Must contain at least one unique item, and each entry must not be empty.
strict	`bool`	If true, and if adding resources would exceed power limits, do not try reducing policies. If false, the server may retry the configured policy bundle at its effective floor when permitted.
allow_reprovisi	`bool`	If true, and adding resources would exceed power limits, allow reprovisioning other resource groups (power stealing) to satisfy the request.

2.5.6.1.1.303 ResourceGroupAddResourcesResponse

ResourceGroupAddResourcesResponse is returned if adding resources to a resource group was successful.

Field	Type	Description
status	`Status`	Operation status

2.5.6.1.1.304 ResourceGroupAsyncOperationStatus

ResourceGroupAsyncOperationStatus returns the status of an asynchronous resource group operation. After it returns that the operation was completed, the results are forgotten.

Field	Type	Description
operation_id	`string`	Internal ID of the asynchronous operation
completed	`bool`	Completed flag
n_hosts	`uint32`	Number of hosts in the request
n_success	`uint32`	Number of requests completed successfully
n_failed	`uint32`	Number of requests failed
n_in_progress	`uint32`	Number of requests in progress
oneof status.activate	`ActivateResourceGroupResponse`	Activate resource group status object
oneof status.update	`ResourceGroupUpdateResponse`	Update resource group status object

2.5.6.1.1.305 ResourceGroupAsyncOperationStatusRequest

ResourceGroupAsyncOperationStatusRequest is used to query the status of an asynchronous operation.

Checks the progress and result of asynchronous resource group operations like activation or updates. Returns operation progress including success count, failure count, and completion status.

Field	Type	Description
oneof id.operation_id	`string`	Operation ID returned from async resource group operation
oneof id.resource_grou	`ResourceGroupAsyncOperationStatusReq ResourceGroupStatusInfo`	Resource group information

2.5.6.1.1.306 ResourceGroupAsyncOperationStatusRequest.ResourceGroupStatusInfo

Resource group information

Field	Type	Description
resource_group_name	`string`	Resource group name
operation_type	`string`	Operation type, i.e. “activate” or “update”

2.5.6.1.1.307 ResourceGroupAsyncStrategy

ResourceGroupAsyncStrategy specifies the asynchronous resource group activation strategy

Field	Type	Description
oneof Options.nHosts	`uint32`	Wait until the operation is complete for nHosts, then return. The operation continues asynchronously.
oneof Options.fracHosts	`double`	Wait until the operation is completed for the given fraction of hosts, then return. The operation continues asynchronously. Must be finite and between 0 and 1, inclusive.
oneof Options.wait	`google.protobuf.Duration`	wait this long before returning. The operation continues asynchronously. If 0, the operation returns immediately.

2.5.6.1.1.308 ResourceGroupCreateRequest

ResourceGroupCreateRequest is used by the ResourceGroupManagementService.ResourceGroupCreate.

Creates a new, empty, and inactive resource group for managing power policies during workload execution. A resource group must be identified by a unique name, such as slurm job id. When created, the resource group does not have any resources and it is not active.

Examples: dpsctl resource-group create –resource-group “rg_$SLURM_JOB_ID” –external-id “$SLURM_JOB_ID” –policy-bundle “MaxQ”

Field	Type	Description
external_id	`int64`	External ID (e.g. SLURM Job ID) - Unique identifier from external workload scheduler (e.g. SLURM_JOB_ID)
group_name	`string`	Unique resource group name - Human-readable identifier, must be unique (e.g. “rg_12345”, “job12345”). Must not be empty.
oneof _policy_name.policy_name	optional `string`	Legacy optional policy name retained for API compatibility. The policy must already exist in the topology. New clients should use policy_bundle.
workload_profile_ids	repeated `int32`	Array of requested workload profile IDs associated with the resource group
oneof _prs_enabled.prs_enabled	optional `bool`	Optional bool flag for the resource group to enable or disable prs dynamic power management default is enabled
properties	`google.protobuf.Struct`	Properties of the resource group
oneof _dpm_enable.dpm_enable	optional `bool`	Boolean flag to enable all dynamic power management, resource groups with `dpm_enable` set to false will follow strict policy management, if enough power for the selected policy is not available, activation will fail, allocated power will not be dynamically adjusted at any time during the lifetime of the resource group. default is true (i.e. enable dynamic power management)
oneof _shared_gpu_enable.share	optional `bool`	Enables shared-GPU behavior for this resource group.

When enabled, DPS allows per-GPU cap updates as long as the total GPU power across the resource group does not increase beyond the currently provisioned total for the resource group. This setting can be used together with PRS when dynamic power management is enabled. default is true; set explicitly to false to opt out | | priority | `uint32` | Priority of the resource group for power stealing and giveback ordering. Lower values indicate higher priority. Default is 0. When stealing power, lower-priority resource groups are stolen from first. When giving back power, higher-priority resource groups receive power first. Must be no greater than 2147483647. | | workload_group | `bool` | If set to true, this is a resource group for a workload, and it allocates its resources so no other workload group can allocate them. If false, this is simply group of resources, and the those resources may belong to zero or more resource groups, but at most one workload group. | | parent_group | `string` | Name of a parent group, if any. A resource group’s lifetime is bound by its parent’s lifetime. Deactivating a resource group deactivates all its children as well. The policies of a parent resource group is computed before the policies of its children, so the child resource groups can override parent resource groups policies when allowed. For instance, a parent resource group may disable DPM which will result in all child resource groups to be DPM disabled by default, but child resource groups may enable DPM for themselves. | | policy_bundle | `string` | Policy bundle configured for the complete resource group. Empty inherits global_policy_bundle; if that global setting is empty, no bundle is configured. |

2.5.6.1.1.309 ResourceGroupCreateResponse

ResourceGroupCreateResponse is returned if the resource group is created successfully.

Field	Type	Description
status	`Status`	Operation status

2.5.6.1.1.310 ResourceGroupDeleteRequest

ResourceGroupDeleteRequest is used by ResourceGroupManagementService.ResourceGroupDelete.

Deactivates and deletes a resource group, returning all hardware resources to their topology defaults. The resource group is automatically deactivated if active before deletion.

Field	Type	Description
group_name	`strin`	Name of resource group to delete. Must not be empty.
wpps_disable_async_ve	`bool`	If true, workload power profile service (WPPS) operations will use synchronous verification instead of asynchronous. Default is false (asynchronous verification enabled).

2.5.6.1.1.311 ResourceGroupDeleteResponse

ResourceGroupDeleteResponse is returned if the resource group deletion was successful.

Field	Type	Description
status	`Status`	Operation status

2.5.6.1.1.312 ResourceGroupListAllRequest

ResourceGroupListAllRequest is used to list resource groups.

Queries all resource groups in the system with optional filtering by activation status. Used for monitoring, administration, and troubleshooting resource group state. Returns comprehensive information including policies, resources, and status for each resource group.

Field	Type	Description
oneof _list_active_only.list_active_on	optional `bool`	Set to true to filter by active resource groups only

2.5.6.1.1.313 ResourceGroupListAllResponse

Response containing all (filtered) resource group information

Field	Type	Description
status	`Status`	none
resource_groups	repeated `ResourceGroupListAllResponse.ResourceGroupInfo`	List of all resource groups

2.5.6.1.1.314 ResourceGroupListAllResponse.ResourceGroupInfo

Field	Type	Description
group_name	`string`	Name of resource group
external_id	`int64`	External ID of resource group
activation_status	`string`	Activation status of the resource group
oneof _policy_name.policy_name	optional `string`	Optional default policy for resource group
resource_names	repeated `string`	Names of resources in resource group
oneof _workload_profile_ids.workload_p	optional `WorkloadProfileIDs`	Resource group workload profile ids
properties	`google.protobuf.Struct`	Properties of the resource group
oneof _prs_enabled.prs_enabled	optional `bool`	Optional bool flag for the resource group to enable or disable prs dynamic power management default is enabled
dpm_enable	`bool`	Boolean flag for enabling dynamic power management for the resource group default is true (dpm enabled)
shared_gpu_enable	`bool`	Enables shared-GPU behavior for this resource group.
oneof _priority.priority	optional `uint32`	Priority of the resource group for power stealing and giveback ordering. Marked optional so a value of 0 is always emitted over the wire instead of being dropped as the proto3 scalar default.
policy_bundle	`string`	Policy bundle configured for the complete resource group. Empty means no bundle.

2.5.6.1.1.315 ResourceGroupPartialActivation

ResourceGroupPartialActivation specifies the parameters for acceptable level of failure during host configuration

Field	Type	Description
oneof Options.atleast_n_host	`uint32`	At least this many hosts must be activated for the resource group activation to be successful
oneof Options.atleast_frac_h	`double`	At least this fraction of hosts must be activated for the resource group activation to be successful. Must be finite and between 0 and 1, inclusive.
host_activation_time	`google.protobuf.Duration`	Host activation timeout. If a host is not accessible after this duration, host is deemed inaccessible

2.5.6.1.1.316 ResourceGroupRemoveResourcesRequest

ResourceGroupRemoveResourcesRequest is used by ResourceGroupManagementService.ResourceGroupRemoveResources.

Removes compute resources (nodes) from a resource group. For active resource groups, policies are reverted to topology defaults and power is reallocated. For inactive resource groups, this is a database-only operation.

Field	Type	Description
group_name	`string`	Name of the resource group to remove resource entities from. Must not be empty.
resource_name	repeated `string`	Names of resource entities to remove from the resource group. Must contain at least one item, and each entry must not be empty.

2.5.6.1.1.317 ResourceGroupRemoveResourcesResponse

ResourceGroupRemoveResourcesResponse is returned if entity removal was successful.

Field	Type	Description
status	`Status`	Operation status

2.5.6.1.1.318 ResourceGroupUpdateRequest

ResourceGroupUpdateRequest is used by ResourceGroupManagementService.ResourceGroupUpdate to modify the resource group policy bundle and/or workload profiles.

Updates the optional policy bundle and optionally workload profiles for the resource group. Can be used on active or inactive resource groups. If resource group is active, changes are applied immediately to hardware.

Examples: dpsctl resource-group update –resource-group “rg_$SLURM_JOB_ID” –policy-bundle “bundle-name” –sync # Synchronous update

Field	Type	Description
group_name	`string`	Name of resource group to update. Must not be empty.
oneof _policy_bundle.policy_bu	optional `string`	Optional policy bundle update for the complete resource group. Omitted preserves the current bundle configuration; an explicitly empty value inherits global_policy_bundle, or resolves to no bundle when that global setting is empty.
strict	`bool`	If the resource group is already active and if strict is true, and if the given resource group cannot be activated because of power limits, do not try reducing the resource group policy. If strict is false, the server will try a lower power policy if the given one fails.
workload_profile_ids	`WorkloadProfileIDs`	Array of new requested workload profile IDs
async	`ResourceGroupAsyncStrategy`	Asynchronous activation strategy. If set, activation is asynchronous. If not set, activation is synchronous.
partial_activation	`ResourceGroupPartialActivation`	Options for partial activation with failure tolerance
wpps_disable_async	`bool`	If true, workload power profile service (WPPS) operations will use synchronous verification instead of asynchronous. Default is false (asynchronous verification enabled).
oneof _priority.priority	optional `uint32`	Optional priority update. When set, updates the resource group’s priority value. Must be no greater than 2147483647.

2.5.6.1.1.319 ResourceGroupUpdateResponse

ResourceGroupUpdateResponse is returned if resource group update was successful.

Field	Type	Description
node_sta	map `ResourceGroupUpdateResp` `NodeStatusesEntry`	Node statuses is a map where key is entity name and value is a struct with policy apply status and workload profile results
status	`Status`	Operation status
operation_id	`string`	The operation id for asynchronous operations, if the status is “async”

2.5.6.1.1.320 ResourceGroupUpdateResponse.NodeStatusesEntry

Field	Type	Description
key	`string`	none
value	`NodeStatusResponse`	none

2.5.6.1.1.321 StreamUpdateGPUPoliciesResponse

StreamUpdateGPUPoliciesResponse describes one streamed GPU policy update message.

Field	Type	Description
task_id	`string`	Internal task id for the update operation.
node_name	`string`	Node name for this streamed update message.
node_status_response	`NodeStatusResponse`	Per-node policy and GPU policy update result.
gpu_policy_update_statu	`UpdateGPUPoliciesStatus`	Aggregate operation progress after this update.

2.5.6.1.1.322 UpdateGPUPoliciesRequest

UpdateGPUPoliciesRequest is used by ResourceGroupManagementService.UpdateGPUPolicies to update GPU power policies without a reference to the resource group.

Updates individual GPU power limits based on real-time telemetry data from external monitoring systems. Used for dynamic power optimization during workload execution without knowing resource group details. All GPUs in a node must be specified or the update fails. Updates node-level policy to satisfy aggregate GPU requirements. Each GPUPolicy selects its GPU by gpu_id or gpu_uuid. When both are present, they must identify the same GPU. DPS resolves UUIDs from its in-memory hardware identity snapshot and continues to use numeric IDs for actuation.

Field	Type	Description
node_gpu_p	map `UpdateGPUPoliciesRequest.NodeGpuPoliciesEntry`	A map of node name -> GPU Policies. Must contain at least one entry.

2.5.6.1.1.323 UpdateGPUPoliciesRequest.NodeGpuPoliciesEntry

Field	Type	Description
key	`string`	none
value	`GPUPolicies`	none

2.5.6.1.1.324 UpdateGPUPoliciesResponse

UpdateGPUPoliciesResponse describes the result of each update operation.

Field	Type	Description
results	repeated `UpdateGPUPoliciesResponse.Result`	Results of the per-GPU update operation.
status	`Status`	Operation status.

2.5.6.1.1.325 UpdateGPUPoliciesResponse.Result

Result describes the result of setting the power limit of one GPU.

Field	Type	Description
resource_name	`string`	The node name containing this GPU.
oneof _gpu_id.gpu_id	optional `uint32`	The GPU id within the node.
ok	`bool`	Whether or not the operation was successful.
set_limit	`double`	The actual limit set.
diag_msg	`string`	Diagnostic msg, if any.
oneof _gpu_uuid.gpu_uuid	optional `string`	Hardware-backed Redfish Processor UUID.

2.5.6.1.1.326 UpdateGPUPoliciesStatus

UpdateGPUPoliciesStatus describes aggregate progress for a streamed GPU policy update.

Field	Type	Description
completed	`bool`	Will be true on the last message of the stream.
status	`Status`	If completed, operation status.
n_hosts	`uint32`	Number of nodes in the request.
n_success	`uint32`	Number of node requests completed successfully.
n_failed	`uint32`	Number of node requests failed.
n_in_progress	`uint32`	Number of node requests in progress.
n_gpus	`uint32`	Number of GPU policy entries in the request.
n_gpus_success	`uint32`	Number of GPU policy entries completed successfully.
n_gpus_failed	`uint32`	Number of GPU policy entries failed.
n_gpus_in_progress	`uint32`	Number of GPU policy entries in progress.

2.5.6.1.1.327 WorkloadProfileIDs

WorkloadProfileIDs is a wrapper around an array of workload profile IDs

Field	Type	Description
ids	repeated `int32`	none

2.5.6.1.1.328 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.329 v1/task.proto

Task service provides operations for monitoring async tasks

2.5.6.1.1.330 Services

2.5.6.1.1.331 TaskService

TaskService provides operations for monitoring async tasks

2.5.6.1.1.332 ListTasks

rpc ListTasks(`.google.protobuf.Empty`) returns `TaskInfo`

ListTasks returns all ongoing async tasks for monitoring and debugging

2.5.6.1.1.333 Messages

2.5.6.1.1.334 TaskInfo

TaskInfo describes an ongoing async task

Field	Type	Description
id	`string`	Unique task identifier
name	`string`	Task name
status	`string`	Current task status
started_at	`google.protobuf.Timestamp`	When the task started
completed_at	`google.protobuf.Timestamp`	When the task completed (if completed)
parent_id	`string`	Parent task ID (if this is a subtask)
diag_message	`string`	Diagnostic message when the task failed (e.g., panic details)

2.5.6.1.1.335 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.336 v1/topology.proto

Topology defines the entities in a data center (PDUs, PSUs, nodes, etc.) and how they are connected.

A topology can be created in one of two ways:

- (Preferred method) Import a topology JSON file. This process creates the data center and the entities in the data center.
- Create a data center, and a topology. Then use the functions of the topology service to add entities and entity relationships.

A newly created topology is not active. You have to explicitly “activate” it using the topology service. Activation validates the topology connectivity and policies before activating the topology and applying policies.

Topology entities (Entity, TopologyEntity) are defined in this file.

Policy definitions are in *policy.proto*

Power policies associated with topology entities can be overridden by resource groups, defined in *resourcegroup.proto*

2.5.6.1.1.337 Services

2.5.6.1.1.338 TopologyManagementService

TopologyManagementService manages the entities and topologies in a data center.

2.5.6.1.1.339 ValidateTopology

rpc ValidateTopology(`ValidateTopologyRequest`) returns `ValidateTopologyResponse`

Validates a topology

The purpose of this API is to aid the topology editing process. It is designed to provide user-friendly feedback without altering the database.

The topology is uploaded as a stream of ValidateTopologyRequest objects. Once all the entities and topology entities are uploaded, the client closes the stream to start validation. The validation takes into account existing entities and combines them with the provided entities. It returns a detailed validation report if it encounters any errors.

This call does not modify any existing entities or topologies in the database.

2.5.6.1.1.340 CreateEntities

rpc CreateEntities(`Entity`) returns `.google.protobuf.Empty`

Creates new datacenter entities. These entities are not part of any topology. The provided entities must be all new entities.

2.5.6.1.1.341 UpsertEntities

rpc UpsertEntities(`Entity`) returns `.google.protobuf.Empty`

Retained for API compatibility. Entity upsert is not supported; the server returns UNIMPLEMENTED without modifying entity or topology state.

2.5.6.1.1.342 GetEntities

rpc GetEntities(`GetEntitiesRequest`) returns `Entity`

Returns all entities or selected entities

2.5.6.1.1.343 DeleteEntities

rpc DeleteEntities(`DeleteEntitiesRequest`) returns `DeleteEntitiesResponse`

Deletes given entities. The call will either fail without any change, or will successfully delete all given entities. If an entity is part of a topology, deletion will fail.

2.5.6.1.1.344 CreateUpdateTopology

rpc CreateUpdateTopology(`CreateUpdateTopologyRequest`) returns `CreateUpdateTopologyResponse`

Creates or updates a topology.

A topology describes a particular power connection layout using the existing entities in a datacenter. A topology may define entity specific default policies and/or entity properties.

Before creating a topology, all the entities and power policies referred in that policy must be created.

When the topology is created, it is inactive. Activating a topology will activate default policies on topology entities and provide power allocation services using the constraints and connections defined in the topology.

The topology is uploaded as a stream of CreateUpdateTopologyRequest objects. The first object in the stream must specify whether this is a create or update operation, and provide the necessary flags for the chosen operation. Each element of the stream provides zero or more topology entities. These entities must refer to already existing datacenter entities.

After all the topology is uploaded, the client closes the stream to start topology validation and actual database modifications. Any failure at this stage will be returned in the response.

2.5.6.1.1.345 TopologyDelete

rpc TopologyDelete(`TopologyDeleteRequest`) returns `TopologyDeleteResponse`

Deletes an inactive topology

2.5.6.1.1.346 GetTopologyModel

rpc GetTopologyModel(`GetTopologyModelRequest`) returns `GetTopologyModelResponse`

GetTopologyModel streams the entities and relationships in a topology

2.5.6.1.1.347 ActivateTopology

rpc ActivateTopology(`ActivateTopologyRequest`) returns `ActivateTopologyResponse`

Activate a given topology.

This call first validates topology connectivity. It then validates the power policies, and then applies those power policies to the topology entities and marks the topology as active.

If another topology is already active, this call can be used to switch to a new topology. This can be useful when sections of the datacenter are enabled/disabled.

2.5.6.1.1.348 DeactivateTopology

rpc DeactivateTopology(`DeactivateTopologyRequest`) returns `DeactivateTopologyResponse`

Deactivate a given topology

Deactivating an active topology will remove existing power policies.

2.5.6.1.1.349 ListTopologies

rpc ListTopologies(`TopologyListRequest`) returns `TopologyListResponse`

List existing topologies

2.5.6.1.1.350 NodeConnectionCheck

rpc NodeConnectionCheck(`TopologyNodeConnectionCheckRequest`) returns `TopologyNodeConnectionCheckResponse`

Perform a node connection check

Performing a node connection check verifies that a connection can be established between DPS and the topology-specified nodes for policy configuration.

2.5.6.1.1.351 UpsertDevices

rpc UpsertDevices(`Devices`) returns `.google.protobuf.Empty`

Insert or update a device definitions in the db.

Each datacenter entity refers to a device definition that specifies the power characteristics of that device.

2.5.6.1.1.352 ListDevices

rpc ListDevices(`.google.protobuf.Empty`) returns `Device`

List all known devices in the database

2.5.6.1.1.353 GetNodesStatus

rpc GetNodesStatus(`GetNodesStatusRequest`) returns `GetNodesStatusResponse`

Get last known status on specified nodes

2.5.6.1.1.354 ValidateAllocation

rpc ValidateAllocation(`ValidateAllocationRequest`) returns `ValidateAllocationResponse`

Validate allocation (dry-run allocation feasibility analysis without persisting state)

Validates whether devices can be allocated without creating a resource group or modifying hardware. Returns power impact analysis, policy selection prediction, and per-device allocation results. Streams response: first message contains overall validation result, subsequent messages contain per-device details. Use this for pre-flight checks and “what-if” allocation feasibility analysis before resource group activation.

2.5.6.1.1.355 GenerateEntities

rpc GenerateEntities(`GenerateEntitiesRequest`) returns `GenerateEntitiesResponse`

GenerateEntities validates auth and config and asynchronously queries the BCM or Nautobot for datacenter entities. Returns a request_id for streaming results.

2.5.6.1.1.356 GetGenerateEntitiesStatus

rpc GetGenerateEntitiesStatus(`GetGenerateEntitiesStatusRequest`) returns `Entity`

GetGenerateEntitiesStatus streams the generated entities for the given request_id. Use the request_id returned by GenerateEntities. Streams Entity messages, not status.

2.5.6.1.1.357 CancelGenerateEntitiesRequest

rpc CancelGenerateEntitiesRequest(`CancelGenerateEntitiesRequestRequest`) returns `CancelGenerateEntitiesRequestResponse`

CancelGenerateEntitiesRequest cancels an async generate entities request.

2.5.6.1.1.358 DeleteGenerateEntitiesRequest

rpc DeleteGenerateEntitiesRequest(`DeleteGenerateEntitiesRequestRequest`) returns `DeleteGenerateEntitiesRequestResponse`

DeleteGenerateEntitiesRequest deletes a generate entities request and any stored results.

2.5.6.1.1.359 SubscribeEvents

rpc SubscribeEvents(`EventSubscriptionRequest`) returns `DPSEvent`

SubscribeEvents allows a third-party application to subscribe to topology events

2.5.6.1.1.360 Messages

2.5.6.1.1.361 ActivateTopologyRequest

ActivateTopologyRequest is used by TopologyManagementService.ActivateTopology.

Activating an inactive topology involves validating connectivity and policy consistency. After topology is validated, the power policies for those entities that have a default power policy assigned will be activated. If an entity does not have a default power policy assigned, then the entity will be capped at its idle power policy if one exists. Otherwise, the entity will be “ping”ed if ping_hosts is set, and the result of the ping will be included in the return status. If ping_hosts is not set, entities with no power policies will not be pinged and they may be inaccessible when the topology starts.

If all hosts are to be pinged, it is also possible to specify a failure tolerance for activation by setting atleast_frac_hosts, which defaults to 0.5. That means, if half of the entities are accessible (either via a ping or by setting entity default policy), then topology activation is deemed successful.

If there is already an active topology when another topology is activated, the second activation request fails. Topology switching is currently disabled; deactivate the active topology before activating another.

Field	Type	Description
topol-ogy_name	strin(	Name of topology to activate. Must not be empty.
switch_if_activ(	bool	Retained for API compatibility. Topology switching is currently disabled, so setting this field while another topology is active returns InvalidOperation. Deactivate first.
ping_hosts	bool	Ping all hosts
atleast_frac_hc	dou-ble	At least this fraction of hosts must be accessible to activate the topology. Must be finite and between 0 and 1, inclusive.

2.5.6.1.1.362 ActivateTopologyResponse

ActivateTopologyResponse will be returned if topology activation was successful

Field	Type	Description
node_sta	map ActivateTopologyResponse. NodeStatusesEntry	Status a map where key is entity name and value is a struct with activation status and actual policy
status	Status	Operation status

2.5.6.1.1.363 ActivateTopologyResponse.NodeStatusesEntry

Field	Type	Description
key	string	none
value	PolicyApplyStatus	none

2.5.6.1.1.364 CancelGenerateEntitiesRequestRequest

CancelGenerateEntitiesRequestRequest is used by TopologyManagementSer-vice.CancelGenerateEntitiesRequest. It contains the request id returned by GenerateEntities.

Field	Type	Description
request_id	string	none

2.5.6.1.1.365 CancelGenerateEntitiesRequestResponse

CancelGenerateEntitiesRequestResponse contains the cancellation status.

Field	Type	Description
status	`Status`	none

2.5.6.1.1.366 CreateUpdateTopologyRequest

CreateUpdateTopologyRequest is used to stream the elements of a topology to the server. The same stream element is used for both creation of a new topology, or update of an existing one. Updating an existing topology requires sending the complete topology (i.e. it is not an incremental update).

The first message in the stream specifies whether it is for creating a new topology, or updating it using the Operation field. The subsequent messages should omit this field. The topology name and topology properties must be given in the first message.

A topology update operation implements an optimistic locking scheme to prevent unintended overwrites. The GetTopologyModel API returns a topology hash computed by alphabetically sorting JSON serialization of the topology. The CreateUpdateTopologyRequest must include this hash, meaning that this update request is for the update of the topology with that hash, and if the topology stored in the database has a different hash (that is, if the topology in the database was modified after GetTopologyModel was called), then the update is rejected. It is expected that the client will refetch the topology and reapply the changes. Setting the force_update field disables this behavior and overwrites the topology without first checking for modifications.

Each entity and policy referenced in CreateUpdateTopologyRequest.entities must already be defined in the database.

The actual topology validation and creation/update happens after the client sends all topology entities and closes the stream.

Field	Type	Description
oneof Operation.create_topology	`CreateUpdateTopologyRequest.CreateRequest`	Create a new topology
oneof Operation.update_topology	`CreateUpdateTopologyRequest.UpdateRequest`	Update an existing topology
entities	repeated `TopologyEntity`	Zero or more topology entities.

2.5.6.1.1.367 CreateUpdateTopologyRequest.CreateRequest

CreateRequest describes the topology to create.

The created topology will be inactive.

Field	Type	Description
topology_name	`string`	Name of topology to create. The topology name must be unique. Must not be empty.
properties	`google.protobuf.Struct`	Optional topology properties.
policy_bundle	`string`	Policy bundle configured for the complete topology. Empty inherits global_policy_bundle; if that global setting is empty, no bundle is configured.

2.5.6.1.1.368 CreateUpdateTopologyRequest.UpdateRequest

UpdateRequest describes the topology to update.

Field	Type	Description
topology_name	`string`	Name of the existing topology to update. Must not be empty.
new_topology_name	`string`	If topology is to be renamed, the new name of the topology. If the topology name is not to be changed, this field should be left empty.
topology_hash	`bytes`	The hash of the topology to update. The update will fail if the topology hash doesn't match what's in the db. The hash is the SHA256 hash of the unindented topology JSON constructed by recursively ordering JSON objects within the topology model.
properties	`google.protobuf.Struct`	Topology properties
force_update	`bool`	Force update. Update the topology even if the topology was modified by someone else
oneof _policy_bundle.policy_bun	optional `string`	Optional policy bundle update for the complete topology. Omitted preserves the current bundle configuration; an explicitly empty value inherits global_policy_bundle, or resolves to no bundle when that global setting is empty.

2.5.6.1.1.369 CreateUpdateTopologyResponse

CreateUpdateTopologyResponse will be returned if adding relations to the topology was successful

Field	Type	Description
status	`Status`	none

2.5.6.1.1.370 DeactivateTopologyRequest

DeactivateTopologyRequest is used by TopologyManagementService.DeactivateTopology.

Deactivating the topology removes the policies applied to the policy entities and marks the topology as inactive. All power caps will be removed.

Field	Type	Description
topology_name	`string`	Name of topology to deactivate. Must not be empty.

2.5.6.1.1.371 DeactivateTopologyResponse

DeactivateTopologyResponse will be returned if topology deactivation was successful

Field	Type	Description
status	`Status`	Operation status

2.5.6.1.1.372 DeleteEntitiesRequest

DeleteEntitiesRequest contains a list of entities to delete

Field	Type	Description
entities	repeated `string`	The list of entities to delete. Must contain at least one item, and each entry must not be empty. If there are any references to the entity from any topology, the deletion will fail.

2.5.6.1.1.373 DeleteEntitiesResponse

DeleteEntitiesResponse contains the status of the delete operation

Field	Type	Description
status	`Status`	Status of the operation
entity_topolo	map `DeleteEntitiesResponse.EntityTopologiesEntry`	If entities cannot be deleted because they are used in topologies, map of entity -> []topology

2.5.6.1.1.374 DeleteEntitiesResponse.EntityTopologiesEntry

Field	Type	Description
key	`string`	none
value	`DeleteEntitiesResponse.Topologies`	none

2.5.6.1.1.375 DeleteEntitiesResponse.Topologies

Field	Type	Description
topologies	repeated `string`	none

2.5.6.1.1.376 DeleteGenerateEntitiesRequestRequest

DeleteGenerateEntitiesRequestRequest is used by TopologyManagementService.DeleteGenerateEntitiesRequest. It contains the request id returned by GenerateEntities.

Field	Type	Description
request_id	`string`	none

2.5.6.1.1.377 DeleteGenerateEntitiesRequestResponse

DeleteGenerateEntitiesRequestResponse contains the delete status.

Field	Type	Description
status	`Status`	none

2.5.6.1.1.378 Entity

Entity represents a topology entity matching the Entity JSON schema specification.

Field	Type	Description
entity	`google.protobuf.Struct`	none

2.5.6.1.1.379 GenerateEntitiesFromBCMRequest

GenerateEntitiesFromBCMRequest contains connection parameters for querying BCM to retrieve datacenter entities. Used as the source in GenerateEntitiesRequest. Results are streamed via GetGenerateEntitiesStatus using the request_id from GenerateEntitiesResponse.

Field	Type	Description
username	`string`	Username for BCM authentication. Must not be empty.
password	`string`	Password for BCM authentication

2.5.6.1.1.380 GenerateEntitiesFromNautobotRequest

GenerateEntitiesFromNautobotRequest is used by GenerateEntitiesRequest. It contains filters to query and generate specific entities from Nautobot. Note: Nautobot generation is reserved and may be unimplemented on the server.

Field	Type	Description
locations	repeated `string`	none
racks	repeated `string`	none
device_types	repeated `string`	none
oneof _request_string.request_strir	optional `string`	Optional free-form query string passed to the Nautobot API for additional filtering.
transport	`Nautobot-Transport`	Nautobot HTTP API for this import. Must be a defined enum value.

2.5.6.1.1.381 GenerateEntitiesRequest

GenerateEntitiesRequest is used by TopologyManagementService.GenerateEntities. It contains either Nautobot or BCM request messages to use as a source to query and generate entities from either.

Field	Type	Description
oneof source.bcm	`GenerateEntities-FromBCMRequest`	none
oneof source.nautob	`GenerateEntities-FromNautobotRequest`	none
timeout	`google.protobuf.Duration`	Optional timeout for the generation operation. If unset or zero, the server uses its default (10 minutes). Clamped to a server-defined maximum (30 minutes).

2.5.6.1.1.382 GenerateEntitiesResponse

GenerateEntitiesResponse is used by TopologyManagementService.GenerateEntities. It contains the validation status and a request id for async retrieval.

Field	Type	Description
status	`Status`	The validation status. The status will be ok only if there are no validation errors
request_id	`string`	The actual import will be async and need to return an id for which the response can be checked

2.5.6.1.1.383 GetEntitiesRequest

GetEntitiesRequest specifies the entities to fetch

Field	Type	Description
names	repeated `string`	If nonempty, entities matching these names will be returned. If empty, all entities will be returned. Each name must not be empty.

2.5.6.1.1.384 GetGenerateEntitiesStatusRequest

GetGenerateEntitiesStatusRequest is used by TopologyManagementService.GetGenerateEntitiesStatus. It contains the request id returned by GenerateEntities.

Field	Type	Description
request_id	`string`	Request id returned by GenerateEntities. Must not be empty.

2.5.6.1.1.385 GetNodesStatusRequest

GetNodesStatusRequest contains a list of nodes to retrieve status

Field	Type	Description
nodes	repeated `string`	List of nodes to query. Each node name must not be empty.

2.5.6.1.1.386 GetNodesStatusResponse

GetNodesStatusResponse contains the last known node status

Field	Type	Description
node_statuse	map `GetNodesStatusResponse.NodeStatusesEntry`	Map of node statuses by entity name

2.5.6.1.1.387 GetNodesStatusResponse.NodeState

Contains node state

Field	Type	Description
resource_group	`string`	Resource group name
allocated_policy_name	`string`	Allocated policy name

2.5.6.1.1.388 GetNodesStatusResponse.NodeStatus

node status contains the latest node status

Field	Type	Description
inaccessible_since	`google.protobuf.Timestamp`	Inaccessible since timestamp
last_successful_acce	`google.protobuf.Timestamp`	Last successful access timestamp
last_access_error	`string`	Last error message
current_state	`GetNodesStatusResponse.NodeState`	Node state
last_policy_error	`string`	Last policy error (e.g., idle policy application failure)
cleanup_warnings	repeated `BestEffortCleanupWarning`	Best-effort cleanup warnings recorded during topology/entity changes

2.5.6.1.1.389 GetNodesStatusResponse.NodeStatusesEntry

Field	Type	Description
key	`string`	none
value	`GetNodesStatusResponse.NodeStatus`	none

2.5.6.1.1.390 GetTopologyModelRequest

GetTopologyModelRequest specifies the topology name whose model will be returned

Field	Type	Description
topology_name	`string`	Name of the topology model to retrieve. Must not be empty.

2.5.6.1.1.391 GetTopologyModelResponse

GetTopologyModelResponse is a stream element containing topology entities and relations. The first message in the stream will contain the header field giving topology hash and topology properties. The topology hash is a hash computed from the alphabetically sorted JSON representation of the topology, and should be used when updating the returned topology to prevent unintentional overwrites.

Field	Type	Description
heade	`GetTopologyModelRespo Header`	The header is sent once as the first message in the stream. The header should be ignored in subsequent messages.
enti- ties	repeated `TopologyEn- tity`	One or more topology connections

2.5.6.1.1.392 GetTopologyModelResponse.Header

Field	Type	Description
topol- ogy_name	`string`	Name of the topology
proper- ties	`google. protobuf. Struct`	Topology properties
topol- ogy_hash	`bytes`	SHA256 hash of the topology. This value should be sent in update topology request to make sure the topology was not changed since the caller retrieved the topology
pol- icy_bundle	`string`	Policy bundle configured for the complete topology. Empty means no bundle.

2.5.6.1.1.393 TopologyDeleteRequest

TopologyDeleteRequest is used to delete a topology

Field	Type	Description
topology_name	`string`	Name of the topology to delete. Must not be empty.

2.5.6.1.1.394 TopologyDeleteResponse

TopologyDeleteResponse has the topology deletion response

Field	Type	Description
status	`Status`	none

2.5.6.1.1.395 TopologyEntity

TopologyEntity defines a reference to an entity, and the relation between topology entities

Field	Type	Description
entity	`google.protobuf.Struct`	The entity reference. The struct elements match the definition of TopologyEntity in the topology schema, so it must include the Name field that identifies the datacenter entity already existing in the database. The topology entity may include other properties, and these properties will be merged with the properties of the existing entity. This allows the topology entity to override matching values of an existing entity.
children	repeated `string`	The child entity ids of this entity

2.5.6.1.1.396 TopologyListRequest

TopologyListRequest is used by TopologyManagementService.ListTopologies.

Field	Type	Description
oneof _is_active.is_active	optional `bool`	Filter by active status. If non-nil, only active or inactive topologies will be returned. If nil, all topologies will be returned regardless of activation status.

2.5.6.1.1.397 TopologyListResponse

TopologyListResponse contains the topology list specified by TopologyListRequest.

Field	Type	Description
status	`Status`	Operation status
topologies	repeated `TopologyListResponse.TopologyInfo`	List of all Topologies matching the TopologyListRequest specification

2.5.6.1.1.398 TopologyListResponse.TopologyInfo

Fields pertaining to topologies returned from listing

Field	Type	Description
topology_name	`string`	Name of Topology
is_active	`bool`	Is Topology active/inactive
leaf_node_name	repeated `string`	Names of configurable leaf entities in Topology
policy_bundle	`string`	Policy bundle configured for the complete topology. Empty means no bundle.

2.5.6.1.1.399 TopologyNodeConnectionCheckRequest

TopologyNodeConnectionCheckRequest is used by TopologyManagementService.NodeConnectionCheck.

Field	Type	Description
topology_name	`string`	Name of topology to perform check. Note: no buf.validate annotation here because empty topology_name is valid. When empty, the server pings all entities regardless of topology (used for health checks and pinging nodes without an active topology).
nodes	repeated `string`	Optional names of nodes to check. If none specified, all nodes will be checked.
oneof _attempts.attempts	optional `int32`	Optional number of connection attempts (default 1). Must be at least 1 when set.

2.5.6.1.1.400 TopologyNodeConnectionCheckResponse

TopologyNodeConnectionCheckResponse contains a message about node connection health.

Field	Type	Description
total_nodes	`int64`	Total number of nodes in the topology
success_node	`int64`	Number of successful node checks
failure_nodes	repeated `TopologyNodeConnectionCheckResponse.TopologyNodeConnectionCheckFailure`	Failed nodes with error message

2.5.6.1.1.401 TopologyNodeConnectionCheckResponse.TopologyNodeConnectionCheckFailure

Fields pertaining to a connection check failure on a node

Field	Type	Description
node_name	`string`	Name of the failed node
error_msg	`string`	Error message

2.5.6.1.1.402 ValidateTopologyRequest

ValidateTopologyRequest is used to stream the elements of a topology to the server for validation. No part of the topology will be stored. Each element of the stream contains zero or more entities and topology entities. The first message must set `header`; subsequent messages must omit it.

The topology validation happens after all requests are sent and the stream is closed by the client.

The purpose of this request is to break a topology definition into gRPC-friendly messages instead of uploading a large topology file that may exceed gRPC message length limits.

Field	Type	Description
oneof _header.head	optional `ValidateTopologyReques` `Header`	none
entities	repeated `Entity`	Zero or more entities that would replace existing datacenter entities in the database, or new entities that would be defined if the topology is saved. Topology validation takes into account existing database entities merged with the entities specified in ValidateTopologyRequests.
topology_entities	repeated `TopologyEntity`	Zero or more topology entities. Each topology entity must refer to an existing entity already stored in the database or specified in one of the ValidateTopologyRequest.entities fields. Since the actual validation takes place after all topology is uploaded via the stream, topology entities of an individual request may refer to entities that will be defined in later instances of ValidateTopologyRequests

2.5.6.1.1.403 ValidateTopologyRequest.Header

Field	Type	Description
topology_name	`string`	none

2.5.6.1.1.404 ValidateTopologyResponse

ValidateTopologyResponse contains topology validation results

Field	Type	Description
status	`Status`	The validation status. The status will be ok only if there are no validation errors
validation_errors	repeated `ValidationError`	Zero or more validation errors.

2.5.6.1.1.405 Enums

2.5.6.1.1.406 NautobotTransport

NautobotTransport selects the Nautobot HTTP API for one generate-entities import. GraphQL is the zero value so clients that omit the field use GraphQL.

Name	Number	Description
NAUTOBOT_TRANSPORT_GRAPHQL	0	none
NAUTOBOT_TRANSPORT_REST	1	none

2.5.6.1.1.407 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.408 v1/verification.proto

Verification service provides endpoints to verify DPS deployment status

2.5.6.1.1.409 Services

2.5.6.1.1.410 VerificationService

VerificationService provides methods to verify DPS deployment status

2.5.6.1.1.411 Verify

rpc Verify(`VerificationRequest`) returns `VerificationResponse`

Verify performs comprehensive verification of DPS components

2.5.6.1.1.412 StartBMCHealthCheck

rpc StartBMCHealthCheck(`BMCHealthCheckRequest`) returns `BMCHealthCheckHandle`

StartBMCHealthCheck submits a deep BMC pre-requisite health check across the cluster and returns immediately with a task handle. The check itself can run for many minutes (large clusters with deep probe enabled), so the RPC is intentionally fire-and-forget to avoid holding the gRPC stream open for the entire run. Callers poll GetBMCHealthCheckStatus and GetBMCHealthCheckReport using BMCHealthCheckHandle.task_id to observe progress and retrieve the final report.

2.5.6.1.1.413 GetBMCHealthCheckStatus

rpc GetBMCHealthCheckStatus(`BMCHealthCheckStatusRequest`) returns `BMCHealthCheckStatus`

GetBMCHealthCheckStatus returns the live status (and coarse progress) of a previously started BMC health check.

2.5.6.1.1.414 GetBMCHealthCheckReport

rpc GetBMCHealthCheckReport(`BMCHealthCheckReportRequest`) returns `BMCHealthCheckReportResponse`

GetBMCHealthCheckReport fetches the final cluster-level report. Returns done=false with no payload while the underlying task is still running; callers should poll until done=true.

2.5.6.1.1.415 Messages

2.5.6.1.1.416 ComponentStatus

ComponentStatus represents the status of an individual component

Field	Type	Description
name	`string`	Component name
healthy	`bool`	Whether the component is healthy
message	`string`	Status message
oneof _details.details	optional `string`	Additional details if check was performed
skipped	`bool`	Whether this check was skipped
oneof _skip_reason.skip_reason	optional `string`	Reason for skipping the check

2.5.6.1.1.417 VerificationRequest

VerificationRequest is the request for DPS verification

Field	Type	Description
check_dps_server	`bool`	Perform DPS server connectivity checks
check_database	`bool`	Perform database connectivity checks
check_auth	`bool`	Perform authentication service checks
check_ui	`bool`	Perform UI accessibility checks
check_bcm	`bool`	Perform BCM connectivity checks (if BCM is configured)

2.5.6.1.1.418 VerificationResponse

VerificationResponse contains the results of DPS verification

Field	Type	Description
status	`Status`	Overall verification status
dps_server	`ComponentStatus`	Individual component check results
database	`ComponentStatus`	none
auth	`ComponentStatus`	none
ui	`ComponentStatus`	none
oneof _bcm.bcm	optional `ComponentStatus`	none

2.5.6.1.1.419 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.1.1.420 v1/version.proto

Version service provides version information about dps-server

2.5.6.1.1.421 Services

2.5.6.1.1.422 VersionService

VersionService provides methods to get version information about the dps-server

2.5.6.1.1.423 GetVersion

rpc GetVersion(`VersionRequest`) returns `VersionInfo`

GetVersion returns version information about the server

2.5.6.1.1.424 Messages

2.5.6.1.1.425 VersionInfo

VersionInfo contains version details about the dps-server

Field	Type	Description
version	`string`	Application version
commit	`string`	Git commit hash
date	`string`	Build date

2.5.6.1.1.426 VersionRequest

VersionRequest is an empty request for getting version information

2.5.6.1.1.427 Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	boolean	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteStr	str

2.5.6.2 Topology Schema Reference

DPS uses JSON schemas to validate topology definitions and ensure data consistency. These schemas are based on the DMTF Redfish specification with DPS-specific extensions.

These pages are generated from the schema files in the `dcpower` repository. Do not edit them by hand; edit `scripts/generate-schema-pages.sh` instead.

2.5.6.2.1 Available Schemas

- *DPS topology*

2.5.6.2.2 Usage

These schemas are used for:

1. **Validation** - Both `dpsctl` and `dps-server` validate topology files against these schemas
2. **Development** - Use as reference when creating topology definitions
3. **Tooling** - IDE integration for JSON validation and autocompletion

2.5.6.2.3 Schema Validation

To validate your topology files against these schemas:

```
# Using dpsctl (recommended)
dpsctl topology validate topology.json

# Using an external JSON schema validator
npm install -g ajv-cli
ajv validate -s schemas/topology/topology.schema.json -d topology.json
```

2.5.6.2.4 Further Reading

- *Topologies* - Understanding topology structure
- *Managing Topologies* - Practical topology management
- DMTF Redfish Specification - Official Redfish documentation

2.5.6.2.4.1 DPS topology

2.5.6.2.4.2 Download

`topology.schema.json`

2.5.6.2.4.3 Schema Definition

DPS topology

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "DPS topology",
    "type": "object",
    "$id": "dcpower_0.1",
    "$ref": "#/definitions/Main",
    "definitions": {
        "Main": {
            "type": "object",
            "properties": {
                "Entities": {
                    "$ref": "#/definitions/Entities"
                },
                "Topology": {
                    "$ref": "#/definitions/Topology"
                },
```

(continues on next page)

(continued from previous page)

```
                "Policies": {
                    "$ref": "#/definitions/Policies"
                }
            }
        },
        "Entities": {
            "description": "List of data center entities",
            "type": "array",
            "items": {
                "$ref": "#/definitions/Entity"
            },
            "uniqueItems": true
        },
        "Topology": {
            "description": "Root topology container",
            "type": "object",
            "properties": {
                "Name": {
                    "description": "Name of the topology being defined in
↪this file",
                    "type": "string",
                    "minLength": 1
                },
                "Entities": {
                    "description": "List of data center relations (entities
↪connectivity)",
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/TopologyEntity"
                    },
                    "uniqueItems": true
                }
            },
            "required": [
                "Name"
            ]
        },
        "TopologyEntity": {
            "description": "A reference to an entity, optional properties of
↪the entity within the topology, and the connected entities",
            "oneOf": [
                {
                    "type": "object",
                    "properties": {
                        "Name": {
                            "type": "string",
                            "description": "Reference to an already defined
↪entity"
                        },
                        "FeedTag": {
                            "type": "string",
                            "description": "Identifier for the power feed
```

(continues on next page)

(continued from previous page)

```
↪supplying this entity"
                        },
                        "AgentId": {
                            "type": "string",
                            "description": "Agent identifier for remote
↪device management. Inherited by children."
                        },
                        "Children": {
                            "$ref": "#/definitions/TopologyLinks"
                        }
                    },
                    "required": ["Name"]
                },
                {
                    "allOf": [
                        {
                            "$ref": "#/definitions/Entity"
                        },
                        {
                            "type": "object",
                            "properties": {
                                "FeedTag": {
                                    "type": "string",
                                    "description": "Identifier for the power
↪feed supplying this entity"
                                },
                                "AgentId": {
                                    "type": "string",
                                    "description": "Agent identifier for
↪remote device management. Inherited by children."
                                },
                                "Children": {
                                    "$ref": "#/definitions/TopologyLinks"
                                }
                            }
                        }
                    ]
                }
            ]
        },
        "TopologyLinks": {
            "description": "List of child entity names for a given parent",
            "type": "array",
            "items": {
                "description": "Child entity name",
                "type": "string",
                "minLength": 1
            },
            "minItems": 1
        },
        "Entity": {
            "description": "A datacenter entity",
```

(continues on next page)

(continued from previous page)

```
            "oneOf": [
                {
                    "$ref": "#/definitions/PowerDomain"
                },
                {
                    "$ref": "#/definitions/PowerDistribution"
                },
                {
                    "$ref": "#/definitions/PowerSupply"
                },
                {
                    "$ref": "#/definitions/ComputerSystem"
                }
            ]
        },
        "PowerValue": {
            "type": "object",
            "description": "Represents a specific non-negative power value",
            "properties": {
                "Value": {
                    "description": "Numerical power value",
                    "type": "number",
                    "exclusiveMinimum": 0
                },
                "Type": {
                    "description": "Power value unit type",
                    "type": "string",
                    "enum": [
                        "W",
                        "kW"
                    ]
                }
            },
            "required": [
                "Value",
                "Type"
            ],
            "additionalProperties": false
        },
        "Policies": {
            "description": "List of power policies that can be applied to
↪entities",
            "type": "array",
            "items": {
                "$ref": "#/definitions/Policy"
            },
            "uniqueItems": true
        },
        "Policy": {
            "description": "Power policy definition",
            "type": "object",
            "required": [ "Name", "Limits" ],
```

(continues on next page)

(continued from previous page)

```
            "properties": {
                "Name": {
                    "description": "Name of the power policy",
                    "type": "string"
                },
                "Limits": {
                    "description": "List of power limits that can be applied
↪to entities",
                    "type": "array",
                    "items": {
                        "type": "object",
                        "required": [ "ElementType", "PowerLimit" ],
                        "properties": {
                            "ElementType": {
                                "type": "string",
                                "enum": [ "Node", "GPU", "CPU", "Memory" ],
                                "description": "Type of entity to which the
↪power limit applies"
                            },
                            "PowerLimit": {
                                "description": "Power limit value",
                                "$ref": "#/definitions/PowerLimit"
                            }
                        }
                    },
                    "contains": {
                        "type": "object",
                        "required": [ "ElementType" ],
                        "properties": {
                            "ElementType": {
                                "const": "Node"
                            }
                        }
                    },
                    "uniqueItems": true,
                    "minItems": 1
                },
                "additionalProperties": false
            }
        },
        "PowerLimit": {
            "type": "object",
            "description": "Represents a specific non-negative power limit
↪value",
            "properties": {
                "Watts": {
                    "type": "number",
                    "description": "Power limit in watts",
                    "exclusiveMinimum": 0
                },
                "Percentage": {
                    "type": "number",
```

(continues on next page)

(continued from previous page)

```
                    "description": "Power limit as a percentage of the total
↪power capacity",
                    "exclusiveMinimum": 0,
                    "maximum": 100
                }
            },
            "oneOf": [
                {
                    "required": [ "Watts" ]
                },
                {
                    "required": [ "Percentage" ]
                }
            ],
            "additionalProperties": false
        },
        "PowerDomain": {
            "description": "DPS power domain definition",
            "type": "object",
            "properties": {
                "Type": {
                    "description": "DPS type disambiguator",
                    "type": "string",
                    "enum": [
                        "PowerDomain"
                    ]
                },
                "Name": {
                    "description": "Unique name of the entity",
                    "type": "string"
                },
                "OperatingLimit": {
                    "description": "Power domain operating limit definition
↪(base operating cap)",
                    "type": "object",
                    "properties": {
                        "PowerValue": {
                            "description": "Power domain operating limit value
↪",
                            "$ref": "#/definitions/PowerValue"
                        },
                        "PowerFactor": {
                            "description": "Power modifier to express loss
↪during transition through this component",
                            "type": "number"
                        }
                    },
                    "required": [
                        "PowerValue"
                    ]
                },
                "OverProvisioningPercentage": {
```

(continues on next page)

(continued from previous page)

```
                    "description": "Percentage by which the provisioning
↪limit exceeds the operating limit. Provisioning limit = OperatingLimit * (1
↪+ pct/100)",
                    "type": "number",
                    "minimum": 0
                },
                "Redfish": {
                    "description": "Reference to underlying Redfish definition
↪",
                    "$ref": "#/definitions/Redfish"
                },
                "Properties": {
                    "description": "Free-form entity metadata",
                    "type": "object",
                    "additionalProperties": true
                },
                "TelemetryId": {
                    "type": "string",
                    "description": "Identifier for binding this entity to
↪telemetry data from the provider"
                }
            },
            "required": [
                "Type",
                "Name"
            ],
            "dependencies": {
                "OverProvisioningPercentage": ["OperatingLimit"]
            }
        },
        "PowerDistribution": {
            "description": "DPS power distribution definition",
            "type": "object",
            "properties": {
                "Type": {
                    "description": "DPS type disambiguator",
                    "type": "string",
                    "enum": [
                        "PowerDistribution"
                    ]
                },
                "Name": {
                    "description": "Unique name of the entity",
                    "type": "string"
                },
                "Model": {
                    "description": "Code model name that represents this
↪entity",
                    "type": "string",
                    "minLength": 1
                },
                "OperatingLimit": {
```

(continues on next page)

(continued from previous page)

```
                    "description": "Power distribution operating limit
↪definition (base operating cap)",
                    "type": "object",
                    "properties": {
                        "PowerValue": {
                            "description": "Power distribution operating
↪limit value",
                            "$ref": "#/definitions/PowerValue"
                        },
                        "PowerFactor": {
                            "description": "Power modifier to express loss
↪during transition through this component",
                            "type": "number"
                        }
                    },
                    "required": [
                        "PowerValue"
                    ]
                },
                "OverProvisioningPercentage": {
                    "description": "Percentage by which the provisioning
↪limit exceeds the operating limit. Provisioning limit = OperatingLimit * (1
↪+ pct/100)",
                    "type": "number",
                    "minimum": 0
                },
                "Redfish": {
                    "description": "Reference to underlying Redfish definition
↪",
                    "$ref": "#/definitions/Redfish"
                },
                "TelemetryId": {
                    "type": "string",
                    "description": "Identifier for binding this entity to
↪telemetry data from the provider"
                }
            },
            "required": [
                "Type",
                "Name"
            ],
            "dependencies": {
                "OverProvisioningPercentage": ["OperatingLimit"]
            }
        },
        "PowerSupply": {
            "description": "DPS power supply definition",
            "type": "object",
            "properties": {
                "Type": {
                    "description": "DPS type disambiguator",
                    "type": "string",
```

(continues on next page)

(continued from previous page)

```
                    "enum": [
                        "PowerSupply"
                    ]
                },
                "Name": {
                    "description": "Unique name of the entity",
                    "type": "string"
                },
                "Model": {
                    "description": "Code model name that represents this
↪entity",
                    "type": "string",
                    "minLength": 1
                },
                "OperatingLimit": {
                    "description": "Power supply operating limit definition
↪(base operating cap)",
                    "type": "object",
                    "properties": {
                        "PowerValue": {
                            "description": "Power supply operating limit value
↪",
                            "$ref": "#/definitions/PowerValue"
                        },
                        "PowerFactor": {
                            "description": "Power modifier to express loss
↪during transition through this component",
                            "type": "number"
                        }
                    },
                    "required": [
                        "PowerValue"
                    ]
                },
                "OverProvisioningPercentage": {
                    "description": "Percentage by which the provisioning
↪limit exceeds the operating limit. Provisioning limit = OperatingLimit * (1
↪+ pct/100)",
                    "type": "number",
                    "minimum": 0
                },
                "Redfish": {
                    "description": "Reference to underlying Redfish definition
↪",
                    "$ref": "#/definitions/Redfish"
                },
                "TelemetryId": {
                    "type": "string",
                    "description": "Identifier for binding this entity to
↪telemetry data from the provider"
                }
            },
```

(continues on next page)

(continued from previous page)

```
            "required": [
                "Type",
                "Name"
            ],
            "dependencies": {
                "OverProvisioningPercentage": ["OperatingLimit"]
            }
        },
        "ComputerSystem": {
            "description": "DPS computer system definition",
            "type": "object",
            "properties": {
                "Type": {
                    "description": "DPS type disambiguator",
                    "type": "string",
                    "enum": [
                        "ComputerSystem"
                    ]
                },
                "Name": {
                    "description": "Unique name of the entity",
                    "type": "string"
                },
                "Model": {
                    "description": "Code model name that represents this
↪entity",
                    "type": "string",
                    "minLength": 1
                },
                "Policy": {
                    "description": "Default policy applied to this entity",
                    "type": "string"
                },
                "Redfish": {
                    "description": "Reference to underlying Redfish definition
↪",
                    "$ref": "#/definitions/Redfish"
                },
                "Properties": {
                    "description": "Free-form entity metadata. Recognized
↪keys include SerialNumber, MacAddress, NicoMachineID, and the import
↪provenance pair ImportSource (e.g. \"bcm\" or \"nautobot\") and
↪ImportExternalID (source-system device ID), which must be set together.",
                    "type": "object",
                    "properties": {
                        "ImportSource": {
                            "type": "string",
                            "minLength": 1
                        },
                        "ImportExternalID": {
                            "type": "string",
                            "minLength": 1
```

(continues on next page)

(continued from previous page)

```
                        },
                        "MacAddress": {
                            "type": "string"
                        },
                        "SerialNumber": {
                            "type": "string"
                        },
                        "NicoMachineID": {
                            "description": "NiCO machine ID",
                            "type": "string"
                        }
                    },
                    "dependencies": {
                        "ImportSource": ["ImportExternalID"],
                        "ImportExternalID": ["ImportSource"]
                    },
                    "additionalProperties": true
                },
                "TelemetryId": {
                    "type": "string",
                    "description": "Identifier for binding this entity to
↪telemetry data from the provider"
                }
            },
            "required": [
                "Type",
                "Name"
            ]
        },
        "Redfish": {
            "type": "object",
            "description": "Redfish endpoint configuration",
            "properties": {
                "URL": {
                    "type": "string",
                    "description": "Redfish endpoint URL"
                },
                "SecretName": {
                    "type": "string",
                    "description": "Name of the secret containing Redfish
↪credentials"
                },
                "AdditionalHeaders": {
                    "type": "array",
                    "description": "Per-entity HTTP headers to inject on
↪Redfish requests. Override deployment-level defaults; overridden by
↪deployment-level override set.",
                    "items": {
                        "type": "object",
                        "properties": {
                            "Name": {
                                "type": "string",
```

(continues on next page)

(continued from previous page)

```
                                "description": "HTTP header name"
                            },
                            "EntityFieldValue": {
                                "type": "string",
                                "description": "Direct dotted path into the
↪entity object or constrained template with {{ dotted.path }} placeholders.
↪The context is the decoded entity JSON (e.g. Properties.SerialNumber or
↪host={{ Redfish.TargetIP }})."
                            }
                        },
                        "required": ["Name", "EntityFieldValue"],
                        "additionalProperties": false
                    }
                }
            },
            "additionalProperties": true
        }
    }
}
```

2.5.6.3 dpsctl

2.5.6.3.1 Overview

`dpsctl` is a CLI used to query or send control commands to the Dynamic Power Software.

2.5.6.3.1.1 Usage

```
dpsctl [global options] [command [command options]]
```

2.5.6.3.1.2 Flags

The following global options configure the endpoint, authentication, and output:

Global dpsctl Options

```
   --host string, -H string                          DPS gRPC host address
↪(default: "localhost") [$DPSCTL_HOST]
   --port uint, -p uint                              DPS gRPC port number
↪(default: 50051) [$DPSCTL_PORT]
   --grpc-timeout duration, -D duration              gRPC timeout (duration)
↪(default: 1m0s) [$DPSCTL_GRPC_TIMEOUT]
   --resolve string                                  Provide a custom address
↪for a hostname in format 'host:port:addr' (similar to curl --resolve)
   --credentials-path string, -c string              The path to the
↪credentials configuration file (default: "${HOME}/.dpsctl/credentials.yaml
↪") [$DPSCTL_CREDENTIALS_PATH]
   --oidc-issuer string                              OIDC issuer URL for
↪browser login endpoint discovery [$DPSCTL_OIDC_ISSUER]
   --oidc-token-url string                           OIDC token endpoint URL
```

(continues on next page)

(continued from previous page)

```
↪override [$DPSCTL_OIDC_TOKEN_URL]
   --oidc-device-auth-url string                    OIDC browser login
↪authorization endpoint URL override [$DPSCTL_OIDC_DEVICE_AUTH_URL]
   --oidc-client-id string, --client-id string      OIDC OAuth client ID [
↪$DPSCTL_OIDC_CLIENT_ID, $DPSCTL_CLIENT_ID]
   --oidc-ca-cert-file string                       Path to PEM CA bundle for
↪OIDC endpoint verification [$DPSCTL_OIDC_CA_CERT_FILE]
   --oidc-allow-http                                Allow HTTP OIDC endpoint
↪URLs for local development only (default: false) [$DPSCTL_OIDC_ALLOW_HTTP]
   --oidc-oauth-scope string, --oauth-scope string  OIDC OAuth scopes to
↪request for browser login or client credentials [$DPSCTL_OIDC_OAUTH_SCOPE,
↪$DPSCTL_OAUTH_SCOPE]
   --insecure                                       Disable TLS encryption
↪for gRPC connections (default: false) [$DPSCTL_INSECURE]
   --insecure-tls-skip-verify                       Disable TLS certificate
↪validation for gRPC connections (default: false) [$DPSCTL_INSECURE_TLS_SKIP_
↪VERIFY]
   --ca-cert-path string                            Path to CA certificate
↪file for TLS verification [$DPSCTL_CA_CERT_PATH]
   --client-cert-path string                        Path to client
↪certificate file for mTLS authentication [$DPSCTL_CLIENT_CERT_PATH]
   --client-key-path string                         Path to client private
↪key file for mTLS authentication [$DPSCTL_CLIENT_KEY_PATH]
   --auth-mode string                               Authentication mode.
↪Valid values are: auto, jwt, mtls (default: "auto") [$DPSCTL_AUTH_MODE]
   --output string, -o string                       Output format. Valid
↪values are: json, yaml (default: "json") [$DPSCTL_OUTPUT]
   --no-retry                                       Whether to avoid retrying
↪a retryable command (default: false) [$DPSCTL_NO_RETRY]
   --no-version-check                               Skip the preflight dpsctl/
↪dps-server version mismatch check (default: false) [$DPSCTL_NO_VERSION_
↪CHECK]
   --max-retries int                                The number of retries to
↪attempt on a retryable command (default: 3) [$DPSCTL_MAX_RETRIES]
   --retry-delay int                                The time to wait between
↪retries, in seconds, for a retryable command (default: 3) [$DPSCTL_RETRY_
↪DELAY]
   --log-file string                                Path to log file
↪(default: "${HOME}/.dpsctl/dpsctl.log") [$DPSCTL_LOG_FILE]
   --debug                                          Enable debug logging
↪(default: false) [$DEBUG]
   --help, -h                                       show help
   --version, -v                                    print the version
```

For the server settings that these options must match, refer to *Configure Server Authentication*.

2.5.6.3.1.3 OIDC Browser Login

Use `--oidc-issuer` with `login` to discover the identity provider's device authorization and token endpoints. Provide `--oidc-client-id` and `--oidc-oauth-scope` when your deployment requires specific values. The browser login defaults are client ID `dpsctl` and scope `dps:api` when no client ID or scopes are supplied through flags or saved client settings.

The interactive CLI uses a public OAuth client without a client secret. It sends no secret during device authorization, token exchange, or token refresh. The UI uses a separate public client. Register these clients through *Configure OIDC Mode*.

You can override the discovered endpoints with `--oidc-device-auth-url` and `--oidc-token-url`. Without issuer discovery, provide both endpoint URLs. Use `--oidc-ca-cert-file` when the provider's certificate requires a private certificate authority (CA).

Refer to *dpsctl login* for the sign-in procedure and token selection. Automation uses the separate client credentials flow described in the following section.

2.5.6.3.1.4 OIDC Client Credentials

In OIDC mode, `dpsctl` can obtain a service account access token for unattended applications using the OAuth 2.0 client credentials grant. Register a separate confidential client with a client ID and secret for this flow. The following global flags configure that flow; each has an environment-variable equivalent.

Flag	Environment variable	Description
`--oidc-token-ur`	DPSCTL_OIDC_TOKEN_UR	OIDC token endpoint URL for the client-credentials grant.
`--oidc-client-i`	DPSCTL_OIDC_CLIENT_I	OAuth client ID.
`--oidc-client-s`	DPSCTL_OIDC_CLIENT_S	Path to a file containing the client secret. Prefer this option to an inline secret.
`--oidc-client-s`	DPSCTL_OIDC_CLIENT_S	Inline client secret. Use the secret-file option to keep the value out of process listings and shell history.
`--oidc-oauth-sc`	DPSCTL_OIDC_OAUTH_SC	OAuth scope to request. Repeatable. Defaults to `dps:api` when unset.
`--oidc-ca-cert-`	DPSCTL_OIDC_CA_CERT_	PEM CA bundle used to verify the token endpoint's TLS certificate (for a private or enterprise CA).
`--oidc-allow-ht`	DPSCTL_OIDC_ALLOW_HT	Allow an `http://` token endpoint. Local development only.

The following details apply to client credentials:

- The token endpoint must use HTTPS unless `--oidc-allow-http` is set.
- `--oidc-client-secret` and `--oidc-client-secret-file` are hidden from `dpsctl --help`. Prefer `--oidc-client-secret-file` or the DPSCTL_OIDC_CLIENT_SECRET_FILE environment variable.
- The aliases `--client-id`, `--client-secret`, `--client-secret-file`, and `--oauth-scope` remain available, with their corresponding DPSCTL_CLIENT_ID, DPSCTL_CLIENT_SECRET, DPSCTL_CLIENT_SECRET_FILE, and DPSCTL_OAUTH_SCOPE environment variables.
- Acquired tokens are cached in the credentials file and reused while valid when the token URL, client ID, and scopes match.
- If DPS is configured with an OIDC `allowedClientIds` allowlist, the client ID must be listed there; otherwise the request is rejected as `invalid_authorized_party`.

- Client credentials apply to normal commands. `dpsctl login` uses LDAP username/password authentication or OIDC browser login and does not use client credentials.

For service account examples, refer to *Obtain a Service Account Token*.

2.5.6.3.1.5 Commands

- *`server-version`* - Get DPS server version information
- *`login`* - Log in to the DPS server
- *`topology`* - Topology operations
- *`resource-group`* - Resource Group operations
- *`gpu-policy`* - Set GPU power policies
- *`device`* - Manage devices
- *`check`* - DPS diagnostic tools
- *`policy`* - Policy operations
- *`settings`* - Settings operations
- *`import`* - Import operations
- *`verify`* - Verify DPS deployment status
- *`validate`* - Validation operations (dry-run analysis without persisting state)
- *`auth`* - Authentication operations
- *`nvgrid`* - NvGrid power management operations
- *`task`* - Task operations
- *`agent`* - Agent operations
- `help`, `h` - Shows a list of commands or help for one command

2.5.6.3.1.6 Aliases

Command	Aliases
`topology`	`tp`
`resource-group`	`rg`
`validate`	`val`
`help`	`h`

2.5.6.3.1.7 dpsctl device

2.5.6.3.1.8 `dpsctl device` Usage Guide

2.5.6.3.1.9 Overview

`dpsctl device` manages devices in DPS.

2.5.6.3.1.10 Usage

```
dpsctl device [command [command options]]
```

2.5.6.3.1.11 Flags

Includes *global dpsctl options*.

```
   --help, -h  show help
```

2.5.6.3.1.12 Commands

- *upsert* - Update or insert device specs from YAML file(s)
- *list* - List devices
- help, h - Shows a list of commands or help for one command

2.5.6.3.1.13 list

2.5.6.3.1.14 dpsctl device list Usage Guide

List device specifications

2.5.6.3.1.15 Usage

```
dpsctl device list
```

2.5.6.3.1.16 Flags

Includes *global dpsctl options*.

```
   --help, -h          show help
```

2.5.6.3.1.17 Examples

2.5.6.3.1.18 Basic Usage

```
$ dpsctl device list
[
  {
    "type": "PowerDomain",
    "description": "Generic power domain",
    "Spec": {
      "PowerDomain": {
        "efficiency_factor": 0.95
      }
    }
  },
  {
    "type": "PowerDistribution",
```

(continues on next page)

(continued from previous page)

```
    "model": "RackPDU95_57500W",
    "Spec": {
      "PowerDistribution": {
        "max_load_watts": 57500,
        "efficiency_factor": 0.95
      }
    }
  },
  {
    "type": "PowerSupply",
    "model": "PowerSupply95_3300W",
    "Spec": {
      "PowerSupply": {
        "max_load_watts": 3300,
        "efficiency_factor": 0.95
      }
    }
  }
]
```

2.5.6.3.1.19 upsert

2.5.6.3.1.20 dpsctl device upsert Usage Guide

Insert or update device specifications. If there are no devices with the given type and model, that device will be inserted. If a device exists with the given type and model, the device will be updated.

2.5.6.3.1.21 Usage

```
dpsctl device upsert <file> [file ...]
```

2.5.6.3.1.22 Flags

Includes *global dpsctl options*.

```
   --help, -h          show help
```

2.5.6.3.1.23 Examples

2.5.6.3.1.24 Basic Usage

```
$ dpsctl device upsert devices.yaml
{
  "status": {
    "diag_msg": "Success",
    "ok": true
  }
}
```

2.5.6.3.1.25 Upsert Multiple Files

```
$ dpsctl device upsert gpu-devices.yaml system-devices.yaml
{
  "status": {
    "diag_msg": "Success",
    "ok": true
  }
}
```

2.5.6.3.1.26 Notes

- At least one YAML device specification file is required.

2.5.6.3.1.27 dpsctl server-version

2.5.6.3.1.28 dpsctl server-version Usage Guide

Get DPS server version information.

2.5.6.3.1.29 Usage

```
dpsctl server-version
```

2.5.6.3.1.30 Flags

Includes *global dpsctl options*.

```
   --help, -h          show help
```

2.5.6.3.1.31 Examples

2.5.6.3.1.32 Basic Usage

```
$ dpsctl server-version
{
  "version": "v0.1.0",
  "build_date": "2024-01-15T10:30:00Z",
  "git_commit": "abc123def456",
  "go_version": "go1.21.0"
}
```

2.5.6.3.1.33 Notes

- Useful for verifying DPS server connectivity and version compatibility
- Can be used in scripts to check server availability

2.5.6.3.1.34 dpsctl topology

2.5.6.3.1.35 dpsctl topology Usage Guide

2.5.6.3.1.36 Overview

`dpsctl topology` includes DPS Topology control commands. For the clean-install validate and import workflow, refer to *Import a Topology*.

2.5.6.3.1.37 Usage

```
dpsctl topology [command [command options]]
```

2.5.6.3.1.38 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.39 Commands

- *`validate`* - Validate a topology file
- *`list`* - List topologies
- *`remove-entities`* - Remove entities from topology
- *`list-entities`* - List topology entities
- *`import`* - Import topology from a JSON file
- *`update`* - Review the current CLI update limitation
- *`export`* - Export a topology to JSON
- *`activate`* - Activate a topology
- *`deactivate`* - Deactivate a topology
- *`remove`* - Remove a topology
- `help`, h - Shows a list of commands or help for one command

2.5.6.3.1.40 list

2.5.6.3.1.41 dpsctl topology list Usage Guide

List topologies

2.5.6.3.1.42 Usage

```
dpsctl topology list
```

2.5.6.3.1.43 Flags

Includes *global `dpsctl` options*.

```
  --active            Is active (optional, true/false)
  --help, -h          show help
```

2.5.6.3.1.44 Examples

2.5.6.3.1.45 Basic Usage

```
$ dpsctl topology list
{
  "topologies": [
    {
      "topology_name": "topology",
      "is_active": false,
      "leaf_node_names": [
        "node001",
        "node002",
        "node003",
        "node004"
      ],
      "policy_bundle": "MaxP"
    }
  ]
}
```

The `policy_bundle` field is the bundle selection persisted on each topology. Use it to verify the current persisted selection. An empty value means no bundle is configured.

2.5.6.3.1.46 Filter on active status

```
$ dpsctl topology list --active=false
{
  "topologies": [
    {
      "topology_name": "topology",
      "is_active": false,
      "leaf_node_names": [
        "node001",
        "node002",
        "node003",
        "node004"
      ],
      "policy_bundle": "MaxP"
    }
  ]
}
```

If `--active` is omitted, all topologies are listed. Use `--active=false` after import to confirm the topology is present and inactive.

2.5.6.3.1.47 Get name of topology containing node

Here, the shortcut `tp` is used in place of `topology`.

```
$ dpsctl tp list | jq -c '.topologies[] | select(.leaf_node_names[] |
↪contains("node003")) | .topology_name'
"topology"
```

2.5.6.3.1.48 validate

2.5.6.3.1.49 dpsctl topology validate Usage Guide

Validate a topology JSON file without importing it.

Validation is two-layered. The CLI first validates the file against the embedded JSON schema. It then streams the model to `TopologyManagementService.ValidateTopology` for semantic checks. Closing the stream starts server validation. The call does not modify the database.

2.5.6.3.1.50 Usage

```
dpsctl topology validate <filename>
```

2.5.6.3.1.51 Flags

Includes *global `dpsctl` options*.

```
   --help, -h          show help
```

2.5.6.3.1.52 Examples

2.5.6.3.1.53 Basic Validation

```
$ dpsctl topology validate topology.json
{
  "status": {
    "ok": true,
    "diag_msg": "Topology validation passed"
  }
}
```

JSON output omits `validation_errors` when the list is empty.

2.5.6.3.1.54 Validation with Errors

```
$ dpsctl topology validate invalid-topology.json
{
  "status": {
    "ok": false,
    "diag_msg": "Topology validation failed with 3 validation error(s)"
  },
  "validation_errors": [
```

(continues on next page)

(continued from previous page)

```
    {
      "object_name": "node001",
      "object_type": "entity",
      "error": "device_not_found",
      "message": "Entity 'node001' references undefined device model
↪'UnknownModel'"
    },
    {
      "object_name": "node001",
      "object_type": "topology_entity",
      "error": "referenced_entity_not_found",
      "message": "Referenced entity not found"
    },
    {
      "object_name": "pdn",
      "object_type": "topology",
      "error": "circular_dependency",
      "message": "Circular dependency detected in topology relations"
    }
  ]
}
```

The `message` strings in the failure example are illustrative. Match failures by the `error` code.

2.5.6.3.1.55 Notes

- A topology JSON filename is required.
- Local schema failures report instance location. JSON syntax errors report line, column, and offset. Those failures do not produce `validation_errors`.
- A non-empty top-level `Policies` array is rejected before the server call.
- Server validation checks entity references, device models, and logical consistency, including cycles and disconnected entities.
- Files are validated against the same schema used during import.
- Validation does not modify any existing topology data.
- The `status.ok` field indicates overall semantic validation success or failure.
- The `validation_errors` array contains object name, type, error code, and message for each server-side failure.
- Server error codes include `device_not_found`, `invalid_model`, `circular_dependency`, `disconnected_graph`, `referenced_entity_not_found`, `duplicate_entity`, `self_reference`, and `invalid_secret_name`.

2.5.6.3.1.56 import

2.5.6.3.1.57 `dpsctl topology import` Usage Guide

Import a datacenter topology from a JSON file. The file must match the published topology schema. Refer to *Import a Topology* for the full workflow and an example file.

Import creates missing entities and then creates the named topology. The created topology is inactive. There is no single import RPC; the CLI orchestrates `ListTopologies`, `GetEntities`, `CreateEntities`, and `CreateUpdateTopology`.

2.5.6.3.1.58 Usage

```
dpsctl topology import <filename>
```

2.5.6.3.1.59 Flags

Includes *global `dpsctl` options*.

```
   --help, -h        show help
```

2.5.6.3.1.60 Examples

2.5.6.3.1.61 Basic Usage

```
$ dpsctl topology import topology.json
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.62 Notes

- A topology JSON filename is required.
- The file must include a `Topology` object with at least one entity.
- A nonempty top-level `Policies` array is rejected by the DPS 0.9 Go CLI before the import request is sent. Create typed policy bundles with `dpsctl policy bundle upsert`.
- The topology JSON cannot carry a bundle selection. Import resolves the current `global_policy_bundle` setting and persists it on the new topology. Set that setting to the intended bundle before import; importing the same file under a different default selects a different bundle.
- Import does not activate the topology or apply BMC power policies.
- If the topology name already exists, import fails before entity creation.
- If entity creation succeeds and topology creation fails, the new entities remain.

2.5.6.3.1.63 activate

2.5.6.3.1.64 `dpsctl topology activate` Usage Guide

Activate a topology. Deactivate the current topology before you activate a different topology.

2.5.6.3.1.65 Usage

```
dpsctl topology activate
```

2.5.6.3.1.66 Flags

Includes *global dpsctl options*.

```
  --topology value                    topology name (required)
  --replace-topology                  request active-topology replacement
↪(currently rejected; deactivate first) (default: false)
  --ping-hosts                        test connectivity of all hosts in
↪topology (default: true)
  --at-least-percent-hosts value      at least this percent of hosts must be
↪reachable for activation to succeed (requires ping-hosts) (default: 50)
  --help, -h                          show help
```

2.5.6.3.1.67 Examples

2.5.6.3.1.68 Basic Usage

Before activation, inspect `dpsctl topology list` and confirm that the topology has the intended `policy_bundle`.

```
dpsctl topology activate --topology datacenter
```

Activation applies the effective typed policies from the selected bundle. A legacy per-entity `Policy` value can remain in topology data for compatibility, but it does not select a bundle or create an activation binding.

Verify the result without relying on a fixed response example:

```
dpsctl topology list --active
dpsctl check status --nodes node001,node002
```

The topology list confirms the active topology and its selected bundle. `check status` reports each node's allocated policy identity and access state.

2.5.6.3.1.69 `--replace-topology` Compatibility

Released 0.9 retains `--replace-topology` for command compatibility, but rejects switching while another topology is active. Deactivate the active topology before activating a different one.

2.5.6.3.1.70 Notes

- Activation applies the effective policies composed from the topology's selected policy bundle.
- Use `--ping-hosts` to test connectivity of all hosts
- Use `--at-least-percent-hosts` to set failure tolerance for activation
- Deactivate the active topology before activating another topology.
- The `--replace-topology` flag is retained for compatibility, but active topology replacement is currently rejected.

2.5.6.3.1.71 export

2.5.6.3.1.72 dpsctl topology export Usage Guide

Export the named topology to JSON. By default, export includes the topology and the entities it references. Use `--entities` to include every entity in the database.

2.5.6.3.1.73 Usage

```
dpsctl topology export
```

2.5.6.3.1.74 Flags

Includes *global dpsctl options*.

```
   --topology value    topology name (required)
   --entities          export entities
   --help, -h          show help
```

2.5.6.3.1.75 Examples

2.5.6.3.1.76 Default Export

```
$ dpsctl topology export --topology datacenter
{
  "Entities": [
    {
      "Type": "PowerDomain",
      "Name": "PD-A",
      "OperatingLimit": {
        "PowerValue": {
          "Value": 1150000,
          "Type": "W"
        },
        "PowerFactor": 0.9
      }
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_H100",
      "Name": "node001"
    }
  ],
  "Topology": {
    "Name": "datacenter",
    "Entities": [
      {
        "Name": "PD-A",
        "Children": ["node001"]
      },
      {
        "Name": "node001"
```

(continues on next page)

(continued from previous page)

```
      }
    ]
  }
}
```

2.5.6.3.1.77 Export with All Entities

```
$ dpsctl topology export --topology datacenter --entities
```

The output uses the same `Entities` and `Topology` members. `--entities` includes entities that are not referenced by the topology.

2.5.6.3.1.78 Save to File

```
$ dpsctl topology export --topology datacenter > exported-topology.json
```

2.5.6.3.1.79 Notes

- Exported JSON uses `Entities` and `Topology`. Remove a nonempty `Policies` member before re-import.
- Exported JSON is compatible with `dpsctl topology import`, but it is not a complete portable configuration.
- Export sanitization deliberately removes `topologyHash`. API clients can get the hash from the `GetTopologyModel` response header when preparing a non-forced API update.
- The current Go CLI update limitations are documented in *update*.
- Use `--entities` to include every entity in the database rather than only topology members.
- Export does not include the topology's persisted policy-bundle selection. Recreate bundles with `dpsctl policy bundle upsert`. To recreate the intended power behavior in another DPS instance:
  1. Upsert the source topology's selected typed bundle on the destination.
  2. Before importing, ensure the destination's `global_policy_bundle` resolves to that bundle. If you change the setting for import, restore its saved value afterward.
  3. After import, use `dpsctl topology list` to verify the imported topology's `policy_bundle`.
- An exported entity `policy` value is legacy. It does not restore the typed bundle selection.
- Refer to *policy-bundle selection* for resolution and lifecycle behavior.

2.5.6.3.1.80 deactivate

2.5.6.3.1.81 `dpsctl topology deactivate` Usage Guide

Deactivate the named active topology. Deactivation stops DPS power management for the topology and attempts to release the DPS-managed policy from each managed node.

You cannot deactivate a topology while it has active resource groups. Delete every active resource group before you run this command.

2.5.6.3.1.82 Behavior

When you deactivate a topology, DPS performs the following actions:

1. Marks the topology inactive and removes its active topology policy binding.
2. Sends an `UnsetPolicy` request to each node that requires policy cleanup.
3. Applies the device's configured idle policy when one exists. Otherwise, the device plugin restores BMC defaults or removes the DPS-managed policy, depending on the platform.
4. Stops active power management for the topology. The topology remains available for later activation or removal.

Deactivation is different from stopping the DPS service. A planned DPS shutdown, accidental DPS server or agent pod deletion, or temporary DPS outage does not send `UnsetPolicy`. If the BMC remains operational, it continues enforcing its last successfully applied power limits.

Important

Device cleanup is best effort. A successful response confirms that the topology is inactive, but it does not confirm that every BMC accepted the cleanup request. Verify BMC power limits when your maintenance procedure requires a specific hardware state.

2.5.6.3.1.83 Usage

```
dpsctl topology deactivate
```

2.5.6.3.1.84 Flags

Includes *global `dpsctl` options*.

```
   --topology value    topology name (required)
   --help, -h          show help
```

2.5.6.3.1.85 Examples

2.5.6.3.1.86 Basic Usage

```
$ dpsctl topology deactivate --topology topology
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.87 Verify Deactivation

Confirm that no topology is active and that the deactivated topology appears in the inactive list:

```
dpsctl topology list --active=true
dpsctl topology list --active=false
```

For outage behavior and maintenance guidance, refer to *Topology Deactivation and Loss of DPS Control* and *Prepare Nodes for Maintenance*.

2.5.6.3.1.88 update

2.5.6.3.1.89 dpsctl topology update Usage Guide

Update an existing topology from a JSON file.

2.5.6.3.1.90 Usage

```
dpsctl topology update <filename>
```

2.5.6.3.1.91 Flags

Includes *global dpsctl options*.

```
   --force             force update topology
   --help, -h          show help
```

2.5.6.3.1.92 Examples

2.5.6.3.1.93 Basic Update

```
$ dpsctl topology update updated-topology.json
{
  "status": {
    "ok": true,
    "diag_msg": "Topology updated successfully"
  }
}
```

2.5.6.3.1.94 Force Update

```
$ dpsctl topology update topology.json --force
{
  "status": {
    "ok": true,
    "diag_msg": "Topology forcefully updated"
  }
}
```

2.5.6.3.1.95 Update with Validation Errors

```
$ dpsctl topology update invalid-topology.json
{
  "status": {
    "ok": false,
    "diag_msg": "Topology update failed: validation errors found"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

2.5.6.3.1.96 Notes

- A topology JSON filename is required.
- Non-forced updates require the input file to include `topologyHash` for the existing topology.
- The topology must be inactive before it can be updated.
- This Go command omits the policy-bundle field, so it preserves the current topology selection. An API client can explicitly update the optional `policy_bundle` field while the topology is inactive.
- The `--force` flag bypasses the missing `topologyHash` client-side check. The topology model is still validated before it is applied.
- Always validate the JSON file with `dpsctl topology validate` first

2.5.6.3.1.97 remove

2.5.6.3.1.98 dpsctl topology remove Usage Guide

Remove a datacenter topology.

2.5.6.3.1.99 Usage

```
dpsctl topology remove
```

2.5.6.3.1.100 Flags

Includes *global dpsctl options*.

```
   --topology value    topology name (required)
   --help, -h          show help
```

2.5.6.3.1.101 Examples

2.5.6.3.1.102 Basic Usage

```
$ dpsctl topology remove --topology old-datacenter
{
  "status": {
    "ok": true,
    "diag_msg": "Topology removed successfully"
  }
}
```

2.5.6.3.1.103 Remove Active Topology (Error)

```
$ dpsctl topology remove --topology active-datacenter
{
  "status": {
    "ok": false,
    "diag_msg": "Cannot remove active topology. Deactivate first."
  }
}
```

2.5.6.3.1.104 Remove Non-Existent Topology

```
$ dpsctl topology remove --topology non-existent
{
  "status": {
    "ok": false,
    "diag_msg": "Topology 'non-existent' not found"
  }
}
```

2.5.6.3.1.105 Notes

- Topology must be deactivated before removal
- This operation is irreversible
- All topology relations and configurations are permanently deleted
- Entities and policies may remain if used by other topologies
- Use `dpsctl topology deactivate` first if the topology is active

2.5.6.3.1.106 list-entities

2.5.6.3.1.107 `dpsctl topology list-entities` Usage Guide

List all inventory entities or only the requested entity names.

2.5.6.3.1.108 Usage

```
dpsctl topology list-entities [entity-names...]
```

2.5.6.3.1.109 Flags

Includes *global `dpsctl` options*.

```
   --help, -h          show help
```

2.5.6.3.1.110 Examples

List every entity:

```
dpsctl topology list-entities
```

The JSON output uses the same case-sensitive entity fields as topology JSON:


```
{
  "Entities": [
    {
      "Type": "PowerDomain",
      "Name": "PD-A",
      "OperatingLimit": {
        "PowerValue": {
          "Value": 1150000,
          "Type": "W"
        },
        "PowerFactor": 0.9
      },
      "UpdatedAt": "2026-09-02T18:00:00Z"
    },
    {
      "Type": "ComputerSystem",
      "Model": "DGX_H100",
      "Name": "node001",
      "Redfish": {
        "URL": "https://node001-bmc.example.com",
        "SecretName": "node001"
      },
      "UpdatedAt": "2026-09-02T18:00:00Z"
    }
  ]
}
```


List selected entities:

```
dpsctl topology list-entities node001 node002
```

2.5.6.3.1.111 Notes

- Entity field names are case-sensitive.
- `OperatingLimit` is the current power-cap field. `Constraints` is not a typed operating limit.
- The legacy `Policy` field does not bind a typed policy bundle.
- Use this command to inspect inventory before building or verifying a topology.

2.5.6.3.1.112 remove-entities

2.5.6.3.1.113 dpsctl topology remove-entities Usage Guide

Permanently deletes entities from the system.

Note: The entities must not be referenced by any topology (active or inactive). If an entity is part of any topology, the deletion will fail.

2.5.6.3.1.114 Usage

```
dpsctl topology remove-entities [entity-names...]
```

2.5.6.3.1.115 Flags

Includes *global dpsctl options*.

```
   --help, -h          show help
```

2.5.6.3.1.116 Examples

2.5.6.3.1.117 Delete Unused Entities

```
$ dpsctl topology remove-entities node001 node002
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.118 Error: Entity in Use

```
$ dpsctl topology remove-entities node001
{
  "status": {
    "ok": false,
    "diag_msg": "entit(ies) referenced from existing topologies"
  },
  "entity_topologies": {
     "node001": {
        "topologies": ["my-topology"]
     }
  }
}
```

2.5.6.3.1.119 Notes

- This operation permanently deletes entities from the database.
- At least one entity name is required.
- Deletion is global (system-wide), not scoped to a topology.
- To remove entities from a topology without deleting them from the system, update the topology definition through `CreateUpdateTopology`. The current Go CLI update limitation is documented in *update*.

2.5.6.3.1.120 dpsctl login

2.5.6.3.1.121 Overview

Use `dpsctl login` to sign in and store credentials for later DPS API requests. In LDAP mode, use a username and password. In OIDC mode, use browser sign-in through the device authorization flow. Your administrator provides the configured mode and connection settings. Refer to *Configure Server Authentication* for server setup.

2.5.6.3.1.122 Usage

Run the command with the global endpoint and TLS options for your deployment:

```
dpsctl login [flags]
```

2.5.6.3.1.123 Flags

The command accepts *global options* and the following login options:

```
   --username string, -u string  The username to use to authenticate to DPS [
↪$DPSCTL_USERNAME]
   --password-stdin              Read password from stdin (e.g. printf '%s\n'
↪'pass' | dpsctl login -u user --password-stdin) (default: false)
   --id-token                    Use the OIDC ID token for requests to DPS
↪from this browser login session (default: false)
   --help, -h                    show help
```

2.5.6.3.1.124 LDAP

2.5.6.3.1.125 Log In with a Password Prompt

In LDAP mode, `login` prompts for the username and password when no credentials are supplied or saved:

```
$ dpsctl login
Please enter the value for username: alice
Please enter the value for password:
{
  "status": {
    "ok": true,
    "diag_msg": "Login successful"
  }
}
```

2.5.6.3.1.126 Log In with a Password from Standard Input

You can supply a password file from your secret-management workflow through standard input with an explicit username:

```
$ dpsctl login --username alice --password-stdin < /path/to/password-file
{
  "status": {
```

(continues on next page)

(continued from previous page)

```
    "ok": true,
    "diag_msg": "Login successful"
  }
}
```

2.5.6.3.1.127 Log In with Environment Variables

Set DPSCTL_PASSWORD through your secret-management workflow, and provide the username through an environment variable:

```
$ export DPSCTL_USERNAME=alice
$ dpsctl login
{
  "status": {
    "ok": true,
    "diag_msg": "Login successful"
  }
}
```

2.5.6.3.1.128 OIDC

2.5.6.3.1.129 Browser Login

Provide the issuer and client ID configured by your administrator. The interactive `dpsctl` client must be registered as a public client without a client secret. It sends no secret during token exchange. The provider must support the device authorization flow.

Use the following command to start browser login:

```
dpsctl --oidc-issuer https://idp.example.com/realms/dps \
  --oidc-client-id dpsctl \
  --oidc-oauth-scope dps:api login
```

Open the URL displayed by `dpsctl`, enter the displayed code if requested, and complete sign-in in your browser. Browser login contacts the provider directly. Set the DPS endpoint and TLS options on later API commands. `dpsctl` stores the access token for those requests. `dpsctl` uses a refresh token supplied by the provider to renew the access token when needed. If the session expires without a usable refresh token, repeat browser login.

Use `--oidc-oauth-scope` for the scopes required by your provider and API configuration. Refer to *OIDC Browser Login* for discovery, endpoint overrides, and certificate options.

2.5.6.3.1.130 Authorization with an ID Token

API requests use the access token by default. Use `--id-token` only when your deployment's API authorization configuration accepts an ID token:

```
dpsctl --oidc-issuer https://idp.example.com/realms/dps \
  --oidc-client-id dpsctl \
  --oidc-oauth-scope dps:api login --id-token
```

This option requests the `openid` scope automatically and stores the ID token for API requests from

that session. It requires OIDC browser login without username/password options. Follow your administrator's token requirements in *Configure OIDC Mode*.

2.5.6.3.1.131 Mutual TLS

Certificate authentication requires no `dpsctl login` step. Supply the client certificate and key on API commands as described in *Mutual TLS*.

2.5.6.3.1.132 Notes

The following details apply when you configure login:

- `dpsctl` stores credentials in `${HOME}/.dpsctl/credentials.yaml` by default.
- Set `--credentials-path` to use a different file.
- In LDAP mode, `dpsctl` uses a valid stored refresh token to renew expiring access tokens before API requests. Repeat `dpsctl login` if the refresh token is missing, expired, or no longer accepted.
- A password supplied through `--password-stdin` or `DPSCTL_PASSWORD` requires `--username` or `DPSCTL_USERNAME`.
- Providing username/password options selects the DPS password login flow, even when OIDC options are also configured.
- OIDC browser login cannot be combined with client credentials. For unattended services, refer to *Obtain a Service Account Token*.
- Successful login confirms authentication. A later API request can still fail authorization. Follow *Verify the Configuration* to check access.

2.5.6.3.1.133 dpsctl resource-group

2.5.6.3.1.134 dpsctl resource-group Usage Guide

2.5.6.3.1.135 Overview

`dpsctl resource-group` includes DPS Resource Group control commands.

2.5.6.3.1.136 Usage

```
dpsctl resource-group [command [command options]]
```

2.5.6.3.1.137 Flags

Includes *global dpsctl options*.

```
   --help, -h  show help
```

2.5.6.3.1.138 Commands

- *create* - Create a resource group
- *delete* - Delete a resource group
- *add* - Add resources to a resource group

- *remove* - Remove resources from a resource group
- *activate* - Activate a resource group
- *update* - Update a resource group
- *list* - List resource groups
- *status* - Query the status of an ongoing asynchronous operation
- help, h - Shows a list of commands or help for one command

`--priority` on create and update sets steal-band order. Giveback uses the same numbers only when a complete restore cannot be admitted. Refer to *Resource Group Priority*.

2.5.6.3.1.139 list

2.5.6.3.1.140 dpsctl resource-group list Usage Guide

List resource groups.

2.5.6.3.1.141 Usage

```
dpsctl resource-group list
```

2.5.6.3.1.142 Flags

Includes *global dpsctl options*.

```
  --active    optional filter by active state (true/false)
  --help, -h  show help
```

2.5.6.3.1.143 Examples

2.5.6.3.1.144 Basic Usage

```
$ dpsctl resource-group list
{
  "resource_groups": [
    {
      "group_name": "test",
      "external_id": 1,
      "activation_status": "inactive",
      "policy_name": "",
      "resource_names": [],
      "properties": {},
      "prs_enabled": true,
      "dpm_enable": true,
      "priority": 0,
      "policy_bundle": "MaxP"
    },
    {
      "group_name": "test-2",
      "external_id": 2,
```

(continues on next page)

(continued from previous page)

```
      "activation_status": "inactive",
      "policy_name": "",
      "resource_names": [
        "node001",
        "node002",
        "node003"
      ],
      "properties": {
        "prs-enabled": false
      },
      "prs_enabled": false,
      "dpm_enable": true,
      "priority": 0,
      "policy_bundle": "MaxQ"
    },
    {
      "group_name": "test-3",
      "external_id": 3,
      "activation_status": "inactive",
      "policy_name": "",
      "resource_names": [
        "node004"
      ],
      "properties": {
        "prs-enabled": true
      },
      "prs_enabled": true,
      "dpm_enable": true,
      "priority": 0,
      "policy_bundle": "MaxLPS"
    }
  ]
}
```

The `policy_bundle` field is the bundle selection persisted on each resource group. Use it to verify the create-time selection. An empty value means no bundle is configured. The `policy_name` field is retained for API compatibility and is not the policy bundle name. The Go CLI bundle-only create path leaves it empty.

2.5.6.3.1.145 Filter on active status

```
dpsctl -H localhost -p 8080 rg list --active
{
  "resource_groups": [
    {
      "group_name": "test-2",
      "external_id": 2,
      "activation_status": "active",
      "policy_name": "",
      "resource_names": [
        "node003",
```

(continues on next page)

(continued from previous page)


```
        "node002",
        "node001"
      ],
      "properties": {},
      "prs_enabled": true,
      "dpm_enable": true,
      "priority": 0,
      "policy_bundle": "MaxQ"
    },
    {
      "group_name": "test-3",
      "external_id": 3,
      "activation_status": "active",
      "policy_name": "",
      "resource_names": [
        "node004"
      ],
      "properties": {},
      "prs_enabled": true,
      "dpm_enable": true,
      "priority": 0,
      "policy_bundle": "MaxLPS"
    }
  ]
}
```


If `--active` is omitted, all resource groups are listed. The `priority` field is the stored resource group priority. `0` is the default and the highest precedence. Refer to *Resource Group Priority*.


```
$ dpsctl resource-group list --active=false
{
  "resource_groups": [
    {
      "group_name": "test",
      "external_id": 1,
      "activation_status": "inactive",
      "policy_name": "",
      "resource_names": [],
      "properties": {
        "prs-enabled": false
      },
      "prs_enabled": false,
      "dpm_enable": true,
      "priority": 0,
      "policy_bundle": "MaxP"
    }
  ]
}
```

2.5.6.3.1.146 create

2.5.6.3.1.147 dpsctl resource-group create Usage Guide

Create a resource group.

2.5.6.3.1.148 Usage

```
dpsctl resource-group create
```

2.5.6.3.1.149 Flags

Includes *global dpsctl options*.

```
  --external-id int                external ID (required)
  --resource-group string          resource group name (required)
  --policy-bundle string           resource group policy bundle (optional;
↪omit to resolve and persist the current global policy bundle)
  --dpm-enable                     enable dynamic power management for the
↪lifetime of the resource group (default: true)
  --workload-profile-ids string    workload power profile Redfish IDs (comma
↪separated list of profile IDs) (optional)
  --prs-enabled                    enable or disable PRS dynamic power
↪management for this resource group (default: true)
  --shared-gpu-enable              enable or disable shared-GPU behavior for
↪this resource group (default: true)
  --priority int                   priority for power stealing/giveback
↪ordering (0 = default and highest precedence, lower number = higher
↪priority)
  --help, -h                       show help
```

2.5.6.3.1.150 Examples

2.5.6.3.1.151 Basic Usage

If you omit `--policy-bundle`, the server resolves the current `global_policy_bundle` setting when it creates the resource group and persists the resolved bundle name on that resource group. A later change to the global setting does not change the existing resource group's selection. If the global setting is empty, the resource group has no configured bundle.

```
$ dpsctl resource-group create --resource-group example1 --external-id 1
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.152 Create a resource group with a policy bundle

Select the required bundle when you create the resource group.

```
$ dpsctl resource-group create --resource-group example1 --external-id 1 --
↪policy-bundle MaxP
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.153 Create a resource group with workload profile IDs

```
$ dpsctl resource-group create --resource-group example1 --external-id 1 --
↪policy-bundle MaxP --workload-profile-ids 3,4
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.154 Create a resource group with prs dynamic power management disabled

By default, prs dynamic power management is enabled for resource groups. You can disable it using the `--prs-enabled=false` flag.

```
$ dpsctl resource-group create --resource-group example1 --external-id 1 --
↪policy-bundle MaxP --prs-enabled=false
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.155 Create a resource group with shared-GPU behavior enabled

Shared GPU mode defaults to on. The following command sets the flag explicitly. Refer to *Shared GPU Budget* for the group-wide budget and topology-constraint checks.

```
$ dpsctl resource-group create --resource-group example1 --external-id 1 --
↪shared-gpu-enable
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.156 Create a resource group with shared-GPU behavior disabled

Pass `--shared-gpu-enable=false` to keep per-node GPU caps instead of a group-wide GPU budget.

```
$ dpsctl resource-group create --resource-group example1 --external-id 1 --
↪shared-gpu-enable=false
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.157 Create a resource group with priority

Lower numbers are higher priority. Omitting `--priority` stores `0`, which outranks every positive value. `--priority 10` is less important than any group left at the default. This command only persists the value; redistribution runs later when giveback or stealing is triggered. Refer to *Resource Group Priority*.

```
$ dpsctl resource-group create --resource-group example1 --external-id 1 --
↪priority 10
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.158 delete

2.5.6.3.1.159 dpsctl resource-group delete Usage Guide

Delete a resource group. If the resource group is active, it will be deactivated.

2.5.6.3.1.160 Usage

```
dpsctl resource-group delete
```

2.5.6.3.1.161 Flags

Includes *global dpsctl options*.

```
  --resource-group string                  resource group name
  --wpps-disable-async-verification        disable asynchronous verification
↪for workload power profile service (WPPS) operations during resource group
↪deletion
  --help, -h                               show help
```

2.5.6.3.1.162 Examples

2.5.6.3.1.163 Basic Usage

```
$ dpsctl resource-group delete --resource-group example1
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.164 status

2.5.6.3.1.165 dpsctl resource-group status Usage Guide

Get the status of a resource group operation.

2.5.6.3.1.166 Usage

```
dpsctl resource-group status
```

2.5.6.3.1.167 Flags

Includes *global dpsctl options*.

There are two mutually exclusive ways to query operation status:

Query by Resource Group:

```
   --resource-group string          resource group name (required)
   --operation-type string          operation type: 'activate' or 'update'
↪(required)
```

Query by Operation ID:

```
   --operation-id string            operation ID
```

```
   --help, -h                      show help
```

2.5.6.3.1.168 Examples

2.5.6.3.1.169 Query by Resource Group

```
$ dpsctl resource-group status --resource-group example1 --operation-type
↪activate
{
  "Status": {
    "Activate": {
      "status": {
        "diag_msg": "resource group example1 is already active"
      }
```

(continues on next page)

(continued from previous page)

```
    }
  },
  "completed": true,
  "operation_id": "08041263-b152-4743-b1b4-09c202229139"
}
```

2.5.6.3.1.170 Query by Operation ID

```
$ dpsctl resource-group status --operation-id 08041263-b152-4743-b1b4-
↪09c202229139
{
  "Status": {
    "Activate": {
      "status": {
        "diag_msg": "resource group example1 is already active"
      }
    }
  },
  "completed": true,
  "operation_id": "08041263-b152-4743-b1b4-09c202229139"
}
```

2.5.6.3.1.171 Notes

- Use `--resource-group` with `--operation-type` to query the latest operation for a given resource group, or use `--operation-id` to query a specific operation directly
- The `completed` field indicates whether the operation has finished

2.5.6.3.1.172 add

2.5.6.3.1.173 dpsctl resource-group add Usage Guide

Add resources to a resource group.

Resources can be added to both **inactive** and **active** resource groups:

- **Inactive resource groups**: Adding resources is a database-only operation. The resource group's configured policy bundle is applied when the resource group is activated.
- **Active resource groups**: The resource group's configured policy bundle is applied to the added resources immediately, and power is reallocated as needed.

Note: Adding resources to a resource group that is currently activating or deactivating will be rejected. Wait for the operation to complete before adding resources.

2.5.6.3.1.174 Usage

```
dpsctl resource-group add
```

2.5.6.3.1.175 Flags

Includes *global `dpsctl` options*.

```
  --resource-group string  resource group name
  --entities string        resource entity names (comma-separated list)
  --strict-policy          if the configured policy bundle is not feasible,
↪fail instead of retrying it at the effective policy floor (only applies
↪when resource group is active)
  --allow-reprovision      allow power policy reprovisioning of other
↪resource groups if there is not enough power (only applies when resource
↪group is active; default: true)
  --help, -h               show help
```

2.5.6.3.1.176 Examples

2.5.6.3.1.177 Add entities to a resource group

```
$ dpsctl resource-group add --resource-group example1 --entities node001,
↪node002
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.178 Add entities to an active resource group with strict policy enforcement

When adding resources to an active resource group, use `--strict-policy` to require its configured policy bundle. If the bundle is not feasible, the operation fails instead of retrying at the effective policy floor.

```
$ dpsctl resource-group add --resource-group example1 --entities node004 --
↪strict-policy
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.179 Add entities without allowing power reprovisioning

By default, when adding resources to an active resource group, power may be redistributed from other resource groups if needed (power stealing). Use `--allow-reprovision=false` to prevent this behavior.

```
$ dpsctl resource-group add --resource-group example1 --entities node005 --
↪allow-reprovision=false
{
  "status": {
```

(continues on next page)

(continued from previous page)

```
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.180 remove

2.5.6.3.1.181 dpsctl resource-group remove Usage Guide

Remove resources (entities) from a resource group.

Resources can be removed from both **inactive** and **active** resource groups:

- **Inactive resource groups**: Removing resources is a database-only operation.
- **Active resource groups**: Policies are reverted to topology defaults and power is reallocated as needed.

Note: Removing resources from a resource group that is currently activating or deactivating will be rejected. Wait for the operation to complete before removing resources.

2.5.6.3.1.182 Usage

```
dpsctl resource-group remove
```

2.5.6.3.1.183 Flags

Includes *global dpsctl options*.

```
  --resource-group string  resource group name
  --entities string        resource entity names (comma-separated list)
  --help, -h               show help
```

2.5.6.3.1.184 Examples

2.5.6.3.1.185 Basic Usage

```
$ dpsctl resource-group remove --resource-group example1 --entities node003
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.186 Remove entities from an active resource group

When removing resources from an active resource group, policies on the removed entities are automatically reverted to the topology defaults and power is reallocated.

```
$ dpsctl resource-group remove --resource-group ml-training-job --entities
↪node004,node005
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.187 activate

2.5.6.3.1.188 dpsctl resource-group activate Usage Guide

Activate a resource group. You must first *create a resource group* before it can be activated.

2.5.6.3.1.189 Usage

```
dpsctl resource-group activate
```

2.5.6.3.1.190 Flags

Includes *global dpsctl options*.

```
  --resource-group string       resource group name
  --strict-policy               if the requested allocation is not possible,
↪fail instead of accepting a reduced allocation (default: false)
  --sync                        synchronous activation. Wait until resource
↪group activation is complete (default: false)
  --async                       asynchronous activation (default). Return
↪immediately after validating the resource group, but activation continues
↪asynchronously
  --async-n-hosts int           asynchronous activation, but wait until at
↪least this many hosts are configured before returning (default: 0)
  --async-percent-hosts int     asynchronous activation, but wait until at
↪least this percent of hosts are configured before returning (default: 0)
  --async-wait duration         asynchronous activation, but wait this much
↪before returning (default: 0s)
  --at-least-n-hosts int        at least this number of hosts must be
↪activated for the resource group activation to succeed. Remaining hosts may
↪fail to be configured, but they remain part of the resource group (default:
↪0)
  --at-least-percent-hosts int  at least this percent of hosts must be
↪activated for the resource group activation to succeed. Remaining hosts may
↪fail to be configured, but they remain part of the resource group (default:
↪0)
  --partial-timeout duration    wait at most this much for the resource group
↪activation to come to an acceptable level. If the resource group cannot be
↪activated for the given failure tolerance levels within this duration,
↪activation fails. (default: 0s)
  --allow-reprovision           allow power policy reprovisioning of other
↪resource groups in order to activate the resource group if there is not
```

(continues on next page)

(continued from previous page)

```
↪enough power to activate the resource group (default: true)
  --help, -h                    show help
```

2.5.6.3.1.191 Async and Partial Completion Rules

- If neither `--sync` nor `--async` is specified, activation runs asynchronously by default.
- `--sync` cannot be combined with async return controls.
- At most one of `--async-n-hosts`, `--async-percent-hosts`, or `--async-wait` can be used.
- At most one of `--at-least-n-hosts` or `--at-least-percent-hosts` can be used.
- Percent values must be between 0 and 100.

2.5.6.3.1.192 Examples

2.5.6.3.1.193 Activation with Normal Allocation

Activate the resource group after creating it, adding its entities, and confirming the selected `policy_bundle` with `dpsctl resource-group list`.

Without `--strict-policy`, DPS can admit a reduced allocation within the effective policy constraints. When `--allow-reprovision` is enabled, it can also adjust eligible active resource groups to make room. This is allocation from the effective typed policies, not fallback through legacy names such as `Node-High`, `Node-Med`, or `Node-Low`.

```
dpsctl resource-group activate --resource-group test-2
```

2.5.6.3.1.194 Activation with Strict Policy Application

Use `--strict-policy` when the requested allocation must be admitted without reduction. The request fails if that allocation cannot be satisfied.

```
dpsctl resource-group activate \
  --resource-group test-2 \
  --strict-policy \
  --sync
```

2.5.6.3.1.195 Apply Per-GPU Limits After Activation

Explicit per-GPU limits use the standalone `dpsctl gpu-policy` command on nodes in an active resource group. They are not configured through a resource-group update flag or as per-GPU entries in the selected bundle.

```
dpsctl gpu-policy --node node001=500,550
```

Refer to *dpsctl gpu-policy* for validation and response details.

2.5.6.3.1.196 update

2.5.6.3.1.197 dpsctl resource-group update Usage Guide

Update supported non-policy settings on a resource group.

Important

In DPS 0.9, the CLI exposes `--policy-bundle` and `--remove-policy-bundle`, but the server rejects either option with `NOT_IMPLEMENTED` and the message `resource group policy updates are not supported`. The server rejects the request before it distinguishes between active and inactive groups. Select the bundle when you create the resource group. To use a different bundle, delete and recreate the group.

Use `dpsctl gpu-policy` for per-GPU watt limits. The `resource-group update` command has no `--entity-policy` or `--entity-gpu-policy` flag.

2.5.6.3.1.198 Usage

```
dpsctl resource-group update --resource-group <name> [flags]
```

2.5.6.3.1.199 Flags

Includes *global `dpsctl` options*.

```
  --resource-group string              resource group name
  --policy-bundle string               currently rejected by the server
  --remove-policy-bundle               currently rejected by the server
  --strict-policy                      require the configured policy to
↪apply without automatic fallback
  --workload-profile-ids string        workload power profile Redfish IDs
↪(comma-separated)
  --remove-all-workload-profiles       remove workload power profiles
  --sync                               wait for the update to complete
  --async                              return after validation while the
↪update continues (default)
  --async-n-hosts int                  wait until at least this many hosts
↪are configured
  --async-percent-hosts int            wait until at least this percentage
↪of hosts are configured
  --async-wait duration                wait this long before returning
  --at-least-n-hosts int               minimum number of hosts that must
↪update successfully
  --at-least-percent-hosts int         minimum percentage of hosts that must
↪update successfully
  --partial-timeout duration           maximum wait for the acceptable
↪completion threshold
  --wpps-disable-async-verification    disable asynchronous WPPS verification
  --priority int                       priority for power stealing and
↪giveback ordering
  --help, -h                           show help
```

2.5.6.3.1.200 Async and Partial Completion Rules

- If neither `--sync` nor `--async` is specified, update runs asynchronously.
- `--sync` cannot be combined with asynchronous return controls.
- At most one of `--async-n-hosts`, `--async-percent-hosts`, or `--async-wait` can be used.
- At most one of `--at-least-n-hosts` or `--at-least-percent-hosts` can be used.
- Percentage values must be between 0 and 100.

2.5.6.3.1.201 Update Priority

`--priority` is sent only when you pass the flag. The change is stored immediately and is used the next time giveback or stealing runs. It does not rebalance power by itself. Refer to *Resource Group Priority*.

```
$ dpsctl resource-group update --resource-group example1 --priority 10
{
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```

2.5.6.3.1.202 Update Workload Profiles

Replace the workload-profile IDs:

```
dpsctl resource-group update \
  --resource-group example1 \
  --workload-profile-ids 3,4
```

Remove all workload profiles:

```
dpsctl resource-group update \
  --resource-group example1 \
  --remove-all-workload-profiles
```

2.5.6.3.1.203 dpsctl auth

2.5.6.3.1.204 `dpsctl auth` Usage Guide

2.5.6.3.1.205 Overview

`dpsctl auth` provides authentication helper commands for working with stored DPS credentials.

2.5.6.3.1.206 Usage

```
dpsctl auth [command [command options]]
```

2.5.6.3.1.207 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.208 Commands

- *`token`* - Print stored access token to stdout
- `help`, `h` - Shows a list of commands or help for one command

2.5.6.3.1.209 Notes

- Authentication credentials are loaded from the configured credentials file.
- Use *`dpsctl login`* to create or refresh stored credentials.
- The credentials file path can be customized with the global `--credentials-path` option.

2.5.6.3.1.210 token

2.5.6.3.1.211 dpsctl auth token Usage Guide

Print the current stored access token to stdout.

2.5.6.3.1.212 Overview

The `auth token` command attempts to load credentials from the configured credentials file and refresh them if necessary. If a valid access token is available, the command prints the token to stdout. If no token is available, the command exits with an error.

This command is useful for scripting or debugging workflows that need the current DPS access token.

2.5.6.3.1.213 Usage

```
dpsctl auth token [command options]
```

2.5.6.3.1.214 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.215 Examples

2.5.6.3.1.216 Print Stored Access Token

```
$ dpsctl auth token
eyJhbGciOi...
```

2.5.6.3.1.217 Use a Custom Credentials File

```
$ dpsctl --credentials-path /path/to/credentials.yaml auth token
eyJhbGciOi...
```

2.5.6.3.1.218 Capture Token for a Script

```
ACCESS_TOKEN="$(dpsctl auth token)"
```

2.5.6.3.1.219 No Token Available

```
$ dpsctl auth token
{
  "status": {
    "diag_msg": "Command execution failed with an error",
    "error": {
      "code": "CLIENT_ERROR",
      "error": "no access token available"
    },
    "ok": false
  }
}
```

2.5.6.3.1.220 Notes

- The command writes only the access token to stdout when successful.
- Before printing, the command attempts to load and refresh credentials using the configured credentials file.
- Global connection and TLS options, such as `--host`, `--port`, `--resolve`, `--insecure`, `--insecure-tls-skip-verify`, and `--ca-cert-path`, can affect credential refresh.
- The command returns a structured error when no access token is available.
- Tokens are sensitive credentials. Avoid printing them in shared logs or terminal recordings.

2.5.6.3.1.221 dpsctl policy

2.5.6.3.1.222 dpsctl policy Usage Guide

2.5.6.3.1.223 Overview

In the Go implementation, `dpsctl policy` manages named, typed policy bundles through the `bundle` subgroup.

2.5.6.3.1.224 Usage

```
dpsctl policy [command [command options]]
```

2.5.6.3.1.225 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.226 Commands

- *`bundle`* - Create, inspect, list, and delete policy bundles
- `help`, h - Shows a list of commands or help for one command

2.5.6.3.1.227 bundle

2.5.6.3.1.228 `dpsctl policy bundle` Usage Guide

Use `dpsctl policy bundle` to manage named, typed policy bundles.

2.5.6.3.1.229 Usage

```
dpsctl policy bundle <command> [command options]
```

2.5.6.3.1.230 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.231 Commands

- *`upsert`* - Create or replace one policy bundle
- *`get`* - Retrieve one policy bundle
- *`list`* - List policy bundles
- *`delete`* - Delete one policy bundle
- `help`, h - Shows a list of commands or help for one command

Managing a bundle stores or removes its definition. It does not select the bundle for a topology or resource group.

For the bundle format, built-in bundles, selection workflow, and a downloadable example, refer to *Power Policies*.

2.5.6.3.1.232 upsert

2.5.6.3.1.233 `dpsctl policy bundle upsert` Usage Guide

Create or replace one named, typed policy bundle from a JSON or YAML file.

2.5.6.3.1.234 Usage

```
dpsctl policy bundle upsert <file>
```

`<file>` must contain one policy bundle. Use `-` to read from standard input.

2.5.6.3.1.235 Flags

Includes *global `dpsctl` options*.

```
  --input-format string  Input format: json or yaml; inferred from file
↪extension and required for "-"
  --help, -h             show help
```

2.5.6.3.1.236 Examples

2.5.6.3.1.237 Upsert a YAML Bundle

Use the downloadable `site-balanced.yaml` example from *Power Policies*:

```
dpsctl policy bundle upsert site-balanced.yaml
```

2.5.6.3.1.238 Read a Bundle from Standard Input

```
dpsctl policy bundle upsert --input-format yaml - < site-balanced.yaml
```

2.5.6.3.1.239 Notes

- The input is limited to 4 MiB.
- File extensions `.json`, `.yaml`, and `.yml` are detected automatically.
- The command rejects an attempt to replace a bundle that has bindings.
- Upserting a bundle stores its definition. It does not select or apply the bundle. Refer to *Power Policies* for the selection workflow.

2.5.6.3.1.240 get

2.5.6.3.1.241 `dpsctl policy bundle get` Usage Guide

Retrieve one named policy bundle.

2.5.6.3.1.242 Usage

```
dpsctl policy bundle get <name>
```

2.5.6.3.1.243 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.244 Examples

2.5.6.3.1.245 Get a Built-In Bundle

```
dpsctl policy bundle get MaxLPS
```

2.5.6.3.1.246 Notes

- The command returns the stored bundle definition. It does not report the effective policy applied to a topology, resource group, or device.

2.5.6.3.1.247 list

2.5.6.3.1.248 `dpsctl policy bundle list` Usage Guide

Retrieve one server page of policy bundles.

2.5.6.3.1.249 Usage

```
dpsctl policy bundle list
```

2.5.6.3.1.250 Flags

Includes *global `dpsctl` options*.

```
  --page-size int      Maximum bundles in one server page; 0 uses the server
↪default
  --page-token string  Opaque page token returned by a previous list
  --help, -h           show help
```

2.5.6.3.1.251 Examples

2.5.6.3.1.252 List Policy Bundles

```
dpsctl policy bundle list
```

2.5.6.3.1.253 Request a Page

```
dpsctl policy bundle list --page-size 25
```

2.5.6.3.1.254 Notes

- When more bundles are available, use the returned `nextPageToken` value as the `--page-token` value in the next request.
- Listing bundles confirms which definitions are stored. It does not report which bundle is effective for a topology, resource group, or device.

2.5.6.3.1.255 delete

2.5.6.3.1.256 `dpsctl policy bundle delete` Usage Guide

Delete one named policy bundle.

2.5.6.3.1.257 Usage

```
dpsctl policy bundle delete <name>
```

2.5.6.3.1.258 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.259 Examples

2.5.6.3.1.260 Delete a Custom Bundle

```
dpsctl policy bundle delete site-balanced
```

2.5.6.3.1.261 Notes

- You cannot delete the configured global policy bundle.
- You cannot delete a bundle referenced by a topology or resource group.

2.5.6.3.1.262 Create Policy Bundles

2.5.6.3.1.263 Policy Creation Migration

The Go implementation of `dpsctl` in the DPS 0.9 source does not expose `dpsctl policy create`. Create or replace a named, typed policy bundle with `dpsctl policy bundle upsert`.

2.5.6.3.1.264 Usage

```
dpsctl policy bundle upsert <file>
```

`<file>` contains one JSON or YAML policy bundle. For the supported input formats, flags, and examples, refer to *upsert*.

2.5.6.3.1.265 List Policy Bundles

2.5.6.3.1.266 Policy Listing Migration

The Go implementation of `dpsctl` in the DPS 0.9 source does not expose `dpsctl policy list`. List stored policy bundle definitions with `dpsctl policy bundle list`.

2.5.6.3.1.267 Usage

```
dpsctl policy bundle list
```

For pagination flags and examples, refer to *list*.

Listing bundles confirms which definitions are stored. It does not confirm which bundle a topology or resource group selected, or which policy object a device received.

2.5.6.3.1.268 dpsctl gpu-policy

2.5.6.3.1.269 dpsctl gpu-policy Usage Guide

Set GPU power policies on an active resource group.

Note: The per-GPU values you pass here are applied to each GPU individually. A node-level GPU policy that carries no per-GPU values is instead spread uniformly across the node's GPUs. An applied set limit can still differ from the value you requested when DPS clamps it to the resource-group GPU budget or to the device's per-GPU hardware limits.

`dpsctl gpu-policy` is the released Go CLI path for explicit per-GPU limits. It operates on nodes in active resource groups; these limits are standalone and are not policy-bundle entries. The released resource-group update command has no GPU-policy flag.

Shared GPU mode defaults to on. On a shared-GPU resource group, DPS validates the post-update **group** total against the policy-derived resource-group GPU budget, then against non-compute topology ancestors such as racks and PDN devices. A request can pass every per-node arithmetic check and still be clamped or rejected. Refer to *Shared GPU Budget*.

2.5.6.3.1.270 Usage

```
dpsctl gpu-policy --node <nodeName>=<watts1>,<watts2>,...
```

2.5.6.3.1.271 Flags

Includes *global `dpsctl` options*.

```
  --node key=value1,value2,...  (can be multiple) nodeName=<watts1>,<watts2>,
↪... (e.g. node1=250,250) (required)
  --help, -h                    show help
```

2.5.6.3.1.272 Examples

2.5.6.3.1.273 Basic Usage

This example uses the default shared-GPU path. You have created and activated resource group `example1` with the `MaxP` policy bundle and three members: `node001`, `node002`, and `node003`. You want to set the GPU policy of `node001` to the following values:

- GPU0: 500W
- GPU1: 550W
- GPU2: 600W
- GPU3: 700W

- GPU4: 650W
- GPU5: 700W
- GPU6: 550W
- GPU7: 600W

DPS does not accept or reject this request by comparing `node001` alone to its node policy. It sums the requested watts on `node001` with the currently allocated GPU watts on `node002` and `node003`, then compares that group total to the resource-group GPU budget. That budget is the policy-derived ceiling described in *Shared GPU Budget*, not the live allocated sum. Each GPU value must still fall inside the device hardware min and max. After the group-budget check, the request must also fit non-compute topology ancestors such as racks and PDN devices.

Apply the update with `dpsctl`:

```
$ dpsctl gpu-policy --node node001=500,550,600,700,650,700,550,600
{
  "results": [
    {
      "resource_name": "node001",
      "gpu_id": 0,
      "ok": true,
      "set_limit": 500.0,
      "diag_msg": "Success"
    },
    {
      "resource_name": "node001",
      "gpu_id": 1,
      "ok": true,
      "set_limit": 550.0,
      "diag_msg": "Success"
    },
    {
      "resource_name": "node001",
      "gpu_id": 2,
      "ok": true,
      "set_limit": 600.0,
      "diag_msg": "Success"
    },
    {
      "resource_name": "node001",
      "gpu_id": 3,
      "ok": true,
      "set_limit": 700.0,
      "diag_msg": "Success"
    },
    {
      "resource_name": "node001",
      "gpu_id": 4,
      "ok": true,
      "set_limit": 650.0,
      "diag_msg": "Success"
    },
    {
```

(continues on next page)

(continued from previous page)


```
      "resource_name": "node001",
      "gpu_id": 5,
      "ok": true,
      "set_limit": 700.0,
      "diag_msg": "Success"
    },
    {
      "resource_name": "node001",
      "gpu_id": 6,
      "ok": true,
      "set_limit": 550.0,
      "diag_msg": "Success"
    },
    {
      "resource_name": "node001",
      "gpu_id": 7,
      "ok": true,
      "set_limit": 600.0,
      "diag_msg": "Success"
    }
  ],
  "status": {
    "ok": true,
    "diag_msg": "Success"
  }
}
```


2.5.6.3.1.274 Non-shared GPU resource groups

If you created the group with `--shared-gpu-enable=false`, DPS also enforces a per-node GPU budget. That budget is the effective bundle-derived node limit after reserving CPU and memory. For `node001` with a 10,200 W node ceiling, 1,530 W CPU, and 1,020 W memory, the requested GPU watts plus CPU and memory must fit that node:

`500 W + 550 W + 600 W + 700 W + 650 W + 700 W + 550 W + 600 W + 1,530 W + 1,020 W = 7,400 W`

That per-node sum is not the acceptance test on the default shared-GPU path.

2.5.6.3.1.275 dpsctl settings

2.5.6.3.1.276 `dpsctl settings` Usage Guide

2.5.6.3.1.277 Overview

`dpsctl settings` includes DPS settings operations.

2.5.6.3.1.278 Usage

```
dpsctl settings [command [command options]]
```

2.5.6.3.1.279 Flags

Includes *global `dpsctl` options*.

```
   --help, -h   show help
```

2.5.6.3.1.280 Commands

- *`list`* - List all settings
- *`update`* - Update global settings
- `help`, h - Shows a list of commands or help for one command

2.5.6.3.1.281 list

2.5.6.3.1.282 `dpsctl settings list` Usage Guide

List all global DPS settings.

2.5.6.3.1.283 Usage

```
dpsctl settings list
```

2.5.6.3.1.284 Flags

Includes *global `dpsctl` options*.

```
   --help, -h           show help
```

2.5.6.3.1.285 Examples

2.5.6.3.1.286 Basic Usage

```
$ dpsctl settings list
{
  "settings": {
    "metrics_interval_seconds": "30",
    "prs_enabled": "false"
  },
  "status": {
    "diag_msg": "Success",
    "ok": true
  }
}
```

2.5.6.3.1.287 Notes

- Global settings affect DPS server behavior
- Settings are applied system-wide and persist across server restarts
- Some settings may require server restart to take effect

2.5.6.3.1.288 update

2.5.6.3.1.289 dpsctl settings update Usage Guide

Update global DPS settings.

2.5.6.3.1.290 Usage

```
dpsctl settings update
```

2.5.6.3.1.291 Flags

Includes *global dpsctl options*.

```
   --set value          Setting to update (format: key=value). Can be
↪specified multiple times.
   --help, -h           show help
```

2.5.6.3.1.292 Examples

2.5.6.3.1.293 Update Single Setting

```
$ dpsctl settings update --set prs_enabled=false
{
  "status": {
    "diag_msg": "Success",
    "ok": true
  },
  "updated_settings": {
    "prs_enabled": "false"
  }
}
```

2.5.6.3.1.294 Update Multiple Settings

```
$ dpsctl settings update --set prs_enabled=false --set metrics_interval_
↪seconds=30
{
  "status": {
    "diag_msg": "Success",
    "ok": true
  },
  "updated_settings": {
    "metrics_interval_seconds": "30",
    "prs_enabled": "false"
  }
}
```

2.5.6.3.1.295 Notes

- Changes take effect immediately unless otherwise noted
- Invalid setting names or values will result in an error
- Some settings may require specific formats (e.g., duration values like “30s”, “5m”)
- Use `dpsctl settings list` to see current values before updating

2.5.6.3.1.296 dpsctl import

2.5.6.3.1.297 `dpsctl import` Usage Guide

2.5.6.3.1.298 Overview

`dpsctl import` includes commands for importing entities from external sources into DPS.

Entity import uses an asynchronous workflow:

1. **Start import** — `dpsctl import entity` initiates entity generation from the specified source (BCM or Nautobot). The server runs generation in the background and returns a request ID immediately.
2. **Check status** — `dpsctl import entity-status` polls the server with the request ID, waits for completion, and then streams the generated entities.
3. **Cancel or delete** — `dpsctl import entity-cancel` cancels a request by ID, and `dpsctl import entity-delete` removes a request and its stored results.

Generation runs with a server default timeout of 10 minutes (maximum 30 minutes). Request state is cleaned up 30 minutes after the request is created.

2.5.6.3.1.299 Supported Sources

Source	`--source` value	Description
Base Command Manager (BCM)	`bcm`	Generate entities from BCM cluster inventory
Nautobot	`nautobot`	Generate entities from Nautobot DCIM inventory via the Topology Management gRPC service

2.5.6.3.1.300 Usage

```
dpsctl import [command [command options]]
```

2.5.6.3.1.301 Commands

- *`entity`* - Start entity generation from an external source
- *`entity-status`* - Get status and results for an entity import request
- *`entity-cancel`* - Cancel an entity import request
- *`entity-delete`* - Delete an entity import request and stored results

- `help`, `h` - Shows a list of commands or help for one command

2.5.6.3.1.302 Flags

Includes *global `dpsctl` options*.

Flag	Source	Description
`--source string`	both	Entity source: `bcm` or `nautobot` (required)
`--username string`	BCM	Authentication username (required when `--source=bcm`)
`--password string`	BCM	Authentication password (required when `--source=bcm`)
`--tls-insecure-ski`	BCM	Skip TLS certificate verification
`--locations string`	Nautobot	Comma-separated locations (UUIDs or names) to filter by
`--racks string`	Nautobot	Comma-separated racks (UUIDs or names) to filter by
`--device-types string`	Nautobot	Comma-separated device types (UUIDs or names) to filter by
`--request-string string`	Nautobot	Optional free-form URL param string; defined fields (`locations`, `racks`, `device-types`) take precedence over overlapping keys
`--help, -h`	both	Show help

When `--source=nautobot`, BCM flags (`--username`, `--password`, `--tls-insecure-skip-verify`) are ignored. When `--source=bcm`, Nautobot filter flags are ignored.

2.5.6.3.1.303 Examples

2.5.6.3.1.304 Generate Entities from BCM

```
$ dpsctl import entity \
  --source bcm \
  --username admin \
  --password secretpassword
{
  "request_id": "abc123-def456-ghi789"
}
```

The command returns a `request_id` that you use to retrieve the generated entities.

2.5.6.3.1.305 Retrieve Entities Using Request ID

After starting entity generation, use the request ID with `import entity-status`:

```
$ dpsctl import entity-status --request-id abc123-def456-ghi789
```

Refer to *`dpsctl import entity-status`* for full details.

2.5.6.3.1.306 Generate Entities with TLS Skip Verify

```
$ dpsctl import entity \
  --source bcm \
  --username admin \
  --password secretpassword \
  --tls-insecure-skip-verify
```

2.5.6.3.1.307 Generate Entities from Nautobot with Filters

```
$ dpsctl import entity \
  --source nautobot \
  --locations "dc1,dc2" \
  --racks "rack-a" \
  --device-types "server-type-1" \
  --request-string "status=active"
{
  "request_id": "abc123-def456-ghi789"
}
```

Configure the API connection and authentication for Nautobot imports through *Configure Server Authentication*.

2.5.6.3.1.308 Notes

- The `--source` flag is required. Supported values are `bcm` and `nautobot`.
- **BCM**: `--username` and `--password` are required. BCM credentials must have sufficient privileges to read cluster inventory.
- **Nautobot**: Uses the Topology Management gRPC service. No BCM credentials are needed. Filters (`--locations`, `--racks`, `--device-types`, `--request-string`) scope the import to a subset of Nautobot DCIM inventory.
- Entity generation runs asynchronously on the server. The command returns immediately with a request ID.
- Use *`dpsctl import entity-status`* with the returned request ID to poll for and retrieve the generated entities.
- Use *`dpsctl import entity-cancel`* to cancel an in-flight request, or *`dpsctl import entity-delete`* to delete stored request state and generated results.

2.5.6.3.1.309 entity

2.5.6.3.1.310 `dpsctl import entity` Usage Guide

Start entity generation from an external source. Returns a request ID that can be used with *`dpsctl import entity-status`* to retrieve the generated entities.

2.5.6.3.1.311 Usage

```
dpsctl import entity [command options]
```

2.5.6.3.1.312 Flags

Includes *global `dpsctl` options*.

Flag	Source	Description
`--source string`	both	Entity source: `bcm` or `nautobot` (required)
`--username string`	BCM	Authentication username (required when `--source=bcm`)
`--password string`	BCM	Authentication password (required when `--source=bcm`)
`--tls-insecure-ski`	BCM	Skip TLS certificate verification
`--locations string`	Nautobot	Comma-separated locations (UUIDs or names) to filter by
`--racks string`	Nautobot	Comma-separated racks (UUIDs or names) to filter by
`--device-types string`	Nautobot	Comma-separated device types (UUIDs or names) to filter by
`--request-string string`	Nautobot	Optional free-form URL param string; defined fields (`locations`, `racks`, `device-types`) take precedence over overlapping keys
`--help, -h`	both	Show help

When `--source=nautobot`, BCM flags (`--username`, `--password`, `--tls-insecure-skip-verify`) are ignored. When `--source=bcm`, Nautobot filter flags are ignored.

2.5.6.3.1.313 Examples

2.5.6.3.1.314 Generate Entities from BCM

```
$ dpsctl import entity \
  --source bcm \
  --username admin \
  --password secretpassword
{
  "request_id": "abc123-def456-ghi789"
}
```

The command returns a `request_id` that you use to retrieve the generated entities.

2.5.6.3.1.315 Retrieve Entities Using Request ID

After starting entity generation, use the request ID with `import entity-status`:

```
$ dpsctl import entity-status --request-id abc123-def456-ghi789
```

Refer to *`dpsctl import entity-status`* for full details.

2.5.6.3.1.316 Generate Entities with TLS Skip Verify

```
$ dpsctl import entity \
  --source bcm \
  --username admin \
  --password secretpassword \
  --tls-insecure-skip-verify
```

2.5.6.3.1.317 Generate Entities from Nautobot with Filters

```
$ dpsctl import entity \
  --source nautobot \
  --locations "dc1,dc2" \
  --racks "rack-a" \
  --device-types "server-type-1" \
  --request-string "status=active"
{
  "request_id": "abc123-def456-ghi789"
}
```

Configure the API connection and authentication for Nautobot imports through *Configure Server Authentication*.

2.5.6.3.1.318 Notes

- The `--source` flag is required. Supported values are `bcm` and `nautobot`.
- **BCM**: `--username` and `--password` are required. BCM credentials must have sufficient privileges to read cluster inventory.
- **Nautobot**: Uses the Topology Management gRPC service. No BCM credentials are needed. Filters (`--locations`, `--racks`, `--device-types`, `--request-string`) scope the import to a subset of Nautobot DCIM inventory.
- Entity generation runs asynchronously on the server. The command returns immediately with a request ID.
- Use *`dpsctl import entity-status`* with the returned request ID to poll for and retrieve the generated entities.
- Use *`dpsctl import entity-cancel`* to cancel an in-flight request, or *`dpsctl import entity-delete`* to remove stored request state and generated results.

2.5.6.3.1.319 entity-status

2.5.6.3.1.320 `dpsctl import entity-status` Usage Guide

Get the status and results of an entity import operation by request ID. The request ID is returned by *`dpsctl import entity`*.

`dpsctl import entity-status` waits for the import request to complete, then returns the generated entities.

2.5.6.3.1.321 Usage

```
dpsctl import entity-status --request-id <id>
```

2.5.6.3.1.322 Flags

Includes *global dpsctl options*.

```
   --request-id string  Request ID returned by 'import entity' command
   --help, -h           show help
```

2.5.6.3.1.323 Examples

2.5.6.3.1.324 Check Status and Retrieve Entities

```
$ dpsctl import entity-status --request-id abc123-def456-ghi789
[
  {
    "Type": "ComputerSystem",
    "Model": "DGX_H100",
    "Name": "node001",
    "Redfish": {
      "URL": "https://node001-bmc.example.com",
      "SecretName": "node001"
    }
  },
  {
    "Type": "ComputerSystem",
    "Model": "DGX_H100",
    "Name": "node002",
    "Redfish": {
      "URL": "https://node002-bmc.example.com",
      "SecretName": "node002"
    }
  }
]
```

2.5.6.3.1.325 Request Lifecycle

- **Generation timeout**: The server runs entity generation with a default timeout of 10 minutes (server maximum 30 minutes). Long-running requests may be marked failed after the timeout.
- **Request cleanup**: Request state and generated entities are removed after 30 minutes (TTL-based cleanup).
- **Completion behavior**: The command blocks until the import request is complete, then streams the generated entities.
- **Error handling**: Returns error details if the import operation failed or the request ID is invalid or not found.
- **Manual cleanup**: Use *dpsctl import entity-delete* to remove stored request state and generated results before TTL cleanup.

2.5.6.3.1.326 entity-cancel

2.5.6.3.1.327 dpsctl import entity-cancel Usage Guide

Cancel an in-flight entity import operation by request ID. The request ID is returned by *dpsctl import entity*.

Use this command when an entity generation request is still pending or running and you want the server to stop processing it.

2.5.6.3.1.328 Usage

```
dpsctl import entity-cancel --request-id <id>
```

2.5.6.3.1.329 Flags

Includes *global dpsctl options*.

```
   --request-id string  Request ID returned by 'import entity' command
   --help, -h           show help
```

2.5.6.3.1.330 Examples

2.5.6.3.1.331 Cancel an Entity Import Request

```
$ dpsctl import entity-cancel --request-id abc123-def456-ghi789
{
  "status": {
    "ok": true
  }
}
```

After cancellation, use *dpsctl import entity-delete* if you also want to remove the stored request record and any stored results.

2.5.6.3.1.332 Cancel vs Delete

- `entity-cancel` stops an entity generation request that is pending or running.
- `entity-cancel` does not remove the request record or stored results.
- `entity-delete` removes a stored request and any stored results.

2.5.6.3.1.333 Notes

- The `--request-id` flag is required.
- The request ID must be the value returned by *dpsctl import entity*.
- The server validates whether the request can be cancelled and returns an error if it cannot.

2.5.6.3.1.334 entity-delete

2.5.6.3.1.335 dpsctl import entity-delete Usage Guide

Delete a stored entity import request and any stored results by request ID. The request ID is returned by *dpsctl import entity*.

Use this command after a request has completed, failed, or been cancelled and you no longer need its stored status or generated entities.

2.5.6.3.1.336 Usage

```
dpsctl import entity-delete --request-id <id>
```

2.5.6.3.1.337 Flags

Includes *global dpsctl options*.

```
   --request-id string  Request ID returned by 'import entity' command
   --help, -h           show help
```

2.5.6.3.1.338 Examples

2.5.6.3.1.339 Delete a Stored Entity Import Request

```
$ dpsctl import entity-delete --request-id abc123-def456-ghi789
{
  "status": {
    "ok": true
  }
}
```

If you need to stop an in-flight request before deleting its stored state, cancel it first:

```
$ dpsctl import entity-cancel --request-id abc123-def456-ghi789
{
  "status": {
    "ok": true
  }
}

$ dpsctl import entity-delete --request-id abc123-def456-ghi789
{
  "status": {
    "ok": true
  }
}
```

2.5.6.3.1.340 Cancel vs Delete

- `entity-cancel` stops an entity generation request that is pending or running.
- `entity-delete` removes the stored request record and any stored generated entities.

- The server validates whether the request can be deleted and returns an error if it cannot.

2.5.6.3.1.341 Notes

- The `--request-id` flag is required.
- The request ID must be the value returned by *`dpsctl import entity`*.
- Use *`dpsctl import entity-status`* before deleting if you need to retrieve generated entities.
- Returns an error if the request ID is invalid, not found, or cannot be deleted.

2.5.6.3.1.342 dpsctl verify

2.5.6.3.1.343 dpsctl verify Usage Guide

Verify DPS deployment status and component connectivity.

2.5.6.3.1.344 Usage

```
dpsctl verify [options]
```

2.5.6.3.1.345 Commands

- *`bmc-health`* - Run or inspect deep BMC pre-requisite health checks across the topology
- `help`, `h` - Shows a list of commands or help for one command

2.5.6.3.1.346 Aliases

Command	Aliases
bmc-health	bmc

2.5.6.3.1.347 Flags

Includes *global `dpsctl` options*.

```
   --dps-server     Check only DPS Server accessibility (default: check all
↪components)
   --database       Check only database accessibility (default: check all
↪components)
   --auth           Check only authentication service (default: check all
↪components)
   --ui             Check only UI accessibility (default: check all components)
   --bcm            Check only BCM connectivity (default: check all components)
   --help, -h       show help
```

2.5.6.3.1.348 Examples

2.5.6.3.1.349 Verify All Components

```
$ dpsctl verify
{
  "dps_server": {
    "status": "ok",
    "message": "DPS server is accessible"
  },
  "database": {
    "status": "ok",
    "message": "Database connection successful"
  },
  "auth": {
    "status": "ok",
    "message": "Authentication service is working"
  },
  "ui": {
    "status": "warning",
    "message": "UI is accessible but some features may be limited"
  },
  "bcm": {
    "status": "error",
    "message": "BCM connection failed: connection timeout"
  }
}
```

2.5.6.3.1.350 Verify Specific Component

```
$ dpsctl verify --dps-server
{
  "dps_server": {
    "status": "ok",
    "message": "DPS server is accessible"
  }
}
```

2.5.6.3.1.351 Verify Multiple Components

```
$ dpsctl verify --dps-server --database
{
  "dps_server": {
    "status": "ok",
    "message": "DPS server is accessible"
  },
  "database": {
    "status": "ok",
    "message": "Database connection successful"
  }
}
```

2.5.6.3.1.352 Notes

- If no specific component flags are provided, all components are checked by default
- Useful for deployment validation and troubleshooting
- Each component check is independent and failures are reported separately
- Status values can be: `ok`, `warning`, `error`

2.5.6.3.1.353 bmc-health

2.5.6.3.1.354 `dpsctl verify bmc-health` Usage Guide

Run and inspect deep BMC pre-requisite health checks across a topology.

DPS 0.9 requires the topology to be active before you run `dpsctl verify bmc-health start`. Activate the intended topology through *Managing Topologies*, then confirm it appears in `dpsctl topology list --active` before starting the check.

Without an active topology, starting a check can return `INVALID_OPERATION` with `no agent configuration available`. For the complete operator procedure, refer to *Run a BMC Health Check*.

The `bmc-health` command also has the alias `bmc`:

```
dpsctl verify bmc start --topology my-topology
```

2.5.6.3.1.355 Overview

`dpsctl verify bmc-health` probes the BMCs in a topology. Device plugins with deep BMC health support collect their supported firmware, power, EDPp, telemetry, and WPPS evidence. They also compare in-band and out-of-band power-limit values where supported. If the selected device plugin does not support the deep BMC health check, DPS checks only BMC reachability and firmware. The server screens available results against built-in latency and WPPS criteria and the default or requested EDPp floor.

For a quick surface-level reachability check, use *`dpsctl check connection`* instead.

The BMC health probe can take many minutes on large clusters, especially when deep telemetry and write-cycle checks are enabled. It is asynchronous:

- `start` submits the check and returns a `task_id`.
- `status <task-id>` shows live progress.
- `report <task-id>` fetches the final cluster-level result.

Only one BMC health check may run at a time on the server. This gate is server-wide, not per topology or per node set. If a second `start` is attempted while another run is active, it is rejected. The public error does not expose the in-flight `task_id`. Reattach only when you retained the ID from the original `start`; otherwise, coordinate with the operator who started the task or wait for it to finish.

2.5.6.3.1.356 Usage

```
dpsctl verify bmc-health start --topology <topology-name> [options]
dpsctl verify bmc-health status <task-id>
dpsctl verify bmc-health report <task-id> [options]
```

2.5.6.3.1.357 Start

Submit a BMC health check and print the task handle.

```
dpsctl verify bmc-health start --topology <topology-name>
```

2.5.6.3.1.358 Flags

Includes *global `dpsctl` options*.

```
   --topology value                 (required) topology name to probe
   --nodes value [ --nodes value ]  (optional) comma separated names of nodes
↪to probe; defaults to all nodes in topology
   --samples-per-telemetry value    Number of telemetry reads per node
↪(default: 0; 0 uses server default of 10; maximum: 2500)
   --telemetry-interval value       Sleep between consecutive telemetry reads
↪on each node (default: 500ms; server-side cap: 60s)
   --write-resolution-timeout value Per-PATCH readback window for the write-
↪cycle probe (default: 1s; server-side cap: 60s)
   --per-node-timeout value         Complete per-node task timeout (default:
↪0; 0 uses server default of 5m; allowed: 5m..48h)
   --concurrency value              Per-cluster fan-out concurrency (default:
↪0; 0 uses server default of 16; maximum: 1024)
   --skip-writes                    Skip power-limit write/read-back/restore
↪and WPPS remediation (read-only mode)
   --force-writes                   Request power-limit write/read-back/
↪restore and WPPS remediation from a supporting device plug-in
   --expected-edpp-pct value        Expected steady-state EDPp current value,
↪in percent (default: 0; 0 uses server default of 100; allowed: 1..200)
   --wait                           Poll status until the task completes,
↪then print the final report
   --poll-interval value            Status polling interval for --wait
↪(default: 5s; ignored without --wait)
   --summary-only                   With --wait, strip per-node detail from
↪the printed report and keep only Status, ClusterSummary, and Issues
↪(ignored without --wait)
   --prs-bmc-latency-check          With --wait, run the BMC latency
↪acceptance check for Power Reservation Steering (PRS) after BMC health
   --prs-metrics-url value          Optional PRS controller Prometheus
↪metrics URL for the PRS latency check
   --prs-metrics-step value         PRS histogram operation step (default:
↪cluster_apply)
   --actuation-p99-max value        Inclusive PRS p99 latency ceiling
↪(default: 1s)
   --prs-latency-window value       PRS latency observation window (default:
↪5m)
   --prs-latency-min-samples value  Minimum PRS histogram sample count
↪(default: 1200)
   --prs-interval-override value    Temporarily set and restore the PRS
↪scheduler interval during the check
   --help, -h                       show help
```

2.5.6.3.1.359 Start Behavior

Without `--wait`, `start` returns immediately with a task handle containing the `task_id`, task name, initial status, and start time.

With `--wait`, `start` blocks after submission. It polls status, writes one-line progress updates to stderr, and prints the final report to stdout when the task finishes. Progress updates use the format:

```
bmc-health <task-id>: <status> (<nodes-completed>/<nodes-total> nodes)
```

If the final report contains a failed `status.ok`, the command exits with status code 1.

On `start`, `--poll-interval` and `--summary-only` are only used by the `--wait` shortcut. Without `--wait`, they are silently ignored because the command returns only the task handle.

`--skip-writes` and `--force-writes` are mutually exclusive. If neither flag is provided, the request leaves write behavior unset and the current server default is read-only. If `--skip-writes` is provided, power-limit write cycles and WPPS remediation are disabled. If `--force-writes` is provided, a device plugin that supports write-enabled deep checks can run power-limit write/readback/restore cycles and can disable active WPPS profiles. The flag does not override a plugin's safety policy.

`--prs-bmc-latency-check` requires `--wait`. The optional `--prs-interval-override` changes the PRS scheduler interval during the task and attempts to restore the original interval afterward. When the PRS latency check is enabled, the CLI prints the BMC report followed by the PRS artifact, and its final exit decision is based on the PRS artifact. Inspect the BMC report's `status.ok` value separately.

2.5.6.3.1.360 Status

Show live status and progress for a previously started check.

```
dpsctl verify bmc-health status <task-id>
```

2.5.6.3.1.361 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.362 Status Output

`status` prints the task ID, task name, task-manager status string, timestamps, diagnostic message, progress counters, completion flag, and terminal error message when present.

The status string can be `Init`, `Started`, `Done`, or `Failed`. The progress counters report how many nodes have completed out of the total in-scope nodes.

Server-side status snapshots are coalesced for about 500ms, so very fast polling does not provide fresher task state.

2.5.6.3.1.363 Report

Fetch the final report for a BMC health check.

```
dpsctl verify bmc-health report <task-id>
```

2.5.6.3.1.364 Flags

Includes *global `dpsctl` options*.

```
  --wait                Block until the task finishes, then print the report
  --poll-interval value  Status polling interval for --wait (default: 5s;
↪ignored without --wait)
  --summary-only        Strip per-node NodeReport detail server-side and
↪return only Status, ClusterSummary, and Issues
  --help, -h            show help
```

2.5.6.3.1.365 Report Behavior

Without `--wait`, `report` performs a single lookup. If the task is still running, it prints a structured response with `done=false` and no report payload. Run `status` and retry later, or use `report --wait`.

With `--wait`, `report` polls status until the task reaches a terminal state, then fetches and prints the final report. Progress updates are written to stderr so stdout remains clean for `--output json` or `--output yaml`.

When `--summary-only` is set, the server omits per-node `NodeReport` entries from the returned report and keeps only cluster-level `Status`, `ClusterSummary`, and `Issues`. The underlying task result is not modified; until the task expires, a later `report` call without `--summary-only` can still retrieve the full report.

If the report response contains an `error_message`, the CLI returns an error instead of printing an empty report.

If the final report contains a failed `status.ok`, the command exits with status code 1.

2.5.6.3.1.366 Polling and Re-Attach

The default `--poll-interval` for `--wait` is `5s`. Non-positive values fall back to `5s`. Values below the client-side minimum of `1s` are clamped to `1s` and the CLI prints a stderr notice explaining the clamp.

Canceling a local `--wait` command, such as with Ctrl-C, stops only local polling. It does not cancel the server-side BMC health check. Re-attach with:

```
dpsctl verify bmc-health status <task-id>
dpsctl verify bmc-health report <task-id> --wait
```

Completed task state remains queryable for the server `--zombie-lifetime` window, which defaults to `10m`. After that window expires, `status` and `report` can no longer re-attach to the completed task.

There is no `dpsctl verify bmc-health cancel` subcommand.

2.5.6.3.1.367 Examples

2.5.6.3.1.368 Start a check and poll later

```
$ dpsctl verify bmc-health start --topology rack-a
{
```

(continues on next page)

(continued from previous page)

```
  "task_id": "task-abc123",
  "name": "bmc_health_check",
  "status": "Started",
  "started_at": "2026-05-21T14:30:00Z"
}
```

2.5.6.3.1.369 Check progress

```
$ dpsctl verify bmc-health status task-abc123
{
  "task_id": "task-abc123",
  "name": "bmc_health_check",
  "status": "Started",
  "started_at": "2026-05-21T14:30:00Z",
  "progress": {
    "nodes_total": 128,
    "nodes_completed": 47
  },
  "done": false
}
```

2.5.6.3.1.370 Wait for a summary report

```
dpsctl verify bmc-health report task-abc123 --wait --summary-only
```

2.5.6.3.1.371 Run a scoped, read-only check

```
dpsctl verify bmc-health start --topology rack-a --nodes node001,node002 --
↪skip-writes --wait
```

2.5.6.3.1.372 Run approved write-enabled checks with slower telemetry sampling

```
dpsctl verify bmc-health start --topology rack-a --force-writes --telemetry-
↪interval 2s --write-resolution-timeout 5s --wait
```

2.5.6.3.1.373 dpsctl check Usage Guide

2.5.6.3.1.374 Overview

`dpsctl check` includes DPS diagnostic tools.

2.5.6.3.1.375 Usage

```
dpsctl check [command [command options]]
```

2.5.6.3.1.376 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.377 Commands

- *`metrics`* - Retrieve node metrics
- *`connection`* - Perform BMC connection check on nodes
- *`status`* - Get node status
- `help`, h - Shows a list of commands or help for one command

2.5.6.3.1.378 Aliases

Command	Aliases
connection	conn

2.5.6.3.1.379 status

2.5.6.3.1.380 dpsctl check status Usage Guide

Get the status of specified nodes.

2.5.6.3.1.381 Usage

```
dpsctl check status
```

2.5.6.3.1.382 Flags

Includes *global `dpsctl` options*.

```
   --nodes value       Node names to get status for
   --help, -h          show help
```

2.5.6.3.1.383 Examples

2.5.6.3.1.384 Check Single Node Status

```
$ dpsctl check status --nodes node001
{
  "node_statuses": {
    "node001": {
      "last_successful_access_time": "2024-01-15T10:30:00Z",
      "current_state": {
        "resource_group": "ml-training-job",
```

(continues on next page)

(continued from previous page)

```
        "allocated_policy_name": "ComputerSystemLimits.v1"
      }
    }
  }
}
```

2.5.6.3.1.385 Check Multiple Node Status

```
$ dpsctl check status --nodes node001,node002,node003
{
  "node_statuses": {
    "node001": {
      "last_successful_access_time": "2024-01-15T10:30:00Z",
      "current_state": {
        "resource_group": "ml-training-job",
        "allocated_policy_name": "ComputerSystemLimits.v1"
      }
    },
    "node002": {
      "last_successful_access_time": "2024-01-15T09:45:00Z",
      "inaccessible_since": "2024-01-15T09:45:00Z",
      "last_access_error": "BMC connection failed",
      "current_state": {
        "resource_group": "",
        "allocated_policy_name": "ComputerSystemLimits.v1"
      }
    },
    "node003": {
      "last_successful_access_time": "2024-01-15T10:29:30Z",
      "current_state": {
        "resource_group": "dev-testing",
        "allocated_policy_name": "ComputerSystemLimits.v1"
      }
    }
  }
}
```

2.5.6.3.1.386 Check Status with Issues

```
$ dpsctl check status --nodes node004
{
  "node_statuses": {
    "node004": {
      "last_successful_access_time": "2024-01-15T08:15:00Z",
      "inaccessible_since": "2024-01-15T08:15:00Z",
      "last_access_error": "BMC connection timeout",
      "current_state": {
        "resource_group": "",
        "allocated_policy_name": ""
      }
```

(continues on next page)

(continued from previous page)

```
    }
  }
}
```

2.5.6.3.1.387 Check Non-Existent Node

```
$ dpsctl check status --nodes non-existent-node
{
  "node_statuses": {}
}
```

2.5.6.3.1.388 Notes

- Node status reflects real-time connectivity and power management state
- `last_successful_access_time` shows when the node was last accessible
- `inaccessible_since` is included when the node has become inaccessible
- `last_access_error` is included when the node has a recent access error
- `current_state` contains the resource group and allocated policy identity
- `allocated_policy_name` identifies the allocated policy object or type, such as `ComputerSystemLimits.v1`; it is not the selected policy bundle. Inspect `policy_bundle` with `dpsctl topology list` or `dpsctl resource-group list`
- Empty `node_statuses` indicates no nodes were found or all nodes are inaccessible

2.5.6.3.1.389 metrics

2.5.6.3.1.390 dpsctl check metrics Usage Guide

Retrieve node power usage & telemetry metrics.

2.5.6.3.1.391 Usage

```
dpsctl check metrics
```

2.5.6.3.1.392 Flags

Includes *global dpsctl options*.

```
   --nodes value      nodes to retrieve metrics for
   --effective        report power-limit fields as currently enforced by the
↪device; adds one Redfish GET per device
   --help, -h         show help
```

2.5.6.3.1.393 Examples

```
$ dpsctl check metrics --nodes node001
{
  "nodes": [
    {
      "name": "node001",
      "num_gpus": 8,
      "num_cpus": 2,
      "num_memory_units": 32,
      "power_usage": 2074.0,
      "gpus": [
        {
          "id": "0",
          "power_usage": 64.0,
          "energy_usage": 195.0,
          "energy_usage_unit": "Joules",
          "set_limit": 700.0,
          "max_limit": 700.0,
          "min_limit": 200.0,
          "temperature_celsius": 45.0
        },
        {
          "id": "1",
          "power_usage": 66.0,
          "energy_usage": 195.0,
          "energy_usage_unit": "Joules",
          "set_limit": 700.0,
          "max_limit": 700.0,
          "min_limit": 200.0,
          "temperature_celsius": 47.0
        }
      ],
      "cpus": [
        {
          "id": "0",
          "power_usage": 195.0,
          "energy_usage": 195.0,
          "energy_usage_unit": "Joules",
          "set_limit": 400.0,
          "max_limit": 350.0,
          "min_limit": 209.0,
          "temperature_celsius": 65.0
        },
        {
          "id": "1",
          "power_usage": 195.0,
          "energy_usage": 195.0,
          "energy_usage_unit": "Joules",
          "set_limit": 400.0,
          "max_limit": 350.0,
          "min_limit": 209.0,
          "temperature_celsius": 63.0
```

(continues on next page)

(continued from previous page)


```
        }
      ],
      "memory": [
        {
          "id": "0",
          "power_usage": 28.0,
          "energy_usage": 28.0,
          "energy_usage_unit": "Joules",
          "set_limit": 300.0,
          "max_limit": 122.0,
          "min_limit": 0.0,
          "temperature_celsius": 35.0
        },
        {
          "id": "1",
          "power_usage": 37.0,
          "energy_usage": 37.0,
          "energy_usage_unit": "Joules",
          "set_limit": 300.0,
          "max_limit": 122.0,
          "min_limit": 0.0,
          "temperature_celsius": 36.0
        }
      ]
    }
  ]
}
```


2.5.6.3.1.394 Notes

- Metrics are returned in a standardized format for all node types
- At least one node must be specified with `--nodes`
- `power_usage` shows current power consumption in watts
- `energy_usage` shows cumulative energy consumption since last reset
- `set_limit`, `max_limit`, and `min_limit` show power limit configurations
- Use `--effective` to read enforced power-limit fields directly from BMC EnvironmentMetrics endpoints; this adds one Redfish GET per device
- `temperature_celsius` shows component temperature
- GPU IDs are typically 0-based within each node
- CPU and memory components are also 0-based

2.5.6.3.1.395 connection

2.5.6.3.1.396 dpsctl check connection Usage Guide

Perform BMC connection check on configured nodes.

2.5.6.3.1.397 Usage

```
dpsctl check connection --topology <topology-name>
```

2.5.6.3.1.398 Prerequisite in DPS 0.9

Activate a newly imported topology through *Managing Topologies* before you check its BMC connections. The check reaches each node through its assigned DPS Agent, and activation, not import, creates that routing. Confirm that the topology appears in `dpsctl topology list --active` before you run the check.

Without routing from a prior activation, a scoped check can return `INVALID_OPERATION` with `no agent configuration available`.

2.5.6.3.1.399 Flags

Includes *global `dpsctl` options*.

```
  --topology string                  (optional) topology name. If provided
↪without --nodes, all nodes in that topology are checked.
  --nodes string [ --nodes string ]  (optional) comma separated names of
↪nodes to perform check on. If provided, only these nodes are checked.
  --help, -h                         show help
```

2.5.6.3.1.400 Behavior

The `dpsctl check connection` command determines which nodes to ping based on the provided flags:

2.5.6.3.1.401 Node Selection Rules:

1. **If `--nodes` is provided**: Only the specified nodes are checked.
2. **If no `--nodes` but `--topology` is provided**: All nodes in the specified topology are checked.
3. **If neither `--nodes` nor `--topology` is provided**: No BMCs are checked. The server returns an empty health response. Use an explicit scope to verify BMC connectivity.

2.5.6.3.1.402 Inactive Topologies

The server does not require the topology to be active, but it can reach a node only through Agent routing that a successful activation created. DPS keeps that routing after deactivation until the next successful activation or a server restart. As a result, a check against an inactive topology can reach only nodes from the most recently activated topology. The check uses the Agent assignments and node settings from that activation. With `--topology`, checks of nodes outside that activation fail individually. With `--nodes`, naming any node outside that activation fails the whole request with `INVALID_OPERATION`. Enabling `dps.agent.allowHandshakeFallback` does not remove the need for a prior successful activation. Do not use such a check to confirm connectivity for a new topology.

2.5.6.3.1.403 Examples

The examples assume `my-topology` is active and the named nodes belong to it.

2.5.6.3.1.404 Check all nodes in a specific topology

```
$ dpsctl check connection --topology my-topology
{
  "total_nodes": 4,
  "success_nodes": 4,
  "failure_nodes": []
}
```

2.5.6.3.1.405 Check specific nodes

```
$ dpsctl check connection --nodes node002,node003
{
  "total_nodes": 2,
  "success_nodes": 2,
  "failure_nodes": []
}
```

2.5.6.3.1.406 Check specific nodes (topology flag ignored when nodes are specified)

```
$ dpsctl check connection --topology my-topology --nodes node002,node003
{
  "total_nodes": 2,
  "success_nodes": 2,
  "failure_nodes": []
}
```

2.5.6.3.1.407 Check with connection failures

```
$ dpsctl check connection --nodes node004
{
  "total_nodes": 1,
  "success_nodes": 0,
  "failure_nodes": [
    {
      "node_name": "node004",
      "error_msg": "BMC connection timeout"
    }
  ]
}
```

2.5.6.3.1.408 dpsctl nvgrid

2.5.6.3.1.409 dpsctl nvgrid Usage Guide

2.5.6.3.1.410 Overview

`dpsctl nvgrid` provides DPS NvGrid power management operations for grid integration and load target scheduling.

2.5.6.3.1.411 Usage

```
dpsctl nvgrid [command [command options]]
```

2.5.6.3.1.412 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.413 Commands

- `set-load-target` - Set a load target for power feeds
- `get-schedule` - Get load schedule for specified feeds and time interval
- `get-current` - Get currently active load targets for all feeds
- `get-metadata` - Get metadata for all power feeds
- `get-status` - Get current system status for all feeds
- `register-webhook` - Register a webhook for power events
- `unregister-webhook` - Unregister a previously registered webhook
- `help`, `h` - Shows a list of commands or help for one command

2.5.6.3.1.414 Examples

2.5.6.3.1.415 Query Feed Information

```
# Get feed metadata
dpsctl nvgrid get-metadata

# Get current status
dpsctl nvgrid get-status

# Get active targets
dpsctl nvgrid get-current
```

2.5.6.3.1.416 Schedule Load Targets

```
# Schedule 6MW target for 1 hour
dpsctl nvgrid set-load-target \
```

(continues on next page)

(continued from previous page)

```
  --value=6 \
  --unit=megawatt \
  --start-time="2025-10-24T15:00:00Z" \
  --end-time="2025-10-24T16:00:00Z" \
  --best-effort=true

# Schedule with specific correlation ID
dpsctl nvgrid set-load-target \
  --value=5 \
  --unit=megawatt \
  --start-time="2025-10-24T17:00:00Z" \
  --end-time="2025-10-24T18:00:00Z" \
  --correlation-id="curtailment-001" \
  --best-effort=true

# Target specific feeds
dpsctl nvgrid set-load-target \
  --value=4.5 \
  --unit=megawatt \
  --start-time="2025-10-24T15:00:00Z" \
  --end-time="2025-10-24T15:30:00Z" \
  --feed-tags="feed-a1,feed-a2"

# Reset to default constraint (method 1: using --default)
dpsctl nvgrid set-load-target \
  --default \
  --start-time="2025-10-24T18:00:00Z" \
  --end-time="2025-10-24T20:00:00Z"

# Reset to default constraint (method 2: using --value=0)
dpsctl nvgrid set-load-target \
  --value=0 \
  --unit=watt \
  --start-time="2025-10-24T18:00:00Z" \
  --end-time="2025-10-24T20:00:00Z"
```

2.5.6.3.1.417 Query Schedules

```
# Query schedule for next 4 hours
dpsctl nvgrid get-schedule \
  --start-time="2025-10-24T14:00:00Z" \
  --end-time="2025-10-24T18:00:00Z"

# Query specific feeds
dpsctl nvgrid get-schedule \
  --start-time="2025-10-24T14:00:00Z" \
  --end-time="2025-10-24T18:00:00Z" \
  --feed-tags="feed-a1"
```

2.5.6.3.1.418 Webhook Management

```
# Register webhook for all events
dpsctl nvgrid register-webhook \
  --url="https://grid-solution.example.com/webhook"

# Register for specific events
dpsctl nvgrid register-webhook \
  --url="https://grid-solution.example.com/webhook" \
  --events="start-ramp-down,end-ramp-down"

# Unregister webhook
dpsctl nvgrid unregister-webhook \
  --url="https://grid-solution.example.com/webhook"
```

2.5.6.3.1.419 Common Flags

2.5.6.3.1.420 set-load-target

Flag	Type	Description
`--value`	float	Target load value (mutually exclusive with `--default`)
`--unit`	string	Unit: `watt`, `w`, `kilowatt`, `kw`, `megawatt`, or `mw` (required with `--value`)
`--default`	bool	Reset to default (zero) power constraint (mutually exclusive with `--value` and `--unit`)
`--start-time`	timestamp	Start time in RFC3339 format (e.g., 2025-10-24T15:00:00Z)
`--end-time`	timestamp	End time in RFC3339 format (e.g., 2025-10-24T16:00:00Z)
`--feed-tags`	string	Comma-separated feed tags (empty = all feeds)
`--correlation-`	string	Tracking ID for this load target (optional)
`--best-effort`	bool	Use best-effort strategy (default: true)

Note: Either `--default` or `--value` must be specified. When using `--value`, `--unit` is required. Setting `--value=0` with any unit is equivalent to using `--default` (both set the default/zero constraint).

2.5.6.3.1.421 get-schedule

Flag	Type	Description
`--start-time`	timestamp	Schedule query start time (ISO 8601)
`--end-time`	timestamp	Schedule query end time (ISO 8601)
`--feed-tags`	string	Comma-separated feed tags (empty = all)

2.5.6.3.1.422 register-webhook

Flag	Type	Description
`--url`	string	Webhook URL (must be whitelisted)
`--events`	string	Comma-separated events (empty = all): `start-ramp-up`, `end-ramp-up`, `start-ramp-down`, `end-ramp-down`

2.5.6.3.1.423 unregister-webhook

Flag	Type	Description
`--url`	string	Webhook URL to unregister

2.5.6.3.1.424 Further Reading

- *NvGrid Concepts* - Understanding NvGrid
- *Managing NvGrid* - Operations guide
- *Grid Integrator Tutorial* - Complete integration with code examples

2.5.6.3.1.425 dpsctl task

2.5.6.3.1.426 dpsctl task Usage Guide

2.5.6.3.1.427 Overview

`dpsctl task` provides operations for monitoring async tasks running in DPS. These tasks include background operations such as topology activation, topology deactivation, resource group activation, and resource group deactivation.

2.5.6.3.1.428 Usage

```
dpsctl task [command [command options]]
```

2.5.6.3.1.429 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.430 Commands

- *`list`* - List ongoing async tasks
- `help`, h - Shows a list of commands or help for one command

2.5.6.3.1.431 list

2.5.6.3.1.432 `dpsctl task list` Usage Guide

List ongoing async tasks running in DPS server.

2.5.6.3.1.433 Overview

The `task list` command displays all currently running asynchronous tasks in the system. This is useful for monitoring long-running operations such as topology activation/deactivation or resource group activation/deactivation.

Each task includes:

- **id**: Unique identifier for the task
- **name**: Task type (e.g., `ResourceGroupActivation`, `TopologyActivation`)
- **status**: Current status (`running`, `pending`, etc.)
- **started_at**: When the task started
- **completed_at**: When the task completed (if finished)
- **parent_id**: Parent task ID for subtasks

2.5.6.3.1.434 Usage

```
dpsctl task list
```

2.5.6.3.1.435 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.436 Examples

2.5.6.3.1.437 List all running tasks

```
$ dpsctl task list
[
  {
    "id": "task-abc123",
    "name": "ResourceGroupActivation",
    "status": "running",
    "started_at": "2025-01-22T10:30:00Z",
    "parent_id": ""
  },
  {
    "id": "task-def456",
    "name": "NodeActivation",
    "status": "running",
    "started_at": "2025-01-22T10:30:01Z",
    "parent_id": "task-abc123"
```

(continues on next page)

(continued from previous page)

```
  }
]
```

2.5.6.3.1.438 No running tasks

When there are no ongoing tasks, the command returns an empty list:

```
$ dpsctl task list
[]
```

2.5.6.3.1.439 Use Cases

- **Monitoring activation progress**: Check the status of topology or resource group activation operations
- **Debugging**: Identify stuck or long-running tasks
- **Automation**: Wait for tasks to complete before proceeding with subsequent operations

2.5.6.3.1.440 dpsctl agent

2.5.6.3.1.441 Overview

`dpsctl agent` provides read-only information about configured and connected DPS Agents.

2.5.6.3.1.442 Usage

```
dpsctl agent [command [command options]]
```

2.5.6.3.1.443 Flags

Includes *global `dpsctl` options*.

```
   --help, -h  show help
```

2.5.6.3.1.444 Commands

- *`list`* - List configured and connected agents

2.5.6.3.1.445 list

2.5.6.3.1.446 `dpsctl agent list` Usage Guide

List the current in-memory inventory of configured Agents and live Agent sessions.

2.5.6.3.1.447 Usage

```
dpsctl agent list
```

2.5.6.3.1.448 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.449 Output

The command returns a JSON array by default. Each entry can contain the following fields:

Field	Description
`agent_id`	Agent identifier from topology assignments or live handshakes.
`configured`	`true` when the active topology assigns devices to the Agent.
`connected`	`true` when the Agent has at least one active connection.
`assigned_devices`	Device instance IDs assigned by the active topology.
`sessions`	Active connections, including version, capabilities, and reported devices.

The default JSON output omits fields whose values are false or empty.

When no Agents are configured or connected, the command returns an empty array.

2.5.6.3.1.450 dpsctl validate

2.5.6.3.1.451 `dpsctl validate` Usage Guide

2.5.6.3.1.452 Overview

`dpsctl validate` includes DPS validation commands that perform dry-run analysis without persisting state.

2.5.6.3.1.453 Usage

```
dpsctl validate [command [command options]]
```

2.5.6.3.1.454 Flags

Includes *global `dpsctl` options*.

```
  --help, -h  show help
```

2.5.6.3.1.455 Commands

- *`allocation`* - Validate policy allocation for devices (dry-run power allocation analysis)
- `help`, h - Shows a list of commands or help for one command

2.5.6.3.1.456 Aliases

Command	Aliases
validate	val

2.5.6.3.1.457 allocation

2.5.6.3.1.458 dpsctl validate allocation Usage Guide

Validate policy bundle allocation for devices without applying changes.

2.5.6.3.1.459 Usage

```
dpsctl validate allocation [command options]
```

You can also use the `validate` alias:

```
dpsctl val allocation [command options]
```

2.5.6.3.1.460 Flags

Includes *global `dpsctl` options*.

```
   --devices string [ --devices string ]  device names (comma-separated)
   --policy-bundle string                 policy bundle name to validate
   --strict-policy                        if the requested policy bundle is
↪not feasible, fail instead of retrying at the effective policy floor
↪(default: false)
   --help, -h                             show help
```

2.5.6.3.1.461 Behavior

The `dpsctl validate allocation` command performs a dry-run power allocation analysis for the requested devices and policy bundle. It reports whether the allocation would succeed and any constraints returned by DPS.

No allocation state is persisted. Use this command to test a policy bundle before selecting it for a resource group or topology.

2.5.6.3.1.462 Required Inputs

1. `--devices` must include at least one device name. Multiple devices can be provided as a comma-separated list.
2. `--policy-bundle` must specify the policy bundle name to validate against the devices.

2.5.6.3.1.463 Name Validation

The policy bundle name and each device name are validated before the request is sent. Invalid names fail fast and return a validation error for the corresponding flag.

2.5.6.3.1.464 Strict Policy

By default, validation can report the result after retrying an infeasible bundle at its effective policy floor. Use `--strict-policy` to disable that retry. If the requested bundle is not feasible, the command still prints a validation response, but `allocation_would_succeed` is `false` with a diagnostic message.

2.5.6.3.1.465 Examples

2.5.6.3.1.466 Validate allocation for one device

```
$ dpsctl validate allocation --devices node-a --policy-bundle MaxP
{
  "status": {
    "ok": true,
    "diag_msg": "would activate successfully"
  },
  "allocation_would_succeed": true
}
```

2.5.6.3.1.467 Validate allocation for multiple devices

```
$ dpsctl validate allocation --devices node-a,node-b --policy-bundle MaxP
{
  "status": {
    "ok": true,
    "diag_msg": "would activate successfully"
  },
  "allocation_would_succeed": true
}
```

2.5.6.3.1.468 Validate allocation with strict policy matching

```
$ dpsctl validate allocation --devices node-a,node-b --policy-bundle MaxP --
↪strict-policy
{
  "status": {
    "ok": true,
    "diag_msg": "would activate successfully"
  },
  "allocation_would_succeed": true
}
```

2.5.6.3.1.469 Use the validate alias

```
$ dpsctl val allocation --devices node-a,node-b --policy-bundle MaxP --strict-
↪policy
{
  "status": {
    "ok": true,
    "diag_msg": "would activate successfully"
  },
  "allocation_would_succeed": true
}
```

2.5.6.3.1.470 Notes

- This command is a dry run and does not persist allocation, policy, resource group, or topology changes
- `--devices` and `--policy-bundle` are required
- Device names are sent in the order provided
- Device and policy bundle names must pass DPS name validation
- Use `--strict-policy` when an effective-policy-floor retry is not acceptable
- Output includes `allocation_would_succeed` and can include policy-selection, power-reclamation, and per-device status details
- Output follows the selected global output format, such as `json` or `yaml`

Chapter 3. DPS SDK and Simulator

The DPS Software Development Kit (SDK) is a Kubernetes-based environment for running DPS without physical data center infrastructure. Use it to evaluate DPS, develop against the API, and build integrations.

3.1. Evaluate DPS Without Hardware

In its Hardware Emulation and Workload-Aware profiles, the simulator emulates 144 DGX GB300 compute nodes across eight racks, together with the complete power distribution hierarchy, Redfish BMC endpoints, Grafana dashboards, and automated simulation playbooks. It serves the full DPS API, so integrations you build against the simulator target the same interface as real hardware.

The Pilot profile provides a dedicated four-rack, 72-node GB300 environment. Rehearse the baseline and managed workflows with simulated power traces and compare their GPU power in Grafana. Follow *Rehearse the MaxLPS Power Pilot With the SDK Simulator* for the full procedure. Inference performance and physical power compliance require a hardware pilot.

It is a good fit for:

- Learning DPS concepts and workflows
- Proof-of-concept demonstrations before committing to a deployment
- Developing and testing custom integrations
- Automated testing of power management scenarios in CI

Start here: the *SDK Simulator User Guide* covers the four deployment profiles, installation, and the simulation playbooks.

The *SDK Simulator Topology* guide explains the general eight-rack topology used by Hardware Emulation and Workload-Aware profiles.

3.2. Build integrations

The following guides cover extending DPS rather than evaluating it. They assume you already have the simulator running.

- ***Administrator Guides*** — deployment architecture, operations, and the partner integration guides for grid integrators, optimization developers, and workload schedulers
- ***Developer Guides*** — building custom integrations and extensions

The *gRPC API reference* documents the `nvidia.dcpower.v1` service that all of these build against.

3.3. Deploying against real hardware

When you are ready to move beyond the simulator, refer to *Deploying DPS*.

3.3.1. SDK Simulator User Guide

The DPS SDK is a Kubernetes-based development environment for testing power management solutions without physical hardware. It ships with four deployment profiles so you can choose a starting point for your goals.

3.3.1.1 Deployment Profiles

- **Default** - Blank DPS environment with no topology or simulators running. Recommended starting point for hands-on `dpsctl` exercises and API exploration.
- **Hardware Emulation** - 144 emulated DGX GB300 nodes across 8 racks with pseudorandom hardware responses. Best for general API development, integration validation, and automated playbooks.
- **Workload-Aware** - Same 144-node GB300 topology, but the BMC simulator calls a pluggable workload model over gRPC to return realistic GPU power traces. Best for power-aware analysis and workload optimization studies. Refer to *Workload-Aware Simulation*.
- **Pilot** - 72 simulated GB300 nodes across four racks for rehearsing the MaxLPS power pilot. Replay a power trace on three racks for the baseline and four racks under MaxLPS, then compare simulated GPU power in Grafana. Follow *Rehearse the MaxLPS Power Pilot With the SDK Simulator*.

3.3.1.2 What's Included

3.3.1.2.1 Emulated Infrastructure (Hardware Emulation and Workload-Aware)

- **144 DGX GB300 Compute Nodes** - Organized across 8 racks (18 nodes per rack)
- **Complete Power Distribution Hierarchy** - Utility, Switchboard, Floor PDUs, Rack PSUs, Compute Systems
- **BMC Simulator** - Provides Redfish API endpoints for all emulated nodes

For this general eight-rack topology, refer to *SDK Simulator Topology*. The Pilot profile uses a separate four-rack topology, described in *Rehearse the MaxLPS Power Pilot With the SDK Simulator*. It demonstrates configuration and control behavior with simulated hardware; inference performance and physical power compliance require real systems.

3.3.1.2.2 Policy Bundles (all profiles)

The DPS server bootstraps the `MaxP`, `MaxQ`, and `MaxLPS` policy bundles regardless of which deployment profile you run, including Default. The SDK also includes `sim/policy-bundles/sdk-site-balanced.yaml`, a tutorial file that you load as the `sdk-site-balanced` bundle rather than a fourth server preset. It is distinct from the guide's downloadable `site-balanced` example: the files and stored names differ, although their current policy payloads are identical.

The *Simulator Playbooks* guide covers loading and inspecting bundles as part of the Default-profile walkthrough.

3.3.1.2.3 Monitoring and Visualization

- **Grafana Dashboards** - Real-time metrics for datacenter power, resource groups, and system operations
- **Prometheus** - Time-series metrics collection and alerting
- **Pyroscope** - Continuous profiling for performance analysis
- **Web UI** - Interactive interface for managing DPS

3.3.1.2.4 Automated Simulation Playbooks

- **Resource Group Simulation** - Automated workload lifecycle testing with configurable parameters
- **Grid Simulation** - Domain-level power management and grid integration testing
- **Load Shedding Simulation** - Power reduction events and recovery scenarios
- **Combined Simulations** - Run multiple scenarios simultaneously for thorough testing

3.3.1.3 Use Cases

The simulator is ideal for:

- **Learning DPS** - Explore concepts, APIs, and workflows in a safe environment
- **SDK Development** - Build and test custom integrations without hardware dependencies
- **Partner Integration** - Develop grid or optimization integrations
- **Quick Start** - Jump-start your custom datacenter deployment

3.3.1.4 Quick Start

System Requirements:

- Linux (Ubuntu/Debian) or macOS
- Minimum 8 GB RAM, 20 GB free disk space
- Internet connection for dependencies

Setup Instructions:

1. **Download and unarchive the DPS SDK files from the NVIDIA NVOnline Portal.**
2. **Install SDK dependencies.** From the SDK directory, run:

   ```
   cd dps-sdk
   task setup
   ```

   This installs docker, kubectl, k3d, helm, helm-git, uv, and `dpsctl`.
3. **Deploy the Default profile.** This is the recommended starting point - a blank DPS environment with no topology loaded:

   ```
   task deploy
   ```

4. **Configure your shell for `dpsctl`.** Paste these exports once to avoid repeating connection flags on every command:

```
export DPSCTL_HOST=api.dps.sdk
export DPSCTL_PORT=80
export DPSCTL_INSECURE_TLS_SKIP_VERIFY=true
```

5. **Continue with the Default-profile playbook.** The *Simulator Playbooks* guide walks you through importing the topology, creating resource groups, and setting grid load targets with `dpsctl`.

3.3.1.4.1 Other Deployment Paths

- **Hardware Emulation (automated playbooks):**

```
task sdk
task sim
```

`task sdk` creates the k3d cluster and deploys DPS; `task sim` imports the topology and runs the combined resource-groups plus grid simulation.

- **Workload-Aware (realistic power traces):** Refer to *Workload-Aware Simulation* for the full deployment flow.
- **Pilot (MaxLPS power-pilot rehearsal):** After `task setup`, deploy the dedicated profile:

```
task deploy:pilot
```

Continue with *Rehearse the MaxLPS Power Pilot With the SDK Simulator* to import the topology, prepare the unmanaged baseline, and run the managed comparison.

3.3.1.5 Access Services

After deployment, the SDK exposes these endpoints:

Service	URL	Default Auth
DPS API (gRPC)	`api.dps.sdk`	`dps/dps`
DPS Web UI	`http://ui.dps.sdk`	`dps/dps`
Grafana	`http://grafana.dps.sdk`	`admin/dps`
Prometheus	`http://prometheus.dps.sdk`	-
Alertmanager	`http://alertmanager.dps.sdk`	-
Pyroscope	`http://pyroscope.dps.sdk`	-

3.3.1.6 Next Steps

- ***Simulator Playbooks*** - Default-profile walkthroughs and Hardware Emulation automated simulations.
- ***Simulator Topology Guide*** - GB300 datacenter topology reference.
- ***Workload-Aware Simulation*** - Deploy and use the surrogate workload model.
- ***Rehearse the MaxLPS Power Pilot With the SDK Simulator*** - Rehearse the MaxLPS workflow with the Pilot profile and compare simulated GPU power.

3.3.1.6.1 SDK Simulator Topology

3.3.1.6.1.1 Overview

This guide describes `sim/topology.json`, the general eight-rack, 144-node topology used by the Hardware Emulation and Workload-Aware profiles. The Pilot profile uses a separate four-rack, 72-node topology in `sim/maxlps-pilot.json`. For its topology, deployment, and comparison procedure, follow *Rehearse the MaxLPS Power Pilot With the SDK Simulator*.

The DPS SDK provides a complete simulated datacenter environment designed to serve as an evaluation playground for the Dynamic Power Software (DPS). This topology, combined with the BMC simulator and simulator playbooks, enables developers and administrators to test power management strategies, resource group operations, and workload optimization without requiring physical hardware infrastructure.

The simulator topology represents a small-scale datacenter deployment with realistic power distribution hierarchies, allowing users to:

- Test and validate policy bundles before deploying to production
- Experiment with resource group creation and management
- Understand power distribution constraints and optimization strategies
- Develop integration code against a stable, reproducible environment
- Perform load testing and capacity planning exercises

3.3.1.6.1.2 Topology Structure

The SDK simulator topology defines a complete power distribution network for a modern AI datacenter featuring **144 DGX GB300 compute nodes** organized across **8 racks**.

3.3.1.6.1.3 Power Distribution Hierarchy

The topology implements a four-tier power distribution architecture that mirrors real-world datacenter deployments:

```
Utility (750 kW)
        Switchboard (1.6 MW)
                FloorPDU-1 (450 kW)   Racks 1, 5
                FloorPDU-2 (450 kW)   Racks 2, 6
                FloorPDU-3 (450 kW)   Racks 3, 7
                FloorPDU-4 (450 kW)   Racks 4, 8
```

3.3.1.6.1.4 Tier 1: Power Domains

Utility Power Domain

- **Capacity**: 750,000 W (750 kW)
- **Purpose**: Represents the building utility connection and primary power feed
- **Design Note**: This intentionally restrictive capacity is chosen to demonstrate power-constrained scenarios. Under maximum load (all 144 nodes using the `MaxP` bundle), the total hardware demand exceeds the utility capacity, enabling testing of power allocation and optimization strategies.

Switchboard Power Domain

- **Capacity**: 1,600,000 W (1.6 MW)
- **Purpose**: Main electrical distribution point for the datacenter floor

3.3.1.6.1.5 Tier 2: Floor PDUs

Four Floor Power Distribution Units distribute power across the datacenter:

FloorPDU	Capacity	Serves Racks	Total Nodes
FloorPDU-1	450 kW	R01, R05	36
FloorPDU-2	450 kW	R02, R06	36
FloorPDU-3	450 kW	R03, R07	36
FloorPDU-4	450 kW	R04, R08	36

3.3.1.6.1.6 Tier 3: Rack Power Supplies (PSUs)

Each of the 8 racks contains 8 redundant Power Supply Units:

Per-Rack Configuration:

- **PSU Count**: 8 power shelves per rack
- **PSU Model**: `PowerSupply95_33000W` (defined in `custom-devices.yaml`)
- **PSU Capacity**: 33,000 W (33 kW) each
- **Efficiency**: 95% (modeled), 97.5% peak in actual hardware
- **Total Rack Capacity**: 264 kW (with redundancy)

Architecture Note: Each PSU in the topology represents a simplified view of the GB300 NVL72 power architecture. In actual GB300 NVL72 deployments, each "PSU" is implemented as a 1U power shelf containing 6 integrated power supply units (each 5.5 kW) in a 3+3 redundant configuration, delivering up to 33 kW per shelf. This topology abstraction treats each power shelf as a single logical PSU for simplicity while maintaining accurate power capacity modeling.

Custom Device Model: PowerSupply95_33000W

This custom power supply model is defined in `custom-devices.yaml`. The SDK simulator playbooks load this custom device definition before importing the topology, making it available for use in `topology.json`.

```
- type: PowerSupply
  description: GB300 Power Shelf at 33000W 95 Pct Eff
  model: PowerSupply95_33000W
  spec:
    maxLoadWatts: 33000
    efficiencyFactor: 0.95
```

Specifications:

- **Maximum Load**: 33,000 W (33 kW)
- **Efficiency**: 95% (0.95 factor)
- **Input Power Calculation**: Output power ÷ 0.95

- **Example**: Delivering 33 kW requires ~34.7 kW input (33,000 ÷ 0.95)

Naming Convention: PSU-R{rack}-{number}

- Example: PSU-R01-1 through PSU-R01-8 for Rack 1

Power Distribution Model:

- All 8 power shelves in a rack power all 18 compute nodes in that rack
- Each node can draw power from any of the 8 power shelves in its rack

3.3.1.6.1.7 Tier 4: Compute Systems

Total Compute Nodes: 144 DGX GB300 systems

Per-Rack Distribution:

- **Nodes per Rack**: 18

Naming Convention: gb300-r{rack}-{node}

- **Rack 1**: gb300-r01-0001 through gb300-r01-0018
- **Rack 2**: gb300-r02-0001 through gb300-r02-0018
- ...continuing through...
- **Rack 8**: gb300-r08-0001 through gb300-r08-0018

3.3.1.6.1.8 Policy Bundles

The topology file defines the simulated power hierarchy and compute entities. It does not contain a top-level `Policies` array or register policy bundle definitions.

The SDK's `sim/policy-bundles/sdk-site-balanced.yaml` tutorial bundle is loaded separately. Follow *SDK Simulator Playbooks* to load and select it. For bundle behavior and lifecycle, refer to *Power Policies*.

3.3.1.6.1.9 Using the Topology

Import and activate the SDK topology with:

```
# Import & activate SDK simulator topology
task sim:tp
```

This command:

1. Creates the custom device models.
2. Imports the topology definition without policy definitions.
3. Deactivates another active topology, if one exists.
4. Activates the topology.
5. Makes all 144 nodes available for resource group assignment.

3.3.1.6.1.10 File Locations

The SDK includes the following key topology files:

- **Topology Definition**: `sim/topology.json`
- **Custom Devices**: `sim/custom-devices.yaml`
- **Tutorial Policy Bundle**: `sim/policy-bundles/sdk-site-balanced.yaml`

3.3.1.6.1.11 Summary

The DPS SDK simulator topology provides a complete, realistic datacenter power distribution model that serves as an invaluable evaluation and development environment. With 144 DGX GB300 nodes across 8 racks and a full four-tier power hierarchy, it enables thorough testing of DPS functionality without physical hardware. Policy bundle definitions are managed separately.

This topology, combined with the BMC simulator and simulator playbooks, allows developers, administrators, and partners to:

- **Understand DPS capabilities** through hands-on experimentation
- **Develop integrations** with grid controllers and optimization engines
- **Test power management strategies** before production deployment
- **Perform capacity planning** and power budget analysis
- **Validate custom policy bundles** and optimization algorithms

3.3.1.6.1.12 Related Documentation

For setup instructions, usage examples, and playbook documentation, refer to the SDK README.

3.3.1.6.2 SDK Simulator Playbooks

This guide walks through the SDK's playbooks for exercising Dynamic Power Software (DPS). The **Default-profile playbook** is the recommended starting point - it uses `dpsctl` against a blank DPS environment so you can learn the API step by step. The **Hardware Emulation playbooks** automate resource group, grid, and load shedding scenarios against the 144-node GB300 topology.

3.3.1.6.2.1 Table of Contents

- *Overview*
- *Default-Profile Playbook*
  - *Deploy the Default Environment*
  - *Configure Your Shell*
  - *Load a Topology*
  - *Load a Policy Bundle*
  - *Create Resource Groups*
  - *Three-Tier Demo*
  - *Set Grid Load Targets*
- *Hardware Emulation Playbooks*

- *Single-Script Playbooks*
  - *Resource Group Simulation*
  - *Grid Simulation*
  - *Load Shedding Simulation*
- *Multi-Script Playbooks*
  - *Combined Simulation (Resource Groups and Grid)*
  - *Load Shedding With Resource Groups*
- *Cleanup*
- *Summary*

3.3.1.6.2.2 Overview

Playbooks fall into two categories:

- **Default-profile playbook**: Manual `dpsctl` exercises against a blank DPS environment. Ideal for learning the API, experimenting with topologies, and running controlled demos.
- **Hardware Emulation playbooks**: Automated shell scripts (`sim/sim_rgs.sh`, `sim/sim_grid.sh`, `sim/sim_loadshed.sh`) that drive the full 144-node GB300 topology. Ideal for soak testing, CI runs, and reproducing realistic workload churn.

For workload-aware playbooks that generate realistic GPU power traces, see the *Workload-Aware Simulation guide*.

3.3.1.6.2.3 Default-Profile Playbook

The Default profile deploys DPS with no topology loaded and no simulators running. Use it to explore the DPS API, practice `dpsctl` commands, or run step-by-step demos against a live DPS server.

3.3.1.6.2.4 Deploy the Default Environment

```
task setup
task deploy
```

`task setup` installs SDK dependencies (docker, kubectl, k3d, helm, helm-git, uv, `dpsctl`). `task deploy` creates a k3d cluster and deploys DPS with default chart values - a blank environment with no topology or simulators.

3.3.1.6.2.5 Configure Your Shell

After `task deploy` completes, paste these exports into your shell once. They configure `dpsctl` to connect to the local SDK endpoint so you can run commands without repeating connection flags:

```
export DPSCTL_HOST=api.dps.sdk
export DPSCTL_PORT=80
export DPSCTL_INSECURE_TLS_SKIP_VERIFY=true
```

3.3.1.6.2.6 Load a Topology

The Default environment starts with no data. Register the custom device types used by the topology, then import and activate it:

```
# Register custom device definitions (required before topology import)
dpsctl device upsert sim/custom-devices.yaml

# Import and activate the topology
dpsctl tp import sim/topology.json
dpsctl tp activate --topology dps-simulator
```

Verify the topology is active:

```
dpsctl tp list
dpsctl tp list-entities
```

3.3.1.6.2.7 Load a Policy Bundle

The server already provides `MaxP`, `MaxQ`, and `MaxLPS`. Load the SDK's separate `sdk-site-balanced` tutorial bundle before using it in the following resource-group example:

```
# Inspect the bootstrapped bundles
dpsctl policy bundle list

# Create or replace the custom tutorial bundle, then inspect it
dpsctl policy bundle upsert sim/policy-bundles/sdk-site-balanced.yaml
dpsctl policy bundle get sdk-site-balanced
```

3.3.1.6.2.8 Create Resource Groups

A resource group represents an active workload. Select a named policy bundle when you create the group. This example uses the custom tutorial bundle loaded in the preceding section.

```
# Create a resource group
dpsctl rg create --external-id 1 --resource-group my-workload --policy-bundle
↪sdk-site-balanced

# Add nodes from the topology
dpsctl rg add --resource-group my-workload --entities "gb300-r01-0001,gb300-
↪r01-0002,gb300-r01-0003"

# Activate - DPS enforces the power cap on all nodes in the group
dpsctl rg activate --resource-group my-workload --sync

# List all resource groups
dpsctl rg list

# Delete when done
dpsctl rg delete --resource-group my-workload
```

3.3.1.6.2.9 Three-Tier Demo

Create, add, and activate three illustrative tiers in one sequence to mimic a multi-tenant datacenter. These examples select the server-authored bundles by name; they do not convert or preserve the legacy `GB300-High`, `GB300-Med`, or `GB300-Low` watt records. Inspect the installed bundle definitions before using them for a real workload. Lower `--priority` numbers are higher precedence, so `premium-tier` at `0` is reduced last when power is stolen:

```
# premium-tier - built-in MaxP bundle, Racks 1-2 (36 nodes)
# --priority 0 is the default and the highest precedence (reduced last when
 ↪power is stolen).
dpsctl rg create --external-id 1 --resource-group premium-tier --policy-
 ↪bundle MaxP --priority 0
dpsctl rg add --resource-group premium-tier --entities "gb300-r01-0001,gb300-
 ↪r01-0002,gb300-r01-0003,gb300-r01-0004,gb300-r01-0005,gb300-r01-0006,gb300-
 ↪r01-0007,gb300-r01-0008,gb300-r01-0009,gb300-r01-0010,gb300-r01-0011,gb300-
 ↪r01-0012,gb300-r01-0013,gb300-r01-0014,gb300-r01-0015,gb300-r01-0016,gb300-
 ↪r01-0017,gb300-r01-0018,gb300-r02-0001,gb300-r02-0002,gb300-r02-0003,gb300-
 ↪r02-0004,gb300-r02-0005,gb300-r02-0006,gb300-r02-0007,gb300-r02-0008,gb300-
 ↪r02-0009,gb300-r02-0010,gb300-r02-0011,gb300-r02-0012,gb300-r02-0013,gb300-
 ↪r02-0014,gb300-r02-0015,gb300-r02-0016,gb300-r02-0017,gb300-r02-0018"
dpsctl rg activate --resource-group premium-tier --sync

# value-tier - built-in MaxQ bundle, Racks 3-4 (36 nodes)
dpsctl rg create --external-id 2 --resource-group value-tier --policy-bundle
 ↪MaxQ --priority 1
dpsctl rg add --resource-group value-tier --entities "gb300-r03-0001,gb300-
 ↪r03-0002,gb300-r03-0003,gb300-r03-0004,gb300-r03-0005,gb300-r03-0006,gb300-
 ↪r03-0007,gb300-r03-0008,gb300-r03-0009,gb300-r03-0010,gb300-r03-0011,gb300-
 ↪r03-0012,gb300-r03-0013,gb300-r03-0014,gb300-r03-0015,gb300-r03-0016,gb300-
 ↪r03-0017,gb300-r03-0018,gb300-r04-0001,gb300-r04-0002,gb300-r04-0003,gb300-
 ↪r04-0004,gb300-r04-0005,gb300-r04-0006,gb300-r04-0007,gb300-r04-0008,gb300-
 ↪r04-0009,gb300-r04-0010,gb300-r04-0011,gb300-r04-0012,gb300-r04-0013,gb300-
 ↪r04-0014,gb300-r04-0015,gb300-r04-0016,gb300-r04-0017,gb300-r04-0018"
dpsctl rg activate --resource-group value-tier --sync

# research - built-in MaxLPS bundle, Racks 5-8 (72 nodes)
dpsctl rg create --external-id 3 --resource-group research --policy-bundle
 ↪MaxLPS --priority 2
dpsctl rg add --resource-group research --entities "gb300-r05-0001,gb300-r05-
 ↪0002,gb300-r05-0003,gb300-r05-0004,gb300-r05-0005,gb300-r05-0006,gb300-r05-
 ↪0007,gb300-r05-0008,gb300-r05-0009,gb300-r05-0010,gb300-r05-0011,gb300-r05-
 ↪0012,gb300-r05-0013,gb300-r05-0014,gb300-r05-0015,gb300-r05-0016,gb300-r05-
 ↪0017,gb300-r05-0018,gb300-r06-0001,gb300-r06-0002,gb300-r06-0003,gb300-r06-
 ↪0004,gb300-r06-0005,gb300-r06-0006,gb300-r06-0007,gb300-r06-0008,gb300-r06-
 ↪0009,gb300-r06-0010,gb300-r06-0011,gb300-r06-0012,gb300-r06-0013,gb300-r06-
 ↪0014,gb300-r06-0015,gb300-r06-0016,gb300-r06-0017,gb300-r06-0018,gb300-r07-
 ↪0001,gb300-r07-0002,gb300-r07-0003,gb300-r07-0004,gb300-r07-0005,gb300-r07-
 ↪0006,gb300-r07-0007,gb300-r07-0008,gb300-r07-0009,gb300-r07-0010,gb300-r07-
 ↪0011,gb300-r07-0012,gb300-r07-0013,gb300-r07-0014,gb300-r07-0015,gb300-r07-
 ↪0016,gb300-r07-0017,gb300-r07-0018,gb300-r08-0001,gb300-r08-0002,gb300-r08-
 ↪0003,gb300-r08-0004,gb300-r08-0005,gb300-r08-0006,gb300-r08-0007,gb300-r08-
 ↪0008,gb300-r08-0009,gb300-r08-0010,gb300-r08-0011,gb300-r08-0012,gb300-r08-
```

(continues on next page)

(continued from previous page)

```
↪0013,gb300-r08-0014,gb300-r08-0015,gb300-r08-0016,gb300-r08-0017,gb300-r08-
↪0018"
dpsctl rg activate --resource-group research --sync
```

3.3.1.6.2.10 Set Grid Load Targets

A grid load target tells DPS that the datacenter power feed is constrained. Each `set-load-target` call immediately overrides the previous one. Run this sequence after creating resource groups to observe DPS reacting to demand-response events in real time.

These commands use GNU `date` syntax (`-d '+1 minute'`), available on both Linux and macOS after running `task setup` (which installs GNU coreutils via Homebrew).

```
# 1. Set an initial constraint - 700 kW, active until overridden
dpsctl nvgrid set-load-target --value 700000 --unit watt --feed-tags Utility -
↪-start-time "$(date -u +%Y-%m-%dT%H:%M:%SZ)" --best-effort

# Check active load targets
dpsctl nvgrid get-current

# 2. Reduce further - 500 kW for one minute, then expires automatically
dpsctl nvgrid set-load-target --value 500000 --unit watt --feed-tags Utility -
↪-start-time "$(date -u +%Y-%m-%dT%H:%M:%SZ)" --end-time "$(date -u -d '+1
↪minute' +%Y-%m-%dT%H:%M:%SZ 2>/dev/null || date -u -v+1M +%Y-%m-%dT%H:%M:
↪%SZ)" --best-effort

# Check active load targets
dpsctl nvgrid get-current

# 3. Restore to original level - 750 kW, active until overridden
dpsctl nvgrid set-load-target --value 750000 --unit watt --feed-tags Utility -
↪-start-time "$(date -u +%Y-%m-%dT%H:%M:%SZ)" --best-effort
```

Open the Web UI at `http://ui.dps.sdk` (auth: `dps/dps`) or Grafana at `http://grafana.dps.sdk` (auth: `admin/dps`) while running the sequence above to watch DPS respond to each constraint change in real time.

3.3.1.6.2.11 Hardware Emulation Playbooks

The Hardware Emulation profile provides a complete emulated datacenter environment using pseudo-random hardware responses based on device characteristics. Deploy it before running the playbooks below:

```
task sdk
task sim
```

`task sdk` creates the k3d cluster and deploys DPS. `task sim` imports the topology, runs the combined resource-groups plus grid simulation, and keeps driving load until you stop it with `Ctrl+C`.

3.3.1.6.2.12 Single-Script Playbooks

3.3.1.6.2.13 Resource Group Simulation

Description: Automated creation and deletion of resource groups with configurable parameters. Simulates realistic workload lifecycle patterns with randomized power management features.

The `task sim:rgs` target runs the `sim/sim_rgs.sh` script, which handles the full lifecycle of resource group management.

Script Dependencies:

- `dpsctl` - DPS command-line interface
- `jq` - JSON parsing

Quick Start:

```
# Start basic simulation
task sim:rgs

# Quick test simulation
task sim:rgs END_AFTER=120 MAX_RGS=5 MIN_DURATION=30 MAX_DURATION=60

# Extended simulation
task sim:rgs END_AFTER=1800 MAX_RGS=20 MIN_DURATION=120 MAX_DURATION=300

# High-intensity simulation
task sim:rgs MAX_RGS=25 MIN_RG_SIZE=5 MAX_RG_SIZE=15
```

Parameters:

Core parameters:

- `MAX_RGS`: Maximum number of resource groups (default: 15)
- `MIN_DURATION`: Minimum RG lifecycle duration in seconds (default: 60)
- `MAX_DURATION`: Maximum RG lifecycle duration in seconds (default: 300)
- `MIN_RG_SIZE`: Minimum resources per RG (default: 2)
- `MAX_RG_SIZE`: Maximum resources per RG (default: 10)
- `END_AFTER`: End simulation after specified seconds (default: 600)

Feature randomization flags:

- `NO_WPPS`: Disable workload profile (WPPS) randomization (default: false)
- `NO_PRS`: Disable PRS randomization (default: false)
- `NO_DPM`: Disable DPM randomization (default: false)
- `NO_ALLOW_REPROVISION`: Disable allow-reprovision randomization (default: false)

What It Does:

1. **Initialization Phase:**
   - Fetches available resources from topology
   - Loads available policy bundles from the DPS server
   - Can optionally clean up existing resource groups

2. **Simulation Loop:**
   - Creates resource groups up to `MAX_RGS` limit
   - Randomly selects resources (between `MIN_RG_SIZE` and `MAX_RG_SIZE`)
   - Randomly selects a policy bundle from the available bundles
   - Randomly enables or disables optional features (WPPS, PRS, DPM, allow-reprovision)
   - Tracks resource allocation to prevent conflicts
   - Automatically deletes RGs after their lifecycle duration expires
3. **Resource Group Lifecycle:**
   - **Create**: Creates an RG with an external ID and selected policy bundle
   - **Add**: Adds selected resources to the RG
   - **Activate**: Activates the RG with optional features
   - **Delete**: Removes RG after duration expires

Script Implementation Details:

- Maintains associative arrays for active RGs and resource usage
- Prevents resource conflicts by tracking which resources are in use
- Generates unique RG IDs in format: `sim-rg-XXXX` (e.g., `sim-rg-0001`)
- Uses unique external IDs starting from 1000
- Default check interval: 10 seconds

Feature Randomization:

When enabled (default), the script randomly applies these features:

- **WPPS (Workload Profile)**: Randomly selects 1-2 profiles from IDs 3-15
- **PRS (Power Regulation Service)**: Randomly enables or disables
- **DPM (Dynamic Power Management)**: Randomly enables or disables
- **Allow Reprovision**: Randomly enables or disables during activation

To disable specific features:

```
# Disable WPPS and PRS randomization
task sim:rgs NO_WPPS=true NO_PRS=true

# Disable only allow-reprovision randomization
task sim:rgs NO_ALLOW_REPROVISION=true

# Disable all randomization features
task sim:rgs NO_WPPS=true NO_PRS=true NO_DPM=true NO_ALLOW_REPROVISION=true
```

3.3.1.6.2.14 Grid Simulation

Description: Simulates nvgrid load targets for testing domain-level power management and grid integration. Schedules multiple concurrent load targets across power domains.

The `task sim:grid` target runs the `sim/sim_grid.sh` script, which manages nvgrid load target scheduling across power domains.

Script Dependencies:

- `dpsctl` - DPS command-line interface
- `jq` - JSON parsing

Quick Start:

```
# Start grid simulation with defaults
task sim:grid

# Run longer simulation (1 hour)
task sim:grid END_AFTER=3600

# Quick test with higher load range
task sim:grid END_AFTER=600 MIN_LOAD_PERCENT=80 MAX_LOAD_PERCENT=95

# Custom check interval
task sim:grid INTERVAL=120 MIN_DURATION=60 MAX_DURATION=300
```

Parameters:

- `MAX_RGS`: Maximum number of resource groups (default: 15)
- `MIN_DURATION`: Minimum RG lifecycle duration in seconds (default: 120)
- `MAX_DURATION`: Maximum RG lifecycle duration in seconds (default: 600)
- `MIN_RG_SIZE`: Minimum resources per RG (default: 2)
- `MAX_RG_SIZE`: Maximum resources per RG (default: 10)
- `END_AFTER`: End simulation after specified seconds (default: 1800)
- `INTERVAL`: Check interval in seconds (default: 300)
- `MIN_LOAD_PERCENT`: Minimum load as percentage of domain capacity (default: 40)
- `MAX_LOAD_PERCENT`: Maximum load as percentage of domain capacity (default: 90)
- `NO_WPPS`: Disable workload profile (WPPS) randomization (default: false)
- `NO_PRS`: Disable PRS randomization (default: false)
- `NO_DPM`: Disable DPM randomization (default: false)
- `NO_ALLOW_REPROVISION`: Disable allow-reprovision randomization (default: false)

What It Does:

1. **Initialization Phase:**
   - Fetches topology and identifies all power domains
   - Retrieves capacity limits for each power domain
   - Displays current load targets before starting
2. **Simulation Loop:**
   - Every `INTERVAL` seconds, checks active targets
   - Schedules 2-5 new random grid events if under capacity limit (max 5 concurrent)
   - Each event targets a random power domain
   - Load values are randomized within specified percentage range

- Duration is randomized between min and max values
- Events are scheduled with random gaps (2-30 minutes) between them

3. **Event Scheduling:**
   - Events can be scheduled on the same domain (non-conflicting times)
   - Automatically adjusts start times to avoid conflicts
   - Maintains at most 5 active targets at any time
   - Tracks earliest start and latest end times per domain
4. **Cleanup:**
   - On interruption (`Ctrl+C`), expires all scheduled events
   - Uses bulk expiration by setting load to 0 W for the scheduled timeframe

Implementation Details:

- Maintains associative arrays for power domains, active targets, and domain timelines
- Generates unique target IDs in format: `domain:counter`
- Cleans up expired targets automatically before scheduling new ones
- Supports multiple load targets on the same domain with time separation

Scheduling Strategy:

- Minimum events per interval: 2
- Maximum events per interval: 5
- Maximum concurrent active targets: 5
- Event gap range: 120-1800 seconds (2-30 minutes)
- Adds 30-second buffer between events on same domain

3.3.1.6.2.15 Load Shedding Simulation

Description: Simulates a single load shedding and reactivation cycle for testing power reduction scenarios and recovery. All events are scheduled upfront, making it ideal for testing system response to planned power constraints.

The `task sim:loadshed` (alias: `task sim:ls`) target runs the `sim/sim_loadshed.sh` script, which schedules and monitors a complete load shedding cycle.

Script Dependencies:

- `dpsctl` - DPS command-line interface
- `jq` - JSON parsing
- `bc` - Floating-point calculations

Quick Start:

```
# Start load shedding simulation with defaults
# (100% to 5% in 5% steps, hold 5 min, reactivate to 100%)
task sim:loadshed

# Quick test with faster intervals
```

(continues on next page)

(continued from previous page)

```
task sim:loadshed SHED_INTERVAL=30 HOLD_TIME=120

# Shed to 10% in 10% increments
task sim:loadshed SHED_INCREMENT=10 MIN_LOAD_PERCENT=10

# Aggressive load shedding (20% steps down to 20%)
task sim:loadshed SHED_INCREMENT=20 MIN_LOAD_PERCENT=20

# Target specific power domain
task sim:loadshed FEED_TAG=PDU-GB200-1

# Use short alias
task sim:ls SHED_INCREMENT=10 MIN_LOAD_PERCENT=10
```

Parameters:

- SHED_INTERVAL: Time between load shedding steps in seconds (default: 60)
- SHED_INCREMENT: Load percentage increment for steps (default: 5)
- HOLD_TIME: Time to hold at minimum load in seconds (default: 300)
- MIN_LOAD_PERCENT: Minimum load percentage to shed down to (default: 5)
- FEED_TAG: Specific feed tag (power domain) to target (default: auto-select first domain)

What It Does:

Phase 1: Load Shedding

- Reduces load from 100% to MIN_LOAD_PERCENT in SHED_INCREMENT steps
- Each step lasts for SHED_INTERVAL seconds
- Example: 100% to 95% to 90% to ... to 5% (with default 5% increment)

Phase 2: Hold

- Maintains minimum load for HOLD_TIME seconds
- Tests system stability under sustained low load

Phase 3: Reactivation

- Increases load from MIN_LOAD_PERCENT back to 100% in SHED_INCREMENT steps
- Each step lasts for SHED_INTERVAL seconds
- Example: 10% to 15% to 20% to ... to 100% (with 5% increment from 10% min)

Implementation Details:

1. **Domain Selection:**
   - If no FEED_TAG specified, auto-selects first power domain
   - Retrieves domain capacity and power factor from topology
   - Calculates maximum load as `capacity x power_factor`
2. **Event Scheduling:**
   - All events are pre-scheduled at simulation start
   - Calculates total simulation duration upfront

- Each load target is scheduled with precise start and end times
- No dynamic adjustments during execution

3. **Calculation Example:**

   With defaults (5% increment, 5% min load, 60s interval, 300s hold):

   - Shedding steps: 100%, 95%, 90%, ..., 5% = 20 events
   - Hold: 1 event
   - Reactivation steps: 10%, 15%, 20%, ..., 100% = 19 events
   - Total events: 40 events
   - Total duration: (20 x 60) + 300 + (19 x 60) = 2,640 seconds (~44 minutes)

4. **Cleanup:**

   - On interruption (`Ctrl+C`), cancels all scheduled events
   - Sets load to 0 W for the entire simulation timeframe
   - Ensures clean exit with no lingering targets

Behavior Notes:

- The simulator schedules all events upfront and then monitors progress
- Actual power adjustments happen automatically per the schedule
- The script does not dynamically adjust based on system response
- Power factor from topology is automatically applied to capacity calculations

3.3.1.6.2.16 Multi-Script Playbooks

3.3.1.6.2.17 Combined Simulation (Resource Groups and Grid)

Description: Runs both resource group and grid simulations simultaneously for thorough testing of workload management and power domain optimization.

Script Location: Uses both `sim/sim_rgs.sh` and `sim/sim_grid.sh`.

Quick Start:

```
# Start combined simulation with defaults
task sim

# Run longer simulation (1 hour)
task sim END_AFTER=3600

# Quick test with higher load range
task sim END_AFTER=600 MIN_LOAD_PERCENT=80 MAX_LOAD_PERCENT=95
```

Parameters:

Resource Group parameters:

- `MAX_RGS`: Maximum number of resource groups (default: 15)
- `MIN_DURATION`: Minimum RG lifecycle duration in seconds (default: 60)
- `MAX_DURATION`: Maximum RG lifecycle duration in seconds (default: 300)

- `MIN_RG_SIZE`: Minimum resources per RG (default: 2)
- `MAX_RG_SIZE`: Maximum resources per RG (default: 10)
- `END_AFTER`: End simulation after specified seconds (default: 600)

Grid simulation parameters:

- `INTERVAL`: Check interval in seconds (default: 300)
- `MIN_LOAD_PERCENT`: Minimum load as percentage of domain capacity (default: 40)
- `MAX_LOAD_PERCENT`: Maximum load as percentage of domain capacity (default: 90)

What It Does:

- Continuously creates and deletes resource groups based on configured lifecycle parameters
- Schedules 2-5 random grid load targets across power domains every interval
- Tests both workload churn and power optimization simultaneously
- Monitors resource utilization and power distribution

Implementation Details:

- Runs two simulator processes in parallel
- Automatically handles cleanup on interruption (`Ctrl+C`)
- Both simulators share the same `END_AFTER` duration for synchronized termination
- Resource groups and grid targets operate independently

3.3.1.6.2.18 Load Shedding With Resource Groups

Description: Runs both load shedding and resource groups simulators in parallel, providing thorough testing of system behavior under combined workload churn and power constraints.

Script Location: Uses both `sim/sim_loadshed.sh` and `sim/sim_rgs.sh`.

Quick Start:

```
# Run with defaults
task sim:loadshed-rgs

# Faster load shedding with aggressive RG spawning
task sim:loadshed-rgs SHED_INTERVAL=30 MAX_RGS=25

# Custom configuration for both simulators
task sim:loadshed-rgs SHED_INCREMENT=10 MIN_LOAD_PERCENT=10 MAX_RGS=20 END_
↪AFTER=2000

# Use short alias
task sim:ls-rgs SHED_INCREMENT=10 MAX_RGS=25
```

Parameters:

Load Shedding:

- `SHED_INTERVAL`: Time between load shedding steps in seconds (default: 60)
- `SHED_INCREMENT`: Load percentage increment for steps (default: 5)
- `HOLD_TIME`: Time to hold at minimum load in seconds (default: 300)

- `MIN_LOAD_PERCENT`: Minimum load percentage to shed down to (default: 5)
- `FEED_TAG`: Specific feed tag (power domain) to target (default: auto-select)

Resource Groups:

- `MIN_DURATION`: Minimum duration for RG lifecycle in seconds (default: 60)
- `MAX_DURATION`: Maximum duration for RG lifecycle in seconds (default: 180)
- `END_AFTER`: End RG simulation after specified seconds (default: 2640 - matches load shed duration)
- `MAX_RGS`: Maximum number of RGs to spawn (default: 15)
- `MIN_RG_SIZE`: Minimum resources per RG (default: 2)
- `MAX_RG_SIZE`: Maximum resources per RG (default: 10)

What It Does:

1. **Parallel Execution:**
   - Starts load shedding simulator in background
   - Starts resource group simulator in background
   - Both run simultaneously and independently
2. **Load Shedding Component:**
   - Executes full load shedding cycle as described in *Load Shedding Simulation*
   - Schedules all power reduction and reactivation events upfront
   - Runs for calculated duration based on shed parameters
3. **Resource Groups Component:**
   - Continuously creates and deletes resource groups as described in *Resource Group Simulation*
   - Runs until `END_AFTER` duration (defaults to match load shed duration)
   - Maintains up to `MAX_RGS` concurrent resource groups
4. **Synchronized Termination:**
   - By default, both simulators end at approximately the same time
   - `END_AFTER` for RGs is set to match load shedding total duration (2640s with defaults)
   - Both simulators clean up gracefully on completion or interruption
5. **Cleanup:**
   - Interrupt signal (`Ctrl+C`) triggers cleanup for both processes
   - Kills both simulator PIDs and waits for clean exit
   - Prevents recursive cleanup with guard variable

Implementation Details:

- Uses bash process management with background jobs
- Trap handles INT, TERM, HUP, QUIT, ABRT, ALRM, USR1, USR2 signals
- Cleanup function ensures both processes terminate together
- Exit code 130 (128 + SIGINT) on manual interruption

Timing Coordination:

With default values:

- Load shedding duration: ~2640 seconds (44 minutes)
- Resource groups `END_AFTER`: 2640 seconds (synchronized)
- Both simulators complete at approximately the same time

3.3.1.6.2.19 Cleanup

Interrupt simulators cleanly:

- Press `Ctrl+C` to trigger cleanup handlers
- Scripts automatically clean up created resources
- Wait for cleanup to complete before restarting

3.3.1.6.2.20 Summary

The SDK provides a complete set of playbooks for testing datacenter power management scenarios:

- **Default-profile playbook**: Manual `dpsctl` walkthrough for learning the API, loading the topology, creating resource groups, and setting grid load targets.
- **Hardware Emulation single-script playbooks**: Resource Group Simulation, Grid Simulation, and Load Shedding Simulation focus on one aspect each.
- **Hardware Emulation multi-script playbooks**: Combined Simulation and Load Shedding With Resource Groups combine multiple simulators for more complex scenarios.

Start with the Default-profile playbook to understand DPS concepts, then move to the Hardware Emulation playbooks for automated, long-running scenarios. For realistic GPU power traces driven by a workload model, see the *Workload-Aware Simulation guide*.

3.3.1.6.3 Workload-Aware Simulation

The Workload-Aware profile deploys the DPS BMC Simulator with a workload-model service that generates realistic power phases, durations, and signal characteristics based on recorded workloads. Unlike Hardware Emulation (which returns pseudorandom power values), the workload model accepts a workload name and a power cap, then delivers a realistic GPU power trace.

The DPS BMC Simulator calls this model over gRPC and serves the resulting trace through its Redfish API, so DPS manages power exactly as it would with real hardware.

The SDK includes a built-in example model. You can replace it with your own implementation — for example, an ML model trained on production telemetry to analyze how different workloads perform under various power caps.

For a guided MaxLPS power-pilot rehearsal, follow *Rehearse the MaxLPS Power Pilot With the SDK Simulator*. It uses the dedicated Pilot profile with 72 nodes across four racks and a supplied power trace. This guide covers the general 144-node Workload-Aware profile and custom workload-model development. Power-trace replay does not run an inference service or measure its throughput or latency.

3.3.1.6.3.1 Overview

The DPS BMC simulator runs a surrogate plugin that delegates GPU power predictions to an external gRPC service (the workload model). The workload model runs as a sidecar container alongside the BMC simulator:

```
bmc-surrogate-simulator pod

                        gRPC
    BMC simulator                   Workload
    (surrogate         :50052       model
     plugin)                        (Python)
```

The gRPC interface (`surrogate.proto`) has two methods:

- **Predict** — Given a power cap and hardware/job context, return a predicted GPU power time series.
- **GetInfo** — Return model metadata. Called at startup for health checks.

When a workload starts on a node, the surrogate plugin calls the workload model's `Predict` method with the current power cap and job context. The model returns a finite series of power and utilization values that the surrogate plugin replays in a loop.

The SDK includes a complete, runnable example workload model in `examples/workload-model/`. The example generates a sin-shaped power trace for demonstration purposes. Copy it and replace the prediction logic in `workload_model_server.py` with your own trained model to produce realistic GPU power traces for any workload.

3.3.1.6.3.2 Use Cases

By building a custom workload model that captures real GPU power behavior, the Workload-Aware profile enables power-aware GPU workload management — showing how power capping affects job performance and cluster power consumption before changes are applied.

Key use cases include:

- **Cluster Power Simulation** - Assess the impact of power limits on job runtimes and overall cluster power consumption without disrupting production workloads
- **Workload Optimization** - Explore performance-power trade-offs across caps to find optimal operating points for GPU applications
- **Power-Aware Scheduling** - Inform scheduling decisions about placement and power allocation based on predicted power behavior
- **Cost Analysis** - Evaluate cost-benefit of capping strategies by balancing performance requirements and energy costs
- **Research and Development** - Study power management behavior across workloads and caps using synthetic but realistic traces

3.3.1.6.3.3 Prerequisites

Before deploying the Workload-Aware profile, ensure you have completed the SDK setup described in the *SDK Simulator User Guide*:

- Linux (Ubuntu/Debian) or macOS (limited support)
- Minimum 8GB RAM, 20GB free disk space
- Internet connection for dependencies
- SDK dependencies installed (`task setup`)

3.3.1.6.3.4 Deployment

3.3.1.6.3.5 Step 1: Create the Cluster

```
task deploy:cluster:create
```

3.3.1.6.3.6 Step 2: Deploy the SDK in Workload-Aware Mode

```
task deploy:sdk:surrogate
```

This command builds the example workload model, publishes it to the local cluster, and deploys the DPS SDK using the surrogate environment configuration (`environments/surrogate/values.yaml`), which:

- Disables all default BMC simulator profiles (Hardware Emulation)
- Deploys the example Python workload model as a sidecar alongside the BMC simulator
- Configures all 144 DGX GB300 nodes for workload-aware simulation
- Enables monitoring (Prometheus, Grafana) and the BMC simulator admin API ingress

3.3.1.6.3.7 Step 3: Import and Activate Topology

```
task simulator:topology
```

This command imports the default topology (`sim/topology.json`) and activates the simulator under the name `dps-simulator`. For details about the topology structure, refer to the *Simulator Topology Guide*.

To override the topology file or name:

```
task simulator:topology TOPOLOGY_FILE=./sim/example.json TOPOLOGY_NAME=example
```

> **Note:** A topology must be active before running workload-aware simulations.

3.3.1.6.3.8 Step 4: Run a Simulation

Run a quick simulation to verify the deployment is working:

```
task simulator:surrogate:install-deps
task sim:surrogate:rgs END_AFTER=120 MAX_RGS=3
```

Open Grafana at `http://grafana.dps.sdk` (admin/dps) to see GPU power traces generated by the workload model. With the default example model, you will see sin-shaped power curves. Refer to *Simulation Playbooks* for all available simulation scenarios.

3.3.1.6.3.9 Custom Workload Model

The SDK includes an example Python workload model in `examples/workload-model/` that implements the gRPC surrogate interface. The example generates a sin-shaped power trace for demonstration. Replace the prediction logic in `workload_model_server.py` with your own ML model to produce realistic GPU power traces.

3.3.1.6.3.10 Deployment Options

Default deployment — the surrogate environment deploys the example model automatically:

```
task deploy:sdk:surrogate
```

Deploy with the built-in Go model (no custom image needed) — override the workload model in `environments/surrogate/values.yaml`:

```
dps-bmc-simulator:
  dpsBmcSimulator:
    surrogateMocks:
      surrogate:
        workloadModel:
          enabled: true
          command: ["/app/dps-bmc-simulator"]
          args: ["serve", "fake-workload-model", "--port", "50052"]
```

Deploy your own custom model — build and push your image, then set:

```
        workloadModel:
          enabled: true
          image: "your-registry/your-workload-model:latest"
```

Then redeploy:

```
helm upgrade --install dps-sdk chart --namespace dps \
  --values environments/surrogate/values.yaml
```

3.3.1.6.3.11 Development Loop

The inner development loop for building a custom workload model is:

1. **Edit** `workload_model_server.py` — replace `generate_fake_timeseries()` with your model inference
2. **Build** — `task example:workload-model:build`
3. **Publish** — `task example:workload-model:push`
4. **Restart** — `kubectl rollout restart deploy/bmc-surrogate-simulator -n dps`
5. **Verify** — check logs and run a simulation

Run locally (without Kubernetes):

```
task example:workload-model:run
```

Regenerate gRPC stubs (after editing `surrogate.proto`):

```
task example:workload-model:generate
```

See `examples/workload-model/README.md` in the SDK repository for the full development guide, gRPC interface reference, and tips.

3.3.1.6.3.12 gRPC Interface

The workload model implements a gRPC service defined in `surrogate.proto` with two methods:

- **Predict** — Given a power cap and hardware/job context, return a predicted GPU power time series.
- **GetInfo** — Return model metadata. Called at startup for health checks.

3.3.1.6.3.13 What Predict Receives

Field	Type	Description
`cap_w`	double	Per-GPU power cap in watts
`hardware.num_gpus`	int32	GPUs per node
`hardware.gpu_model`	string	GPU model (e.g. “GB300”)
`hardware.hostname`	string	Simulator hostname
`hardware.gpu_id`	int32	GPU index within the node
`job.job_id`	string	DPS job identifier
`job.workload.*`	WorkloadMetadata	Workload info (model name, batch size)
`duration_hint_sec`	double	Optional hint for trace length

3.3.1.6.3.14 What Predict Must Return

A `PredictResponse` with a list of `TimeSeriesPoint`:

Field	Type	Description
`timestamp_sec`	int64	Seconds from job start (0, 30, 60, ...)
`power_usage_w`	double	Per-GPU power consumption in watts
`utilization_pct`	double	GPU utilization percentage (0-100)

The series must be finite. The surrogate plugin replays it in a loop. Sample period is 30 seconds by convention.

3.3.1.6.3.15 Simulation Playbooks

The Workload-Aware profile provides dedicated playbooks that leverage the surrogate workload model. These playbooks mirror the *Hardware Emulation playbooks* but produce power traces driven by the workload model instead of pseudorandom values.

3.3.1.6.3.16 Install Dependencies

Before running workload-aware playbooks, install the required Python dependencies:

```
task simulator:surrogate:install-deps
```

3.3.1.6.3.17 Single-Node Workload Simulation

Simulates a single workload execution on a specific node using the workload-aware BMC simulator.

```
task simulator:surrogate:workload NODE=gb300-r01-0001 WORKLOAD=nemotron-12b
```

List available options:

```
task simulator:surrogate:workload -- --list-workloads
task simulator:surrogate:workload -- --list-hosts
task simulator:surrogate:workload -- --list-jobs
```

3.3.1.6.3.18 Resource Group Simulation

Creates and manages resource groups with realistic workloads. Nodes are randomly distributed between resource groups, and when a workload completes, nodes are recycled for new resource groups.

```
# Start basic simulation (10 minutes, 10 RGs max, 3 parallel)
task sim:surrogate:rgs

# Quick test simulation
task sim:surrogate:rgs END_AFTER=120 MAX_RGS=5 MAX_PARALLEL=2

# Extended simulation with more concurrent RGs
task sim:surrogate:rgs END_AFTER=1800 MAX_RGS=20 MAX_PARALLEL=5

# Custom workload and time scale
task sim:surrogate:rgs WORKLOAD=canonical_trace_100pct TIME_SCALE=100
```

List available options:

```
task sim:surrogate:rgs -- --list-nodes
task sim:surrogate:rgs -- --list-workloads
```

Parameters:

- `END_AFTER`: End simulation after N seconds (default: 600)
- `MAX_RGS`: Maximum total resource groups to create (default: 10)
- `MAX_PARALLEL`: Maximum concurrent resource groups (default: 3)
- `MIN_RG_SIZE`: Minimum nodes per resource group (default: 1)

- MAX_RG_SIZE: Maximum nodes per resource group (default: 20)
- WORKLOAD: Surrogate workload name (default: nemotron-12b)
- TIME_SCALE: Time acceleration factor (default: 50.0)

3.3.1.6.3.19 Combined Simulation (Resource Groups and Grid)

Runs both workload-aware resource group simulation and grid simulation together. This is the workload-aware equivalent of `task sim`.

```
# Start combined simulation (default 10 minutes)
task sim:surrogate

# Run longer simulation (1 hour)
task sim:surrogate END_AFTER=3600

# Quick test with more RGs and higher grid load
task sim:surrogate END_AFTER=300 MAX_PARALLEL=5 MIN_LOAD_PERCENT=80

# Custom workload configuration
task sim:surrogate WORKLOAD=canonical_trace_100pct TIME_SCALE=100 MAX_RGS=15
```

Parameters (combines resource group and grid parameters):

- END_AFTER: End simulation after N seconds (default: 600)
- MAX_RGS: Maximum total resource groups to create (default: 10)
- MAX_PARALLEL: Maximum concurrent resource groups (default: 3)
- WORKLOAD: Surrogate workload name (default: nemotron-12b)
- TIME_SCALE: Time acceleration factor (default: 50.0)
- MIN_LOAD_PERCENT: Minimum grid load percentage (default: 40)
- MAX_LOAD_PERCENT: Maximum grid load percentage (default: 90)

3.3.1.6.3.20 Load Shedding with Resource Groups

Runs load shedding simulation alongside workload-aware resource group simulation. This is the workload-aware equivalent of `task sim:loadshed-rgs`, testing how load shedding affects nodes running realistic workloads.

```
# Run with default settings
task sim:surrogate:loadshed-rgs

# Run with faster load shedding
task sim:surrogate:loadshed-rgs SHED_INTERVAL=30 MAX_PARALLEL=5

# Run with custom parameters
task sim:surrogate:loadshed-rgs SHED_INTERVAL=45 HOLD_TIME=180 MAX_RGS=15
```

Parameters (combines load shedding and surrogate resource group parameters):

- SHED_INTERVAL: Time between load shedding steps in seconds (default: 60)
- SHED_INCREMENT: Load percentage increment for steps (default: 5)

- `HOLD_TIME`: Time to hold at minimum load in seconds (default: 300)
- `MIN_LOAD_PERCENT`: Minimum load percentage to shed down to (default: 5)
- `END_AFTER`: End resource group simulation after specified seconds (default: 2640)
- `MAX_RGS`: Maximum total resource groups to create (default: 10)
- `MAX_PARALLEL`: Maximum concurrent resource groups (default: 3)
- `WORKLOAD`: Surrogate workload name (default: `nemotron-12b`)

3.3.1.6.3.21 GPU Power Limit Control

The Workload-Aware profile supports direct GPU power limit manipulation through two methods.

3.3.1.6.3.22 Using the Redfish API (BMC Simulator)

Sets the total node power limit by patching the Redfish `EnvironmentMetrics` `PowerLimitWatts` property directly on the BMC simulator. Power is distributed evenly across all GPUs.

```
# Set total node power limit to 2800W (700W per GPU for 4 GPUs)
task sim:gpu-power-limit NODE=gb300-r01-0001 -- --power-limit 2800
```

3.3.1.6.3.23 Using the DPS API

Sets the total node power limit using the DPS `gpu-policy` command (`UpdateGPUPolicies` API). No resource group is required. Changes propagate through DPS to the BMC simulator. Power is distributed evenly across all GPUs.

```
# Set total node power limit to 2800W (700W per GPU for 4 GPUs)
task sim:gpu-policy NODE=gb300-r01-0001 -- --power-limit 2800
```

3.3.1.6.3.24 Surrogate Environment Configuration

The Workload-Aware profile uses `environments/surrogate/values.yaml` to configure the SDK deployment. This file controls which BMC simulator profiles are active and how the surrogate workload model is configured.

3.3.1.6.3.25 Key Configuration Settings

The surrogate values file makes the following changes compared to the default deployment:

- **Disables all default BMC mock profiles** (`h100Mocks`, `b200Mocks`, `b300Mocks`, `gb200Mocks`, `gb300Mocks` are all set to `enabled: false`)
- **Enables the surrogate mock profile** (`surrogateMocks.enabled: true`)
- **Configures the BMC simulator** to use the GB300 mockup directory and the surrogate metrics plugin
- **Lists all 144 nodes** from the topology as surrogate-enabled nodes

3.3.1.6.3.26 Surrogate Plugin Settings

The surrogate BMC simulator is configured with:

```
dps-bmc-simulator:
  dpsBmcSimulator:
    surrogateMocks:
      enabled: true
      ssl: true
      debug: true
      mockupDir: /app/mockups/gb300
      ingress:
        enabled: true
        hostname: bmc-sim.dps.sdk
      surrogate:
        workloadModel:
          enabled: true
          image: "registry:5001/dps-example-workload-model:latest"
      nodes:
        - gb300-r01-0001
        - gb300-r01-0002
        # ... all 144 nodes
```

The `surrogateMocks` section enables the surrogate plugin, which connects to the workload model sidecar over gRPC. The workload model runs as a sidecar container in the same pod when configured through the Helm chart.

3.3.1.6.3.27 Customizing the Configuration

To override surrogate settings, create a custom values file and deploy with:

```
task deploy:custom VALUES_FILE=path/to/my-surrogate-values.yaml
```

3.3.1.6.3.28 Operations

3.3.1.6.3.29 Restart the BMC Simulator

If the workload-aware BMC simulator needs to be restarted (for example, after a configuration change):

```
task simulator:surrogate:restart-bmc
```

3.3.1.6.3.30 Access the BMC Simulator Admin API

When the surrogate ingress is enabled, the BMC simulator admin API is available at:

```
https://bmc-sim.dps.sdk/api/v1/admin/Plugins/Surrogate/Status
```

This endpoint returns the current plugin status, active workloads, and model information.

3.3.1.6.3.31 Monitoring

The Workload-Aware profile shares the same monitoring stack as the default profile:

- **Grafana**: `http://grafana.dps.sdk` (admin/dps)
- **Prometheus**: `http://prometheus.dps.sdk`

The Grafana dashboards display GPU power traces generated by the workload model. With the default example model you will see sin-shaped curves; with a custom model the traces will reflect your model's power predictions.

3.3.1.6.3.32 Comparison: Hardware Emulation Versus Workload-Aware

Aspect	Hardware Emulation	Workload-Aware
Metrics source	Pseudorandom values based on device characteristics	Power traces from a pluggable workload model (example: sin-shaped demo; replace with your own ML model for realistic traces)
Power phases	Not modeled	Determined by the workload model (custom models can preserve startup, compute, idle, and cooldown phases)
Duration awareness	Fixed or random duration	Determined by the workload model (custom models can scale duration with power cap)
Deploy command	`task sdk`	`task deploy:surrogate`
Simulate command	`task sim`	`task sim:surrogate`
Use case	General API testing, integration development	Power-aware analysis, workload optimization, custom model development

3.3.1.6.4 Rehearse the MaxLPS Power Pilot With the SDK Simulator

Use the DPS SDK Pilot profile to practice the MaxLPS power pilot without physical hardware. Deploy a local Kubernetes cluster, configure DPS, replay a power trace, and compare simulated GPU power across a three-rack baseline and a four-rack managed run.

This guide complements *Run the MaxLPS Power Pilot*, which remains the procedure for evaluating real systems. For the power-management model, refer to *MaxLPS Power Management*.

3.3.1.6.4.1 What the Rehearsal Measures

The SDK runs DPS and Power Steering against simulated hardware. Use the following boundaries when interpreting results.

Evidence	What You Can Learn
DPS application telemetry	Observe actual API health, topology state, resource-group membership, provisioned load, Redfish requests, and control-loop activity.
Power Steering telemetry	Observe actual controller scheduling, allocations, errors, and limit requests and confirmations against simulated endpoints.
GPU and node power, utilization, and limits	Observe values produced by the selected simulation model.

The built-in `nemotron-12b` workload replays a power trace. It does not launch an inference model. Job completion and simulated GPU utilization do not measure tokens per second, time to first token, or request latency.

A completed rehearsal demonstrates the configuration workflow and control behavior. The comparison shows three racks with workload activity in the baseline and four under MaxLPS, against the same configured power envelope.

3.3.1.6.4.2 Step 1: Obtain and Deploy the Pilot Profile

Download and extract the DPS SDK customer package from the NVIDIA NVOnline Portal. Run the commands in this guide from the extracted SDK directory.

Prepare your host before deployment:

- Use Linux (Ubuntu or Debian) or macOS, with Task installed and a running Docker daemon accessible to your user.
- Allow dependency downloads and access to the container and chart registries referenced by the package, including `nvcr.io`. Resolve any image-pull or registry authentication errors before continuing.
- Allow the setup tasks to install tools and configure `/etc/hosts` with elevated privileges. macOS setup uses Homebrew.
- Make local ports `80`, `443`, and `5001` available for the cluster and registry. The workload helper also uses port `18443` for a temporary simulator connection.
- Install `curl` and `jq` separately if needed. `task setup` does not install these helper dependencies.
- Provide enough Docker memory, CPU, and disk for the monitoring stack, DPS, and simulator. The general SDK requirements in *SDK Simulator User Guide* are a starting point; a Pilot-specific minimum has not been established. Check for pending Pods, memory pressure, and restarts.

Install the SDK dependencies first:

```
task setup
```

Setup installs or checks Docker, `kubectl`, K3D, Helm, `uv`, and the Docker-based `dpsctl` wrapper. On Linux, the wrapper is installed in `/usr/local/bin`. On macOS, add its installation directory to your shell path before deployment:

```
export PATH="$HOME/.local/bin:$PATH"
```

Confirm the tools and deploy the Pilot profile:

```
command -v task docker kubectl helm dpsctl curl jq
docker info
task deploy:pilot
task simulator:wait-for-dps-server
```

The deployment creates the `dps-sdk` K3D cluster and installs DPS, Power Steering, a BMC simulator with a workload-model sidecar, Prometheus, Grafana, and the web UI. It does not import the pilot topology or run the paired workloads for you.

Configure the CLI endpoint and authenticate explicitly:

```
export DPSCTL_HOST=api.dps.sdk
export DPSCTL_PORT=80
export DPSCTL_INSECURE_TLS_SKIP_VERIFY=true
dpsctl login --username dps
```

Enter the local SDK password, `dps`. These connection settings apply to the local SDK deployment. The packaged wrapper runs the CLI image through Docker and maps SDK ingress hostnames inside the container.

Check the deployment and record the server version:

```
kubectl --context k3d-dps-sdk get pods -n dps
dpsctl server-version
dpsctl verify
```

Inspect reported status and diagnostics, including any `status.ok` field. Confirm that the server, agent, authentication, database, simulator, and monitoring components are available. Record the package version, CLI image tag from `scripts/dpsctl-wrapper.sh`, and deployed component images with your results.

The SDK does not deploy NVIDIA Base Command Manager, so its skipped connectivity check is expected. Verification can also report a missing `dps-ui` ingress because the SDK chart names it `dps-sdk-ui`. Check the actual ingresses and open the UI to distinguish that naming mismatch from a failure. Do not ignore API, authentication, database, or Pod readiness failures.

Use the following local services throughout the rehearsal.

Service	Address	Credentials
DPS API	`api.dps.sdk:80`	`dps` / `dps`
DPS web UI	Local UI	`dps` / `dps`
Grafana	Local Grafana	`admin` / `dps`
Prometheus	Local Prometheus	None

If a service does not resolve, inspect the cluster ingresses and rerun the hostname configuration task:

```
kubectl --context k3d-dps-sdk get ingress -n dps
task deploy:etc:hosts
```

3.3.1.6.4.3 Step 2: Import and Activate the Pilot Topology

The Pilot profile uses the following inventory and configured limits.

Item	Configuration
Simulated hardware	Four GB300 racks, each with 18 compute nodes and 72 GPUs.
Total inventory	72 compute nodes and 288 GPUs. Names follow `gb300-nvl-<rack>-compute<node>`, from rack `001` through `004`.
Baseline workload	54 nodes and 216 GPUs across racks `001` through `003`. Rack `004` remains idle.
Managed workload	All 72 nodes and 288 GPUs across racks `001` through `004`.
Root domain	`PDU-Root`, with `OperatingLimit.PowerValue` of 405,000 W.
Rack limits	135,000 W each, represented by `FloorPDU-Rack01` through `FloorPDU-Rack04`.
Static load	7,416 W per rack, included in the topology budget.

The four racks share a root envelope equal to three rack limits.

Validate, import, and activate the topology:

```
dpsctl topology validate sim/maxlps-pilot.json
dpsctl topology import sim/maxlps-pilot.json
dpsctl topology activate --topology maxlps-pilot
dpsctl topology list --active
```

Confirm that `maxlps-pilot` is active and that each operation reports success. If import reports an existing topology, follow *Step 9: Retain Results and Clean Up* before starting a new paired run.

Check the inventory and simulated node connectivity:

```
dpsctl topology list-entities
dpsctl check connection --topology maxlps-pilot
```

Resolve inaccessible nodes or simulator DNS errors before proceeding.

3.3.1.6.4.4 Step 3: Confirm Monitoring With the Topology Active

Open Prometheus and check its scrape targets. Confirm that DPS, Power Steering, and the simulator are being scraped successfully. Then run the following expressions in the Prometheus query view to verify the data required for the comparison.

1. Confirm that the topology is present and active:

```
topology_service_active_status{topology_name="maxlps-pilot"}
```

2. Confirm that 72 distinct nodes have topology metadata:

```
count(count by (device_id) (device_total_power_watts{
  device_type="ComputerSystem",topology_name="maxlps-pilot"
}))
```

3. Confirm that 288 GPU power series are available:

```
count(max by (Hostname, gpu, UUID) (DCGM_FI_DEV_POWER_USAGE{
  Hostname=~"gb300-nvl-00[1-4]-compute.*"
}))
```

4. Confirm utilization samples for all four racks, including idle GPUs:

```
DCGM_FI_DEV_GPU_UTIL{Hostname=~"gb300-nvl-00[1-4]-compute.*"}
```

5. Confirm a root target of 405,000 W without a schedule or target override:

```
nvgrid_system_status_load_target_watts{feed_tag="PDU-Root"}
```

Allow several collection cycles after activation. The Pilot values configure a 5-second DPS scrape interval, a 10-second simulator scrape interval, and 5-second topology, grid, and node polling. Verify Power Steering's actual scrape interval in Prometheus. Healthy scrape targets do not guarantee that every required metric or dashboard query is populated.

Use the DPS Server Health Dashboard for application diagnostics. The Datacenter and Resource Group dashboards are optional views for inventory, allocation, and membership checks. Refer to *DPS Sample Dashboards* for query and power-boundary details.

3.3.1.6.4.5 Step 4: Deactivate and Restore Simulator Defaults

Remove DPS from the power-control path before the baseline:

```
dpsctl topology deactivate --topology maxlps-pilot
dpsctl topology list --active
dpsctl resource-group list --active
```

Confirm that `maxlps-pilot` is absent from the active topology list and no resource group manages the pilot nodes. The simulator can retain power limits written during activation. Deactivation alone does not establish an unmanaged baseline.

Restart the Pilot simulator to restore its defaults:

```
kubectl --context k3d-dps-sdk rollout restart \
  deploy/bmc-surrogate-simulator-pilot -n dps
kubectl --context k3d-dps-sdk rollout status \
  deploy/bmc-surrogate-simulator-pilot -n dps --timeout=240s
```

After the simulator becomes ready, wait for fresh scrapes from the replacement Pod. Query the effective GPU limits in Prometheus:

```
DCGM_FI_DEV_POWER_MGMT_LIMIT{Hostname=~"gb300-nvl-00[1-4]-compute.*"}
```

The bundled GB300 simulator defaults to 1,400 W per GPU. Confirm that all 288 GPU limits report that value, rather than retained managed caps. Exclude old Pod series or wait for them to become stale before counting samples. Check GPU power, utilization, and DPS node metadata again with the expressions in *Step 3: Confirm Monitoring With the Topology Active*.

If limits remain capped, confirm that the topology is inactive and no other resource group or simulation is writing limits. Resolve that condition and repeat the reset before collecting a baseline. This reset is the simulator equivalent of restoring default GPU limits in the hardware procedure.

3.3.1.6.4.6 Step 5: Prepare Comparable Observation Windows

Create a rehearsal record before starting the workloads. Retain the following information for both phases:

- Package, CLI, server, controller, simulator, and dashboard versions.
- Topology JSON, Pilot values, root and rack limits, and the `MaxLPS` bundle.
- Trace name (`nemotron-12b`), `TIME_SCALE`, job identifiers, and node sets.
- Collection and scrape intervals, workload start and end times, and selected observation windows in UTC.
- Grafana datasource, topology, comparison offset, and active-GPU threshold.

Keep the trace, time scale, topology envelope, and collection settings consistent across both runs. The 54-node and 72-node replays exercise different simulated fleet sizes. They do not establish matched aggregate inference request demand.

The following commands use the task default, `TIME_SCALE=1`. A larger value accelerates replay. At `TIME_SCALE=1`, each run takes about 30 minutes. At `TIME_SCALE=20`, each run takes about 1 to 2 minutes. Choose a value that leaves enough time to observe metrics and controller behavior, and use it for both runs. The helper polls progress every 10 seconds and defaults to a 3-hour completion timeout. Record the baseline window during the run so you can select a matching managed window.

Use the same measured power envelope in both phases: GPU draw across all 72 topology nodes, including idle GPUs in rack four during the baseline. CPU power and static rack loads are outside the dashboard's GPU draw, although they consume part of the modeled topology budget.

3.3.1.6.4.7 Step 6: Run the Three-Rack Baseline

Recheck that the topology is inactive, no pilot group is active, and simulator limits are restored. Record a UTC timestamp and start the baseline:

```
date -u '+%Y-%m-%dT%H:%M:%SZ'
task pilot:workload:baseline TIME_SCALE=1 JOB_ID=pilot-baseline
date -u '+%Y-%m-%dT%H:%M:%SZ'
```

The helper submits the trace to 54 nodes across racks `001` through `003` and waits for completion. Expect confirmation that 54 nodes were accepted, progress updates, and a completion message.

If submission fails, check simulator readiness, `curl`, `jq`, port 18443, and the reported HTTP response. If progress times out, retain diagnostics and inspect the workload and caps before repeating.

Each submission helper stops existing simulator jobs before starting its job. Run the phases sequentially, without another simulation in parallel. If metrics are absent or caps were retained, correct the issue and repeat the baseline before proceeding.

3.3.1.6.4.8 Step 7: Enable MaxLPS and Run the Managed Phase

Reactivate the topology and inspect the server's bundled policy:

```
dpsctl topology activate --topology maxlps-pilot
dpsctl topology list --active
dpsctl policy bundle list
dpsctl policy bundle get MaxLPS
```

Confirm successful activation, then create, populate, and activate the resource group:

```
dpsctl resource-group create --resource-group maxlps-pilot --external-id 1 \
  --dpm-enable --prs-enabled --shared-gpu-enable --policy-bundle MaxLPS
dpsctl resource-group add --resource-group maxlps-pilot \
  --entities "$(task pilot:nodes:72)"
dpsctl resource-group activate --resource-group maxlps-pilot \
  --allow-reprovision --sync
dpsctl resource-group list --active
```

Inspect status and diagnostics from each command. Confirm the following fields for the active group before submitting the workload.

Field	Expected Value
`group_name`	`maxlps-pilot`
`activation_status`	`active`
`policy_bundle`	`MaxLPS`
`dpm_enable`, `prs_enabled`, `shared_gpu_enable`	All `true`
`resource_names`	All 72 pilot compute nodes.

Run the managed phase with the same time scale as the baseline:

```
date -u '+%Y-%m-%dT%H:%M:%SZ'
task pilot:workload:managed TIME_SCALE=1 JOB_ID=pilot-managed
date -u '+%Y-%m-%dT%H:%M:%SZ'
```

Confirm acceptance and completion for 72 nodes. Preserve the run output and group configuration with your timestamps.

3.3.1.6.4.9 Step 8: Compare the Results in Grafana

After both runs, open **DPS MaxP Window Comparison** in Grafana. **Comparison** means the earlier baseline window. **Current** means the selected managed window, even after both runs have finished.

Configure the page with your recorded windows:

1. Select the provisioned Prometheus datasource under **Metrics source**.
2. Select **maxlps-pilot** under **Datacenter / topology**.
3. Set Grafana's time picker to an absolute range covering the managed observation window. Use UTC to match your recorded timestamps.
4. Set **Comparison window** to the positive time difference between the managed and baseline window starts. The entire current range shifts backward by this offset. Verify that shifting the managed end by the same amount reaches the recorded baseline end.
5. Set **Active GPU threshold** to **More than 5%**. The count uses each GPU's average utilization over the window. **Any activity** can include the simulated idle fourth rack because idle utilization is nonzero.
6. Confirm that both sets of summary cards, node power charts, and four named rack bars populate. Save the dashboard's absolute URL and a screenshot.

The comparison offset is the difference between window starts. It is not the baseline duration or the gap between runs. The dropdown defaults to one day and accepts custom Prometheus durations, such as `2m` or `125s`.

The following example illustrates equal-duration windows separated by a 2-minute offset. Use your own timestamps and corresponding loaded trace portions; these example times do not identify an acceptance window.

Selection	Example Value
Baseline observation	October 14, 9:04:30 p.m. to 9:05:30 p.m. UTC.
Managed observation and Grafana absolute range	October 14, 9:06:30 p.m. to 9:07:30 p.m. UTC.
Comparison window	`2m`
Datacenter / topology	`maxlps-pilot`
Active GPU threshold	**More than 5%**

You can also supply the custom offset as `var-baseline_offset` in the saved dashboard URL. For this example, retain the following query parameters, alongside the absolute `from` and `to` parameters and datasource selection:

- `var-topology=maxlps-pilot`
- `var-baseline_offset=2m`
- `var-active_gpu_util_threshold=5`

3.3.1.6.4.10 Interpret the Power Envelope

Use the following definitions when reading the comparison.

Signal	Meaning
Power envelope	Operating target from the Pilot's single `PowerDomain`, `PDU-Root`.
GPU draw and node power charts	`DCGM_FI_DEV_POWER_USAGE` across all 288 GPUs. A node chart sums its GPU draw, rather than total BMC-reported node power.
Active GPUs	Count of GPUs whose window-average `DCGM_FI_DEV_GPU_UTIL` exceeds the selected threshold.
Mean power utilization	Mean GPU draw divided by the root envelope. It is not a percentage of GPU maximum power or an applied GPU cap.
Mean power headroom	Root envelope minus mean GPU draw. CPU power and rack static loads still consume part of the modeled budget.
Observed power change by rack	Relative change in mean GPU draw from the baseline, displayed as a percentage. An initially idle rack can have a large relative increase.
Power utilization change by rack	Change in mean GPU draw divided by that rack's 135 kW target, in percentage points. The panel displays a percent symbol.
Active GPU change by rack	Absolute count change. The fleet **Active GPU change** card instead shows a percentage.

The peak utilization value sums individual GPU peak powers, which need not occur simultaneously. It is not an exact simultaneous fleet peak.

3.3.1.6.4.11 Check the Rehearsal Outcome

Use the following observations together with successful workload completion and verified resource-group configuration.

Observation	Expected Result in Loaded Windows
Power envelope	405 kW in both windows. In the inactive baseline, this is a reference target rather than evidence of enforcement.
Active GPU counts	**Comparison active GPUs** is 216, **Current active GPUs** is 288, and **Active GPU change** is approximately +33.3%.
Rack activity	Four named rack bars populate. **Active GPU change by rack** is zero for racks one through three and +72 for rack four.
Simulated power behavior	Rack four contributes loaded GPU draw in the managed window. The first three racks can draw less under managed caps.

Exact power changes depend on trace phase, caps, and window selection. Do not require a particular reduction percentage.

Use the following checks for unexpected results.

Symptom	Recovery
Empty comparison cards	Check the absolute range, offset, and availability of baseline telemetry. The default 1-day offset usually does not reach a run you just recorded.
Empty current cards	Check the managed window, datasource, topology, and the required metric series from Step 3.
Equal active GPU counts	Select **More than 5%**, inspect the loaded windows, and verify which node set received each workload.
Missing rack bars or wrong topology	Check topology selection, DPS node metadata, hostname matching, and the packaged dashboard version.
Zero or unexpected envelope	Check PDU-`Root`, schedules, target overrides, and root-domain metadata.
Power appears capped in the baseline	Restore simulator defaults and repeat the baseline, then select a matching managed window.

3.3.1.6.4.12 Step 9: Retain Results and Clean Up

If desired, preserve the configuration and version record, workload output, active group status, membership, and both run timestamps. Retain the selected observation windows, computed offset, Grafana variables, saved absolute URL, and a screenshot or exported comparison. A dashboard view does not itself retain an evidence package.

Delete the resource group, then deactivate the topology:

```
dpsctl resource-group delete --resource-group maxlps-pilot
dpsctl topology deactivate --topology maxlps-pilot
dpsctl resource-group list --active
dpsctl topology list --active
```

Confirm success in the output and active lists. For another rehearsal, repeat the simulator reset in *Step 4: Deactivate and Restore Simulator Defaults* before collecting a new baseline. You can reuse the imported topology; skip its import step on a repeat run.

When you finish with the local environment, optionally remove the SDK cluster. This deletes its workloads and locally retained monitoring data:

```
task deploy:cluster:delete
```

3.3.1.6.4.13 Optional Follow-On: Move to the Hardware Pilot

Use *MaxLPS Power Pilot* to plan the real-system evaluation and *Run the MaxLPS Power Pilot* to execute it.

3.3.2. Administrator Guides

Documentation and operational guides for DPS system administrators.

3.3.2.1 Operations

This section provides operational guides for DPS administrators, covering deployment, configuration, and advanced management tasks.

3.3.2.1.1 Getting Started

These guides provide operational information for DPS administrators, covering deployment, configuration, and advanced management tasks.

- ***DPS Architecture*** - Comprehensive overview of DPS system architecture, components, deployment model, and integrations

3.3.2.1.1.1 Topology and Infrastructure

- ***PDN Topology Processing*** - Guide for processing engineering diagrams into Power Distribution Network (PDN) topology configurations

3.3.2.1.1.2 Integration

- ***Nautobot Integration*** - Guide for integrating DPS with Nautobot DCIM to import device inventory for power management

3.3.2.1.1.3 Power Management

- ***Power Policies Guide*** - Comprehensive guide to creating, applying, and enforcing DPS power policies for effective datacenter power management
- ***Power Compliance*** - Detailed explanation of how DPS ensures power compliance through topology validation, resource group management, and automatic policy adjustment

3.3.2.1.2 Overview

These guides are designed for system administrators who need to deploy, configure, and maintain DPS in production environments. Each guide provides step-by-step instructions, best practices, and troubleshooting information for successful DPS operations.

3.3.2.1.2.1 DPS Architecture

3.3.2.1.2.2 Overview

The Dynamic Power Software (DPS) is a power management solution designed as a cloud-native Kubernetes service. This document describes the overall architecture, core components, deployment model, and system integrations that make up the DPS ecosystem.

DPS provides centralized power management for datacenter infrastructure through a microservices architecture deployed on Kubernetes. The system manages power policies, monitors consumption, and controls hardware through standardized protocols.

3.3.2.1.2.3 Key Architectural Principles

- **Cloud-Native Design**: Built for Kubernetes deployment with containerized microservices
- **API-First Architecture**: All functionality exposed through gRPC APIs with HTTP/REST gateway
- **Protocol Standards**: Uses industry-standard Redfish protocol for hardware control
- **Scalable Design**: Horizontally scalable components with persistent storage
- **Authentication**: LDAP, OIDC, service-specific mTLS, or an external authentication proxy

3.3.2.1.2.4 Core Components

3.3.2.1.2.5 DPS Server (`dps-server`)

The core orchestration engine of the DPS system.

Functionality:

- **gRPC API Server**: Primary interface for all DPS operations
- **Topology Management**: Manages power distribution network topology
- **Policy Engine**: Implements and enforces power policies
- **Resource Group Management**: Handles dynamic resource allocation
- **BMC Integration**: Communicates with Baseboard Management Controllers via Redfish

Key Responsibilities:

1. **API Management**: Serves gRPC API for client interactions
2. **Topology Validation**: Ensures power topology constraints are satisfied
3. **Hardware Control**: Manages BMC connections and Redfish communications
4. **Policy Enforcement**: Applies and monitors power policies across resources
5. **Database Operations**: Manages persistent state in PostgreSQL

3.3.2.1.2.6 DPS Control Utility (`dpsctl`)

Command-line interface for DPS operations.

Architecture:

- **gRPC Client**: Communicates with dps-server via gRPC protocol
- **Command Parser**: Processes CLI commands and arguments
- **Output Formatter**: Provides human-readable and machine-parseable output

3.3.2.1.2.7 DPS Web UI (`dps-ui`)

Web-based user interface for DPS management.

Functionality:

- **Topology Visualization**: Interactive topology creation and editing
- **Policy Management**: Web-based policy configuration interface
- **Telemetry Dashboard**: Real-time power consumption monitoring
- **Import/Export**: Topology and configuration management tools

Integration:

- **DPS Server Communication**: RESTful API calls to dps-server HTTP gateway
- **Authentication**: Integrates with DPS authentication system

3.3.2.1.2.8 Data Storage

3.3.2.1.2.9 PostgreSQL Database

Primary persistent storage for DPS state and configuration.

Schema Components:

- **Topologies**: Power distribution network definitions
- **Policies**: Power management policy configurations
- **Resource Groups**: Dynamic resource allocations
- **Entities**: Hardware component specifications

3.3.2.1.2.10 Authentication and Authorization

DPS supports the following authentication setups:

- **LDAP**: The server checks directory credentials and issues tokens for API requests.
- **OpenID Connect (OIDC)**: The identity provider issues tokens, and the server validates them against the configured issuer, audience, and scopes.
- **Mutual TLS (mTLS)**: Verified client certificates authenticate callers to services with a configured certificate common name allowlist.
- **External Authentication and Authorization**: An external proxy runs as a sidecar in the DPS server Pod. It authenticates callers, authorizes requests, and forwards permitted calls to the server. Refer to *Configure External Authentication and Authorization* for configuration. Built-in server authentication and authorization are disabled in this mode.

Refer to *Configure Server Authentication* to choose and configure a mode. Configure secure transport and Secrets separately through *Prepare a Secure Deployment*.

The UI and `dpsctl` use the login method configured for the authentication mode. The API enforces access through the server or external proxy. Authentication confirms the caller's identity; authorization determines whether its request is allowed.

3.3.2.1.2.11 Hardware Integration

3.3.2.1.2.12 BMC Communication

DPS controls hardware through Baseboard Management Controllers using the Redfish protocol.

Protocol Stack:

```
DPS Server

Redfish API (HTTPS)

BMC Controllers

Hardware Components
```

Supported Hardware:

- **Real BMCs**: Production server BMCs supporting Redfish
- **BMC Simulators**: Development and testing environments

Connection Management:

- **Credential Storage**: BMC credentials stored in Kubernetes secrets
- **Connection Pooling**: Efficient BMC connection management
- **Health Monitoring**: BMC availability and status tracking
- **Error Handling**: Robust error recovery and retry logic

3.3.2.1.2.13 BMC Simulation System

Development and testing infrastructure simulating NVIDIA DGX systems.

Simulated Systems:

- **H100 Systems**: NVIDIA H100 GPU-based servers
- **B200 Systems**: NVIDIA B200 GPU-based servers
- **B300 Systems**: NVIDIA B300 GPU-based servers
- **GB200 Systems**: NVIDIA GB200 Grace Blackwell systems

Simulator Features:

- **Redfish Compliance**: Partial Redfish API implementation
- **Realistic Behavior**: Accurate power consumption modeling
- **Dynamic Response**: Real-time power limit changes
- **Monitoring Integration**: Prometheus metrics export

3.3.2.1.2.14 Python API Integration

Python API for programmatic DPS interaction.

API Features:

- **Complete Coverage**: All DPS functionality accessible via Python
- **Type Safety**: Strongly typed interfaces with validation
- **Async Support**: Asynchronous operation support
- **Error Handling**: Exception handling

3.3.2.1.2.15 Deployment Architecture

3.3.2.1.2.16 Kubernetes Deployment Model

DPS deploys as a collection of Kubernetes services using Helm charts.

Namespace Organization:

```
dps (namespace)
    dps-server (deployment + service)
    dps-ui (deployment + service)
    postgresql (statefulset + service)
    openldap (deployment + service)
    bmc-simulators (deployment + service)
```

3.3.2.1.2.17 Security Architecture

3.3.2.1.2.18 Network Security

Service Isolation:

- **Network Policies**: Kubernetes network policies restrict inter-pod communication
- **TLS Encryption**: All external communications encrypted with TLS
- **Internal Communication**: Secure internal service mesh

3.3.2.1.2.19 Authentication Security

Use separate identities for operators and services. For automation, use LDAP service accounts, OIDC client credentials, or mTLS configured for the required service. Refer to *Service Accounts*.

BMC credentials authenticate hardware connections separately from API callers. Store them through the configured credential store, as described in *Credentials and Secrets Configuration*.

3.3.2.1.2.20 Monitoring & Observability

3.3.2.1.2.21 Metrics Collection

Prometheus Integration:

- **Service Monitors**: Automatic service discovery for metrics
- **Custom Metrics**: DPS-specific power consumption metrics
- **Hardware Metrics**: BMC-sourced telemetry data

3.3.2.1.2.22 Logging Architecture

Structured Logging:

- **JSON Format**: Machine-parseable log formats
- **Correlation IDs**: Request tracing across components
- **Log Levels**: Configurable verbosity levels

3.3.2.1.2.23 Tracing Integration

Distributed Tracing:

- **Tempo Integration**: Request tracing across microservices
- **Span Collection**: Detailed operation timing
- **Performance Analysis**: Bottleneck identification

3.3.2.1.2.24 Power Policy Administration

The authoritative policy documentation is *Power Policies*. It defines current policy bundles, topology and resource-group selection, composition, idle policy, and plugin-specific fallback behavior.

This SDK page is retained as a stable entry point and does not duplicate the policy model. Use these current resources:

- *Power Policies* for behavior and lifecycle.
- *Topologies* for the topology import boundary.
- *Resource Groups* for resource-group precedence.
- *Manage Policy Bundles and Set the Global Default* for the UI workflow.
- *bundle* for the Go `dpsctl` bundle commands.

Current Go CLI documentation uses `dpsctl policy bundle` and `--policy-bundle`. Legacy examples that show `policy` commands without the `bundle` subgroup, or resource-group `--policy` and `--entity-policy` flags, do not describe the current Go CLI.

3.3.2.1.2.25 Overview

This guide provides instructions for converting engineering one-line diagrams that show datacenter power connectivity into Power Distribution Network (PDN) topology that can be consumed by the Dynamic Power Software (DPS). One-line diagrams are standard electrical engineering drawings that illustrate the single-phase equivalent of a three-phase power system, showing all major components and their interconnections in a simplified, schematic format.

> **Alternative Approach**: All manual topology entity creation and relationship definitions described in this guide can also be managed through the DPS user interface. For a graphical approach to topology management, see the *Topology UI Guide*.

3.3.2.1.2.26 Understanding One-Line Diagrams

3.3.2.1.2.27 What is a One-Line Diagram?

A one-line diagram (also called a single-line diagram) is a simplified notation for representing a three-phase power system. It shows:

- **Power Sources**: Utility feeds, generators, UPS systems
- **Distribution Equipment**: Transformers, switchgear, panels, PDUs
- **Protection Devices**: Circuit breakers, fuses, disconnect switches
- **Load Equipment**: Servers, networking equipment, storage systems
- **Power Flow Paths**: How electricity flows from source to loads

3.3.2.1.2.28 Key Components in Datacenter One-Line Diagrams

1. **Utility Feed**: Primary power source from the electrical grid
2. **Main Distribution Panels (MDP)**: High-voltage distribution points
3. **Transformers**: Voltage step-down equipment
4. **Switchgear**: Protection and switching equipment
5. **Floor PDUs**: Building-level power distribution units
6. **Rack PDUs**: Rack-level power distribution units
7. **Power Supply Units (PSUs)**: Server-level power conversion
8. **Compute Nodes**: End-user equipment (servers, switches, storage)

3.3.2.1.2.29 DPS Topology Structure

3.3.2.1.2.30 Hierarchical Power Distribution Model

DPS models datacenter power infrastructure as a hierarchical tree structure where:

- **Root Nodes**: Utility feeds or primary power sources (power domains)
- **Intermediate Nodes**: Distribution equipment (transformers, PDUs, switchgear)
- **Leaf Nodes**: End-consuming devices (compute nodes)

3.3.2.1.2.31 Entity Types in DPS

DPS is using DMTF/Redfish terminology and supports the following entity types for power infrastructure:

Entity Type	Description	Examples
`PowerDomain`	Power domain	Utility feed or any other resource, constrained or unconstrained
`PowerDistribution`	Power distribution equipment	Floor PDUs, Rack PDUs
`PowerSupply`	Power conversion units	Server PSUs, UPS systems
`ComputerSystem`	Computing equipment	Servers, workstations

3.3.2.1.2.32 Step-by-Step Conversion Process

3.3.2.1.2.33 Step 1: Analyze the One-Line Diagram

1. **Identify Power Flow Direction**: Start from utility feeds and trace power flow to end loads
2. **Catalog All Components**: List every piece of equipment shown in the diagram
3. **Map Power-Relevant Connections**: Document how components with power limits connect to others in the power distribution hierarchy
4. **Note Power Ratings**: Record power limits (wattage) where specified

3.3.2.1.2.34 Step 2: Create Entity Inventory

Important: Not all components must be included. It makes sense to include components that have power limits, consume power, power management capabilities, or serve as power distribution points. Components that are purely electrical, do not have any power limits and do not support power management features can be omitted from the DPS topology.

Note: The examples used in this guide are simplified. Their purpose is to demonstrate how to create a functional topology file. In real-world scenarios, your topology will be significantly more complex with many more entities, detailed power specifications, and comprehensive relationship mappings.

For each power-relevant component in the diagram, collect the following information:

3.3.2.1.2.35 Required Information

- **Type**: DPS entity type (PowerDomain, PowerDistribution, PowerSupply, ComputerSystem)
- **Name**: Human readable unique identifier
- **Parent Connections**: What power source feeds this component
- **Child Connections**: What power loads this component feeds

3.3.2.1.2.36 Optional Information

- **Power Limit**: Power limit of this component
- **Static Load**: Fixed power consumption (in watts)
- **Model**: DPS predefined device model identifier
- **Redfish Endpoint**: BMC management interface (currently for specific NVIDIA compute nodes only)

3.3.2.1.2.37 Step 3: Build the Topology JSON Structure

Create a JSON file with the following structure:

```
{
  "Entities": [
    // Individual entity definitions
  ],
  "Topology": {
    "Name": "pdn",
    "Entities": [
```

(continues on next page)

(continued from previous page)

```
      // Topology relationship definitions
    ]
  }
}
```

3.3.2.1.2.38 Step 4: Map Components to DPS Entities

3.3.2.1.2.39 Power Domain

For utility feeds or primary power sources with an operating limit:

```
{
    "Type": "PowerDomain",
    "Name": "PD-A",
    "OperatingLimit": {
        "PowerValue": {
            "Value": 10000,
            "Type": "W"
        }
    }
}
```

This entity defines a power domain with a 10,000 W operating limit. The limit caps the aggregate power consumption of its children.

3.3.2.1.2.40 Power Distribution Equipment

For floor PDUs, transformers, and switchgear:

```
{
    "Type": "PowerDistribution",
    "Model": "FloorPDU95",
    "Name": "PDU"
}
```

This entity defines one possible floor PDU for power distribution. In this case, FloorPDU95 represents standard device with 0.95 efficiency factor as defined in the DPS device registry. Device registry and defining custom devices are covered in *Step 6: Add Custom Device Specifications*.

3.3.2.1.2.41 Compute Nodes

For servers and workstations:

```
{
    "Type": "ComputerSystem",
    "Name": "NODE-GB200-1",
    "Model": "DGX_GB200",
    "Redfish": {
        "URL": "https://NODE-GB200-1",
        "SecretName": "NODE-GB200-1"
    }
}
```

This entity defines a compute node with specific management capabilities. The ComputerSystem type represents end-consuming devices in the power hierarchy. The DGX_GB200 model references a predefined device specification in the DPS device registry. Besides standard Redfish definitions, Redfish section includes management endpoint information - the URL specifies the BMC endpoint for power monitoring and control, while SecretName references stored authentication credentials. This allows DPS to actively manage and monitor the server's power consumption and apply power policies directly to the hardware.

3.3.2.1.2.42 Step 5: Define Hierarchical Relationships

Establish the parent-child relationships that represent power flow:

```
"Topology": {
    "Name": "pdn",
    "Entities": [
        {
            "Name": "NODE-GB200-1"
        },
        {
            "Name": "NODE-GB200-2"
        },
        {
            "Name": "RPDU",
            "Children": [
                "NODE-GB200-1",
                "NODE-GB200-2"
            ]
        },
        {
            "Name": "PDU",
            "Children": [
                "RPDU"
            ]
        },
        {
            "Name": "PD-A",
            "Children": [
                "PDU"
            ]
        }
    ]
}
```

Policy-bundle selection is topology-wide and is not stored on these relationship entries. The topology import path resolves an empty selection to the current `global_policy_bundle`; a legacy per-entity `Policy` value does not create an activation assignment. Refer to *Power Policies* for policy selection and fallback behavior.

Note: Leaf nodes (nodes without children) must be listed as their own Entities entry.

3.3.2.1.2.43 Complete Topology Example

The following example demonstrates a complete topology JSON file that incorporates all the concepts covered in the previous steps. This example defines a simple power distribution hierarchy with:

- One power domain (utility feed) limited to 10,000 watts
- One floor PDU with 95% efficiency
- One rack PDU capable of 57,500 watts
- Two DGX GB200 compute nodes

```
{
    "Entities": [
        {
            "Type": "PowerDomain",
            "Name": "PD-A",
            "OperatingLimit": {
                "PowerValue": {
                    "Value": 10000,
                    "Type": "W"
                }
            }
        },
        {
            "Type": "PowerDistribution",
            "Model": "FloorPDU95",
            "Name": "PDU"
        },
        {
            "Type": "PowerDistribution",
            "Model": "RackPDU95_57500W",
            "Name": "RPDU"
        },
        {
            "Type": "ComputerSystem",
            "Name": "GB200-1",
            "Model": "DGX_GB200",
            "Redfish": {
                "URL": "https://GB200-1",
                "SecretName": "GB200-1"
            }
        },
        {
            "Type": "ComputerSystem",
            "Name": "GB200-2",
            "Model": "DGX_GB200",
            "Redfish": {
                "URL": "https://GB200-2",
                "SecretName": "GB200-2"
            }
        }
    ],
    "Topology": {
```

(continues on next page)

(continued from previous page)

```
        "Name": "pdn",
        "Entities": [
            {
                "Name": "GB200-1"
            },
            {
                "Name": "GB200-2"
            },
            {
                "Name": "RPDU",
                "Children": [
                    "GB200-1",
                    "GB200-2"
                ]
            },
            {
                "Name": "PDU",
                "Children": [
                    "RPDU"
                ]
            },
            {
                "Name": "PD-A",
                "Children": [
                    "PDU"
                ]
            }
        ]
    }
}
```

3.3.2.1.2.44 Key Features of This Example

1. **Hierarchical Structure**: Power flows from PD-A → PDU → RPDU → GB200 nodes
2. **Operating Limit**: The PD-A power domain limits total consumption to 10,000 W
3. **Device Models**: Uses standard DPS device models (FloorPDU95, RackPDU95_57500W, DGX_GB200)
4. **Management Endpoints**: Compute nodes include BMC URLs and authentication secrets
5. **Topology Policy Boundary**: Policy bundles are managed separately and selected for the entire topology

Note: The example intentionally contains no inline policy definitions or per-entity policy assignments. Refer to *Power Policies* for current policy-bundle management.

3.3.2.1.2.45 Step 6: Add Custom Device Specifications (Optional)

Each topology entity can be backed by a device. Each device is represented by a unique string model and can be referenced in the topology definition using the Model parameter (as can be seen in the above examples). DPS embeds a standard `devices.yaml` file that contains predefined device specifications for common datacenter equipment. These standard devices are always available and cover

most typical use cases, including NVIDIA DGX systems, standard PDUs, and common power distribution equipment. The latest `devices.yaml` file is included with the SDK as a reference.

However, if you need to define custom devices or override standard specifications, you can create your own device specification YAML file.

3.3.2.1.2.46 Standard Device Examples

Here are some examples from the standard devices.yaml that DPS provides:

ComputerSystem (DGX GB200):

```yaml
- type: ComputerSystem
  description: NVIDIA GB200 Compute Tray (Bianca)
  model: DGX_GB200
  spec:
    devices:
      - type: CPU
        model: Grace
        count: 2
      - type: GPU
        model: GB200
        count: 4
    minLoadWatts: 1000
    maxLoadWatts: 6667
    processorModulesCount: 2
    idlePolicy:
      Node:
        watts: 1000
      GPU:
        watts: 800
      CPU:
        watts: 100
```

PowerDistribution (Rack PDU):

```yaml
- type: PowerDistribution
  description: Generic Rack PDU at 57500W 95 Pct Eff
  model: RackPDU95_57500W
  spec:
    maxLoadWatts: 57500
    efficiencyFactor: 0.95
```

GPU Component:

```yaml
- type: GPU
  description: NVIDIA B200 GPU
  model: B200
  spec:
    minLoadWatts: 200
    maxLoadWatts: 1000
    wppsSupport: true
```

3.3.2.1.2.47 Field Explanations

- **type**: DPS entity type (PowerDomain, PowerDistribution, PowerSupply, ComputerSystem, GPU, CPU)
- **description**: Human-readable description of the device
- **model**: Unique identifier used to reference this device specification
- **spec**: Device-specific power and capability specifications
  - **minLoadWatts/maxLoadWatts**: Power consumption range in watts
  - **efficiencyFactor**: Power conversion efficiency (0.0-1.0)
  - **devices**: List of sub-components (for ComputerSystems)
  - **wppsSupport**: Whether device supports Workload Power Profile Settings
  - **idlePolicy**: Default power allocations for different component types
  - **processorModulesCount**: Number of processor modules (for specific systems)

3.3.2.1.2.48 Creating Custom Device Specifications

Important: When defining ComputerSystem devices, pay careful attention to the contained devices (CPU and GPU components). Each referenced device model in the `devices` list must also be defined in your device specifications file or available in the standard DPS device registry.

Create a YAML file with your custom device definitions:

```yaml
# custom-devices.yaml
- type: ComputerSystem
  description: Custom Server Configuration
  model: CustomServer_8GPU
  spec:
    devices:
      - type: GPU
        model: CustomGPU
        count: 8
      - type: CPU
        model: CustomCPU
        count: 2
    minLoadWatts: 2000
    maxLoadWatts: 12000
    idlePolicy:
      Node:
        watts: 2000
      GPU:
        watts: 1500
      CPU:
        watts: 400
      Memory:
        watts: 100

- type: GPU
  description: Custom High-Performance GPU
  model: CustomGPU
```

(continues on next page)

(continued from previous page)

```
  spec:
    minLoadWatts: 250
    maxLoadWatts: 800
    wppsSupport: true

- type: CPU
  description: Custom High-Performance CPU
  model: CustomCPU
  spec:
    minLoadWatts: 100
    maxLoadWatts: 400
    wppsSupport: true
```

3.3.2.1.2.49 Importing Custom Device Specifications

Use the DPS CLI to import your custom device specifications:

```
dpsctl device upsert custom-devices.yaml
```

3.3.2.1.2.50 Upsert Mode Behavior

Device specifications work in **upsert mode**, meaning:

- **New devices**: Custom devices with unique model names are added to the registry
- **Override existing**: If your custom device uses the same model name as a standard device, your specification will override the standard one
- **Merge capability**: This allows you to customize standard devices for your specific environment while keeping other standard devices unchanged

This flexibility enables you to adapt DPS to your specific hardware configurations while leveraging the library of standard device specifications that DPS provides out of the box.

3.3.2.1.2.51 Best Practices and Guidelines

3.3.2.1.2.52 Component Selection Criteria

When converting engineering diagrams to DPS topology, include only components that:

1. **Have Power Management**: Components with configurable power limits or power monitoring capabilities
2. **Consume Significant Power**: Equipment with meaningful power consumption (typically >100W)
3. **Act as Distribution Points**: Equipment that distributes power to multiple downstream devices
4. **Require Monitoring**: Critical infrastructure components that need power oversight

Exclude purely electrical components like:

- Circuit breakers and fuses (unless they have smart monitoring)
- Manual disconnect switches
- Basic electrical panels without power management
- Passive electrical components (cables, connectors)

3.3.2.1.2.53 Naming Conventions

Consistent naming conventions are crucial for managing large-scale DPS topologies effectively. Well-structured entity names improve readability, simplify troubleshooting, and make topology maintenance much easier as your infrastructure grows. Consistent naming also helps administrators quickly understand the role and location of components when reviewing topology configurations or investigating power-related issues.

1. **Consistent Prefixes**: You may use standard prefixes for component types
   - `PDU-` for floor PDUs
   - `RPDU-` for rack PDUs
   - `NODE-` for compute systems
2. **Location Encoding**: You can include location information in names
   - `PDU-A1-` (Building A, Floor 1)
   - `RPDU-A1-C19` (Building A, Floor 1, Cabinet 19)

3.3.2.1.2.54 Validation and Activation

3.3.2.1.2.55 Step 1: Schema Validation

Use DPS validation tools to check topology structure:

```
dpsctl topology validate datacenter-topology.json
```

Note: Validation can be performed at any time and does not import the topology into DPS.

3.3.2.1.2.56 Step 2: Import Custom Device Specifications (if needed)

If you created custom device specifications, import them first:

```
dpsctl device upsert custom-devices.yaml
```

3.3.2.1.2.57 Step 3: Import Topology

Import the topology into DPS:

```
dpsctl topology import datacenter-topology.json
```

Note: Topologies are uniquely identified by their name, and multiple topologies can be imported.

3.3.2.1.2.58 Step 4: Activate Topology

Activate the topology for power management:

```
dpsctl topology activate --topology pdn
```

Important: Only one topology can be active at a time. Deactivate the active topology before activating a different one.

3.3.2.1.2.59 Ongoing Management

After successfully activating your topology, ongoing management includes:

1. **Regular Updates**: Update topology as infrastructure changes
2. **Capacity Monitoring**: Monitor power utilization and plan for growth
3. **Policy Application**: Manage topology-wide policy bundles as described in *Power Policies*
4. **Performance Tuning**: Optimize based on actual usage patterns

3.3.2.1.2.60 Example Workflow

3.3.2.1.2.61 Sample One-Line Diagram Conversion

Consider a simple datacenter with:

- Utility feed Main switchgear 3 Floor PDUs Rack PDUs Compute nodes

The conversion process would be:

1. **Identify hierarchy levels**: Utility Floor PDUs Rack PDUs Nodes
2. **Create entity definitions** for each piece of equipment
3. **Map relationships** showing power flow from utility to compute nodes
4. **Add management endpoints** for BMC-enabled devices
5. **Assign power ratings** based on nameplate or measured data
6. **Validate, import and activate** into DPS

This structured approach ensures that the engineering diagram is accurately represented as a DPS topology that can be used for comprehensive power management and optimization.

By following this guide, administrators can successfully convert engineering one-line diagrams into DPS topology configurations that enable effective datacenter power management and optimization.

3.3.2.1.2.62 Overview

This guide provides instructions for creating and managing Power Distribution Network (PDN) topology through the Dynamic Power Software (DPS) user interface. While topology entities and relationships can be defined manually through configuration files as described in the *PDN Diagrams guide*, the web-based user interface provides an intuitive graphical approach for both creating topology entities and managing their relationships.

> **Note**: The examples and procedures in this guide will create the same topology structure and entities as demonstrated in the *PDN Diagrams guide*. Both approaches result in identical topology configurations, allowing you to choose the method that best suits your workflow: graphical UI or configuration files.

3.3.2.1.2.63 Benefits of UI-Based Topology Management

Using the user interface for topology management offers several advantages:

1. **Visual Representation**: See the power distribution hierarchy in a graphical format
2. **Ease of Use**: No need to manually edit configuration files
3. **Error Prevention**: Built-in validation prevents common configuration mistakes

4. **Collaborative Editing**: Multiple administrators can work on topology configurations

3.3.2.1.2.64 Getting Started

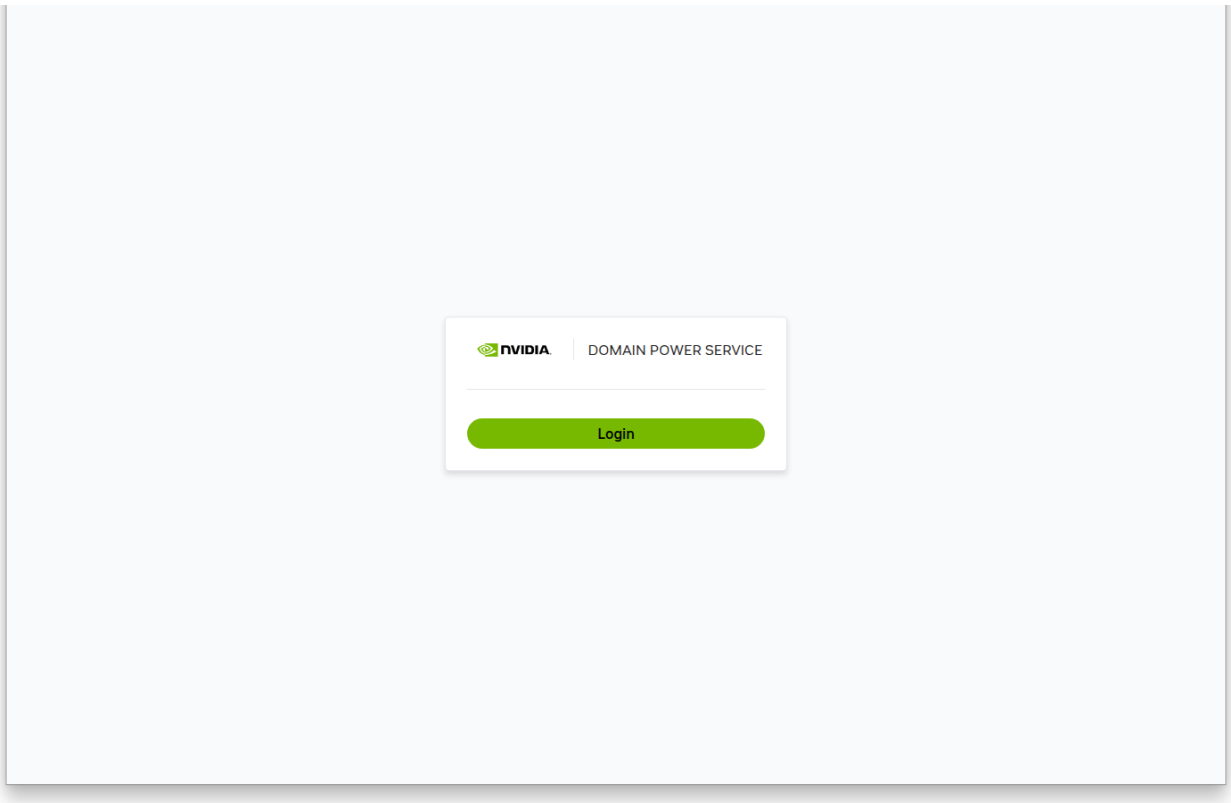
3.3.2.1.2.65 Prerequisites

- Deployed DPS system
- Web browser with JavaScript enabled
- Understanding of datacenter power distribution concepts

3.3.2.1.2.66 Accessing the UI

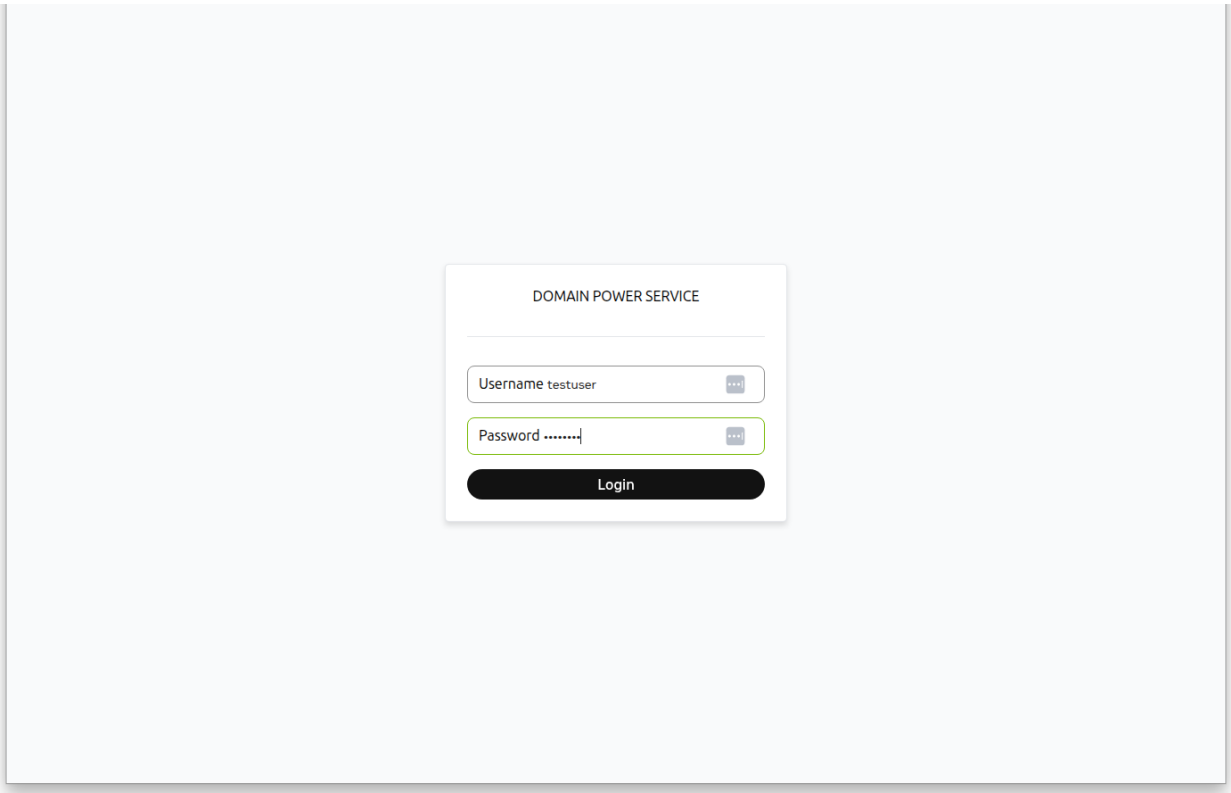
In the DPS SDK environment, the user interface is available at:

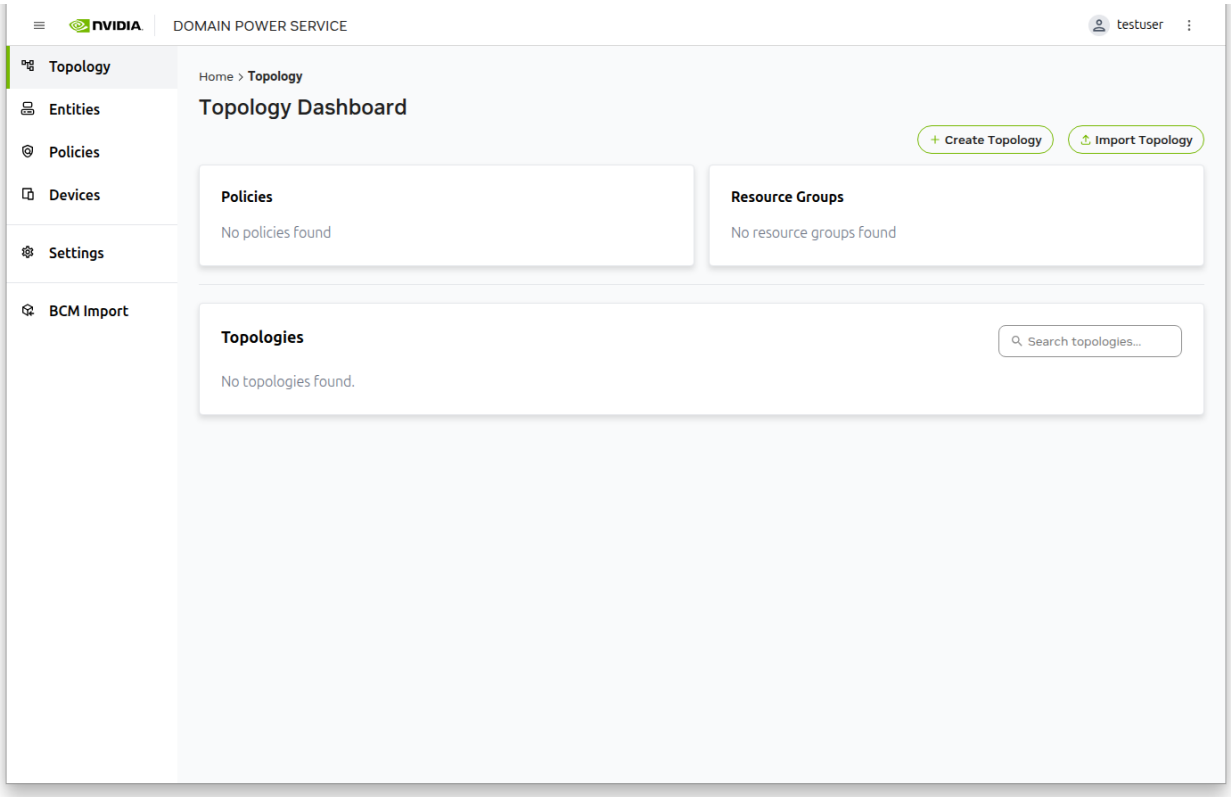
URL: http://ui.dps.sdk



Authentication:

- Username: `dps`
- Password: `dps`


DOMAIN POWER SERVICE
Username testuser
Password
Login


NVIDIA
DOMAIN POWER SERVICE
testuser
Topology
Entities
Policies
Devices
Settings
BCM Import
Home > Topology
Topology Dashboard
Create Topology
Import Topology
Policies
No policies found
Resource Groups
No resource groups found
Topologies
Search topologies...
No topologies found.

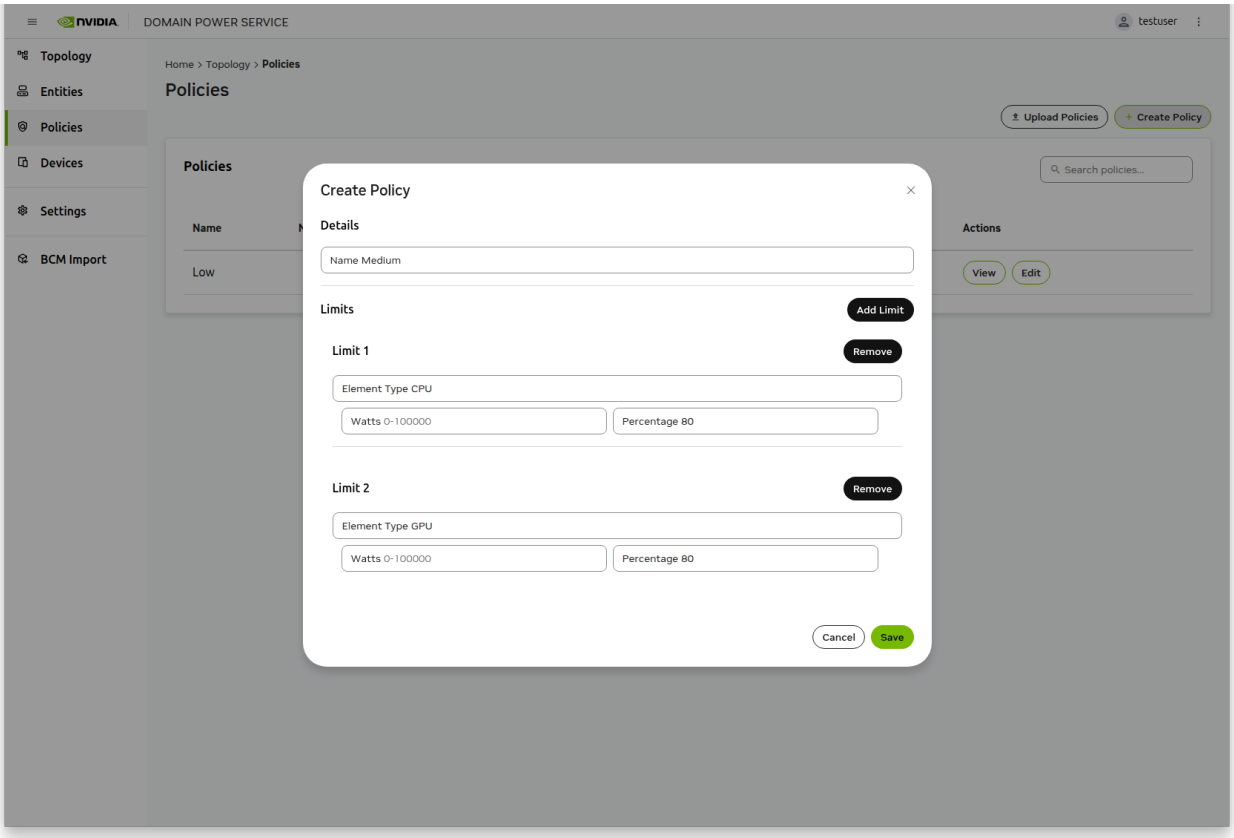
3.3.2.1.2.67 Topology Creation

Creating a topology involves several sequential steps: defining policies, optionally importing custom devices, creating entities, building the topology structure with their relationships, and finally activating it.

3.3.2.1.2.68 Step 1: Create Policies

Before creating a topology, you must first define the policies that will govern power limits.

1. Select **Policies** from the left navigation panel
2. Click the **Create Policy** button
3. Specify a policy name
4. Click **Add Limit** to define power constraints
   - Multiple limits can be added, one per element type
   - Limits can be specified in either Watts or Percentage
5. Click **Save** to store the policy
6. Repeat these steps for each additional policy you need

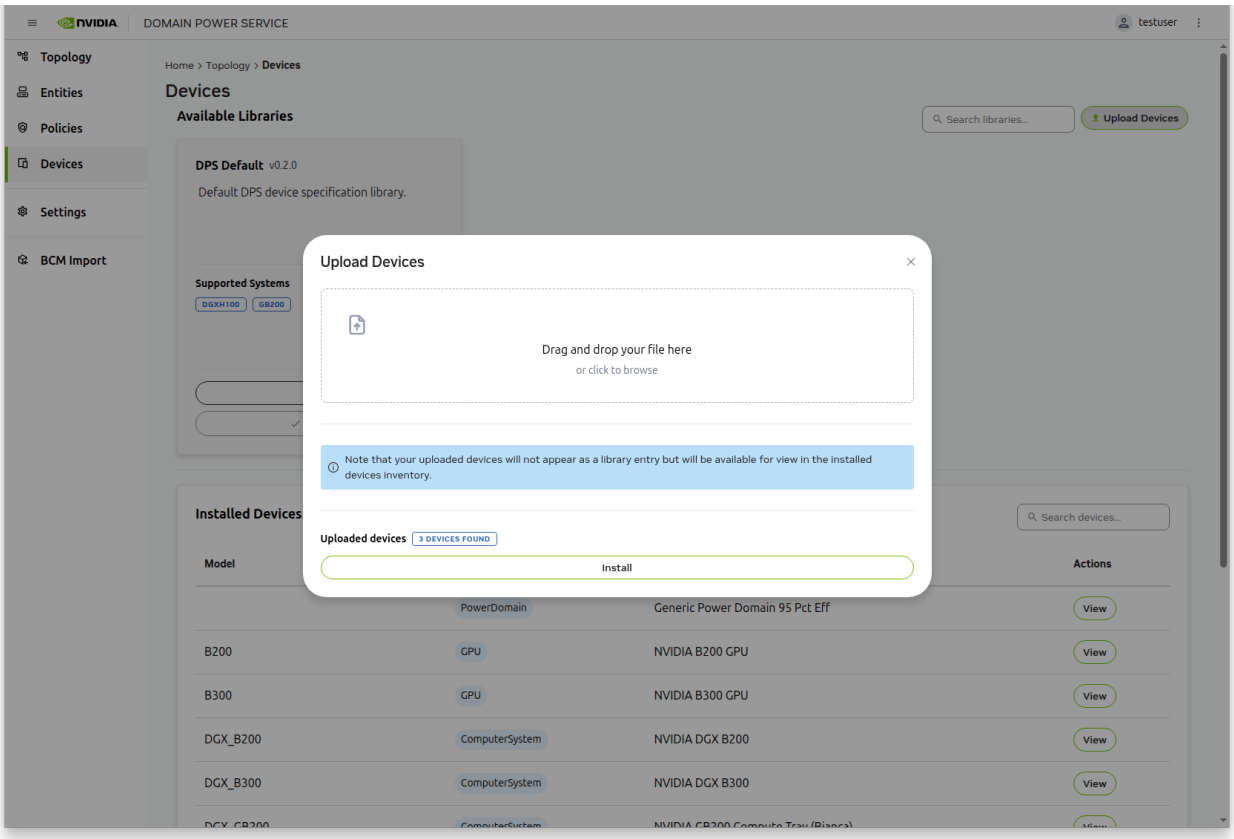


3.3.2.1.2.69 Step 2: Import Custom Devices (Optional)

If any of your entities require custom device specifications, you must import those devices before creating entities. The UI does not support defining custom devices directly through a form.

1. Select **Devices** from the left navigation panel
2. Click **Upload Devices**
3. Choose your device specification YAML file
4. The devices will be imported and available for use in entity creation

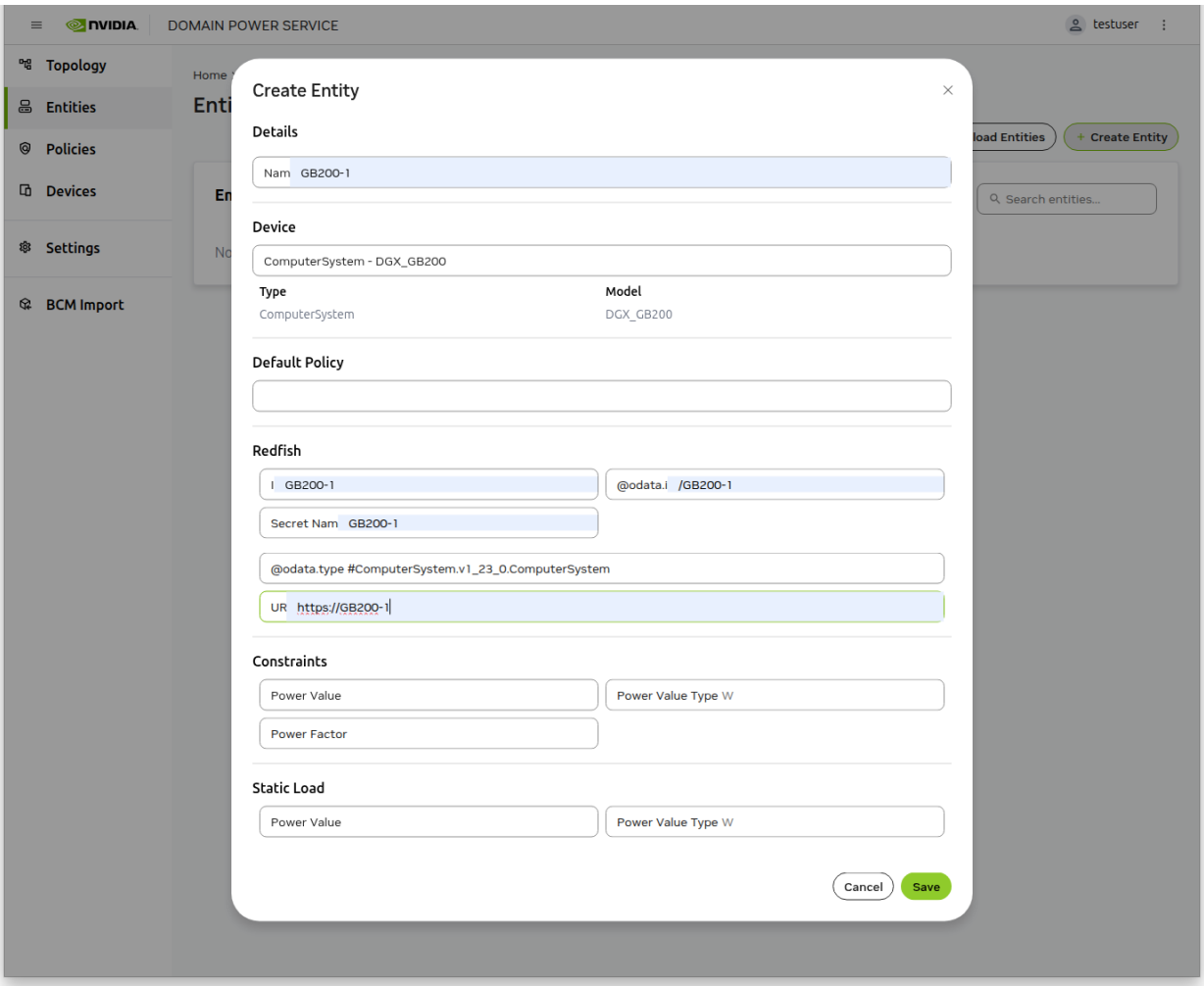
For examples of device specification YAML files and instructions on how to create them, refer to the *Add Custom Device Specifications* section in the PDN Diagrams guide.

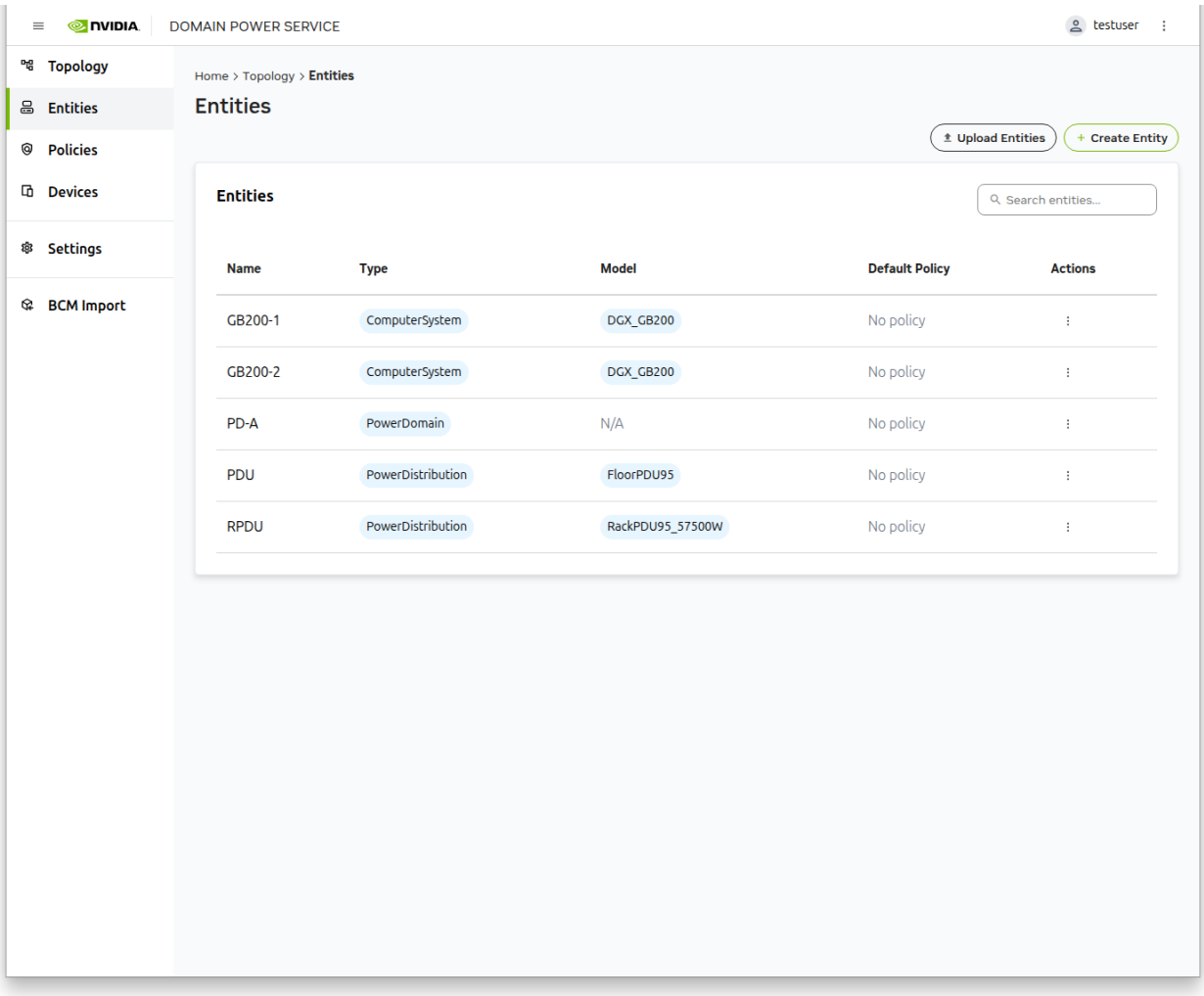


3.3.2.1.2.70 Step 3: Create Entities

With policies defined and any required custom devices imported, you can now create the entities that represent your power distribution components.

1. Select **Entities** from the left navigation panel
2. Click the **Create Entity** button
3. Fill in the entity details
4. Click **Save**
5. Repeat these steps for each entity in your topology


NVIDIA
DOMAIN POWER SERVICE
testuser
Topology
Entities
Policies
Devices
Settings
BCM Import
Create Entity
Details
Nam GB200-1
Device
ComputerSystem - DGX_GB200
Type
ComputerSystem
Model
DGX_GB200
Default Policy
Redfish
@odata.type #ComputerSystem.v1_23_0.ComputerSystem
Constraints
Power Value
Power Value Type W
Power Factor
Static Load
Power Value
Power Value Type W
Cancel
Save



3.3.2.1.2.71 Step 4: Build and Configure the Topology

With policies and entities defined, you can now construct your topology and define the relationships between entities. This is done in a single continuous workflow within the topology builder.

3.3.2.1.2.72 Add Entities to the Topology

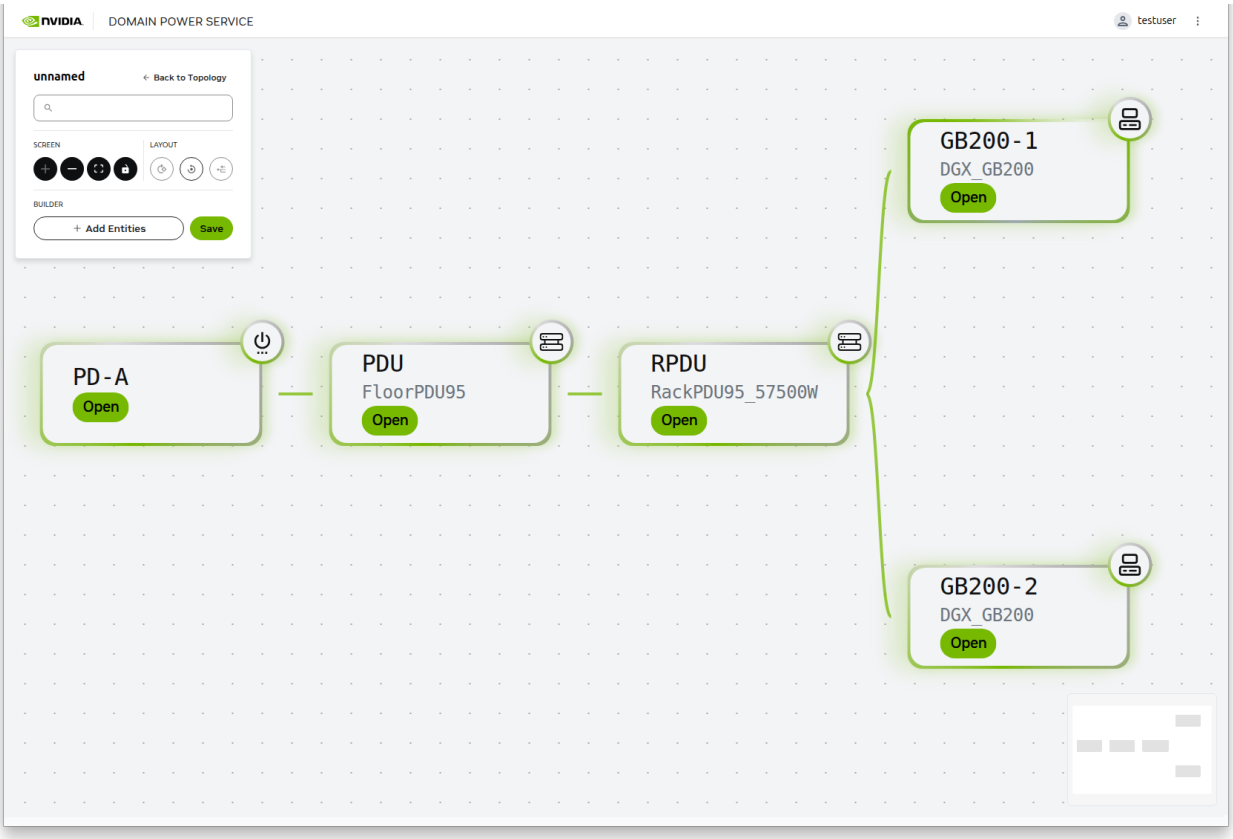
1. Navigate to **Topology** in the left navigation panel
2. Click **Create Topology**
3. In the dialog window, click **Add Entities**
4. Select all entities from the previous step
5. Click **Add Selected Entities**
   - ▶ The topology builder will populate with your selected entities

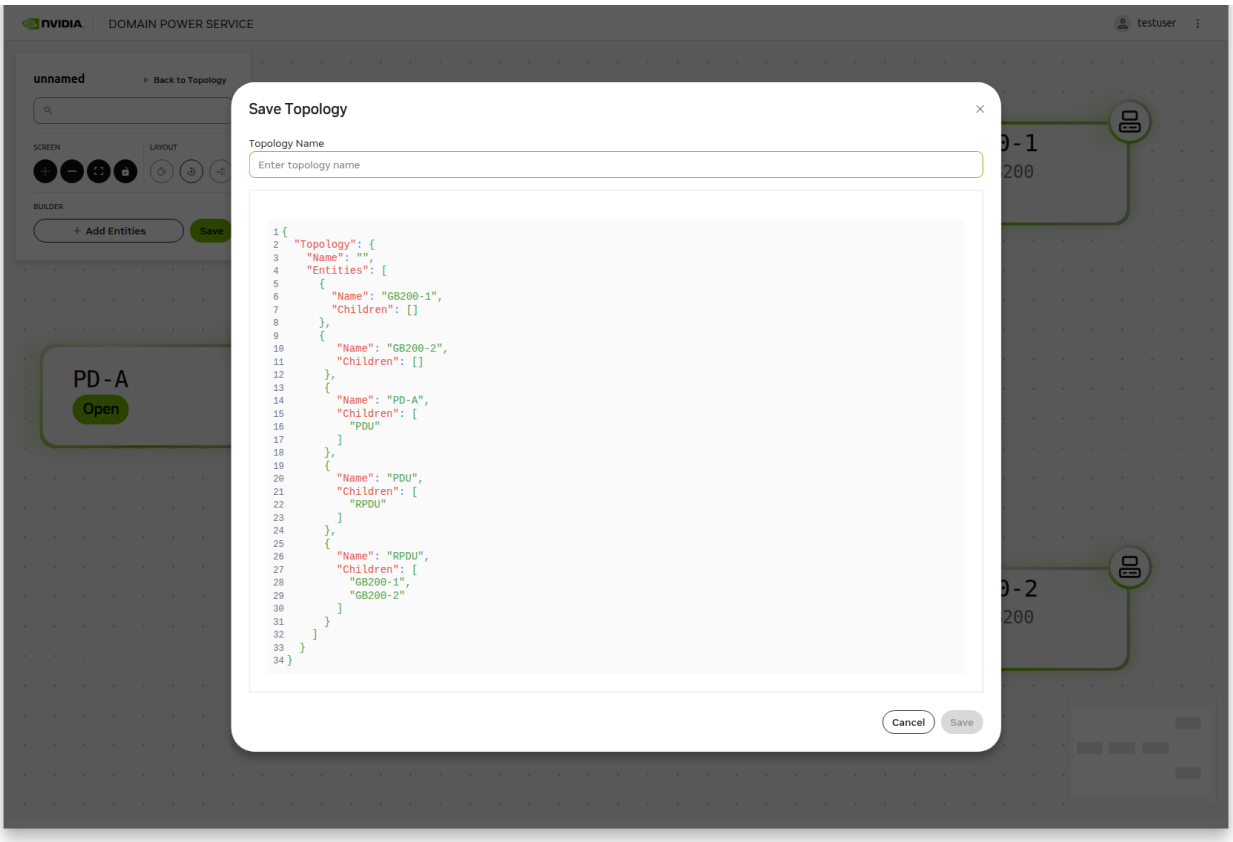
3.3.2.1.2.73 Define Entity Relationships

Configure the hierarchical relationships between entities using the interactive builder.

1. The builder automatically enters interactive mode after adding entities
2. Drag entities close to each other to create connections
   - **Parent-Child Hierarchy**: The leftmost item in a connection is the parent; the rightmost item is the child
   - Position determines the power distribution flow through your topology
3. Once all relationships are established, click **Save**
4. Specify a topology name in the save dialog
5. Review the generated topology JSON displayed at the bottom of the dialog
6. Click **Save** to create your topology

   Tip: To prevent undesired connection lines from being created accidentally, space entities wider apart before dragging them to create intentional connections. This gives you better control over which entities connect to each other.





3.3.2.1.2.74 Step 5: Activate the Topology

After creating your topology, activate it to make it operational.

1. Navigate to **Topology** in the left navigation panel to view the Topologies list
2. Locate your newly created topology
3. Select **Activate** from the **Actions** column

   Note: Only one topology can be active at a time. Activating a new topology will automatically deactivate any previously active topology.

3.3.2.1.2.75 Slurm Integration

Note

The authoritative procedure for integrating DPS with Slurm is *Integrating with Slurm*. Start there.

This page predates that one and covers the same ground from the SDK administrator section. Its prerequisites checklist, verification steps, and troubleshooting are being folded into the task guide.

Integrating the Dynamic Power Software (DPS) with Slurm allows for automated, workload-specific power management through Slurm's job scheduling. This integration is primarily achieved by configuring **prolog and epilog scripts** within your Slurm environment. These scripts create and manage DPS resource groups that correspond to High-Performance Computing (HPC) jobs.

3.3.2.1.2.76 Prerequisites for Slurm Integration

Before integrating DPS with Slurm, ensure the following are in place:

- A **Slurm cluster installed** and configured. Your user will need administrative access to the Slurm configuration and root access on the head node.
- **DPS server is running and accessible**.
- **`dpsctl` command-line client is installed and authenticated** on the head node through *Configure Server Authentication*.
- **Device specifications are defined** in DPS.
- An **active topology with entities configured** exists in DPS.
- Your user account has **administrative access to Slurm and root access on the head node**.

3.3.2.1.2.77 How DPS Integrates with Slurm

DPS integration with Slurm is managed through the `PrologSlurmctld` and `EpilogSlurmctld` parameters in the `slurm.conf` file. These parameters point to scripts that execute automatically before a job starts and after it completes. Each HPC job is represented in DPS as a resource group.

In a nutshell: The prolog and epilog scripts simply execute `dpsctl` commands to manage the compute nodes that Slurm has allocated for each job. When a job starts, the prolog script uses `dpsctl` to create a resource group and add the allocated nodes to it. When the job ends, the epilog script uses `dpsctl` to clean up by removing the resource group.

3.3.2.1.2.78 1. Prolog Script (Job Start)

The prolog script runs **before a job starts** on a compute node. Its main responsibilities include:

- **Creating a resource group** using the job ID as an external identifier.
- **Adding allocated compute nodes** to the newly created resource group.
- **Activating power policies** to apply specific power configurations to the hardware resources.
- **Configuring workload-specific settings**, such as power policies, Power Reservation Steering (PRS) status, Dynamic Power Management (DPM) status, and Workload Power Profile Settings (WPPS).
  - If DPS detects insufficient power to start the job at the requested settings, it may adjust or return a failure, causing the job to be re-queued if the prolog fails.

3.3.2.1.2.79 2. Epilog Script (Job End)

The epilog script runs **after a job completes** on a compute node. Its primary function is to **delete the resource group** created by the prolog script. This action ensures that the devices return to their original power configuration, as defined in the topology, after the job has finished.

3.3.2.1.2.80 Configuration Steps

To integrate DPS with Slurm, you will configure the prolog and epilog scripts in your Slurm configuration.

1. **Install DPS Integration Scripts**: First, ensure the DPS integration scripts are available on your system. These scripts should be installed alongside the DPS software package.

2. **Configure Slurm Prolog/Epilog Scripts**: Edit your `slurm.conf` file to include the prolog and epilog script paths. Add or modify the following lines in your `slurm.conf`:

```
PrologSlurmctld=/usr/share/dps/prolog.sh
EpilogSlurmctld=/usr/share/dps/epilog.sh
```

Note: Adjust the preceding paths to match the actual location of the DPS prolog and epilog scripts on your system.

3. **Restart Slurm Services**: After modifying the `slurm.conf` file, restart the Slurm controller daemon to apply the changes:

```
sudo systemctl restart slurmctld
```

4. **Verify Script Settings**: Verify that the script settings were correctly configured in your `slurm.conf` file:

```
grep -E "(PrologSlurmctld|EpilogSlurmctld)" /etc/slurm/slurm.conf
```

Expected output will show lines like:

```
PrologSlurmctld=/usr/share/dps/prolog.sh
EpilogSlurmctld=/usr/share/dps/epilog.sh
```

3.3.2.1.2.81 Advanced Slurm Integration with Job Comments

Advanced Slurm integration supports parsing DPS settings directly from job comments. The prolog script can automatically parse these settings to apply specific configurations:

- `dps_policy:<string>`: Sets a specific power policy.
- `dps_prs:<bool>`: Disables Power Reservation Steering.
- `dps_dpm:<bool>`: Enables Dynamic Power Management.
- `dps_wpps:<comma-separated ints>`: Sets workload power profile IDs for supported GPUs. For WPPS, the Out-Of-Band (OOB) value used by Redfish APIs on the BMC is offset by 3 from the DCGMI/NVSMI value (OOB Value = DCGMI/NVSMI Value + 3).

3.3.2.1.2.82 Troubleshooting

- **Prolog Script Failures**: If the DPS prolog script fails (returns a non-zero exit code), the job will be re-queued. This typically indicates that DPS was unable to configure a corresponding resource group with the requested power settings, and thus the job should not run.
- **Logging**: The DPS prolog/epilog scripts log to STDOUT by default.
- **Authentication and Authorization**: Configure the Slurm service account using *Configure Server Authentication* and *Service Accounts*. Verify API access under the account that runs the prolog and epilog scripts.
- **Slurm Documentation**: Refer to the official Slurm documentation for further information on prolog and epilog scripts.

3.3.2.1.2.83 Nautobot Integration

Note

The authoritative procedure for importing devices from an external inventory is *Importing External Inventory*, which covers both Nautobot and Base Command Manager and includes the guided import workflow. Start there.

This page covers the Helm configuration and the Nautobot-side prerequisites that the task guide assumes are already in place.

Nautobot serves as the authoritative device inventory source for DPS, allowing operators to import physical devices directly from their existing DCIM into DPS for power management. If enabled, DPS can query Nautobot's REST API on demand to discover devices by location, rack, or device type and map them into DPS entities.

3.3.2.1.2.84 Prerequisites

- A running Nautobot instance reachable from the DPS pod (HTTP or HTTPS).
- `dps.nautobot.enabled: true` in your Helm values.
- A Nautobot API token with read access to DCIM, stored in a Kubernetes Secret and referenced via `dps.nautobot.tokenSecret`.

3.3.2.1.2.85 How DPS Integrates with Nautobot

DPS can import devices from Nautobot on demand through either the DPS UI or the `dpsctl` CLI. When triggered, the DPS server receives an import request over gRPC, then uses its built-in Nautobot client to query the Nautobot REST API with the specified filters. Nautobot responds with the matching devices from its DCIM database, and DPS automatically maps these devices into entities for power management.

3.3.2.1.2.86 Configuration Steps

Refer to the *Deployment Guide* for how to configure Nautobot

3.3.2.1.2.87 Troubleshooting

1. Verify Helm values are set correctly

```
helm get values <release> -n <namespace> | grep -A 10 nautobot
```

Confirm `enabled: true`, `url` is non-empty and includes the scheme (`https://`), and either `tokenSecret.existingSecret` or `tokenSecret.secretName` is set

2. Confirm the token Secret exists in the cluster

```
kubectl get secret <secret-name> -n <namespace>
kubectl get secret <secret-name> -n <namespace> -o jsonpath='{.data.token}' |
↪base64 -d
```

The decoded value should be a non-empty Nautobot API token.

3. Test connectivity to Nautobot from the DPS pod network namespace

Deploy an ephemeral container sharing the same network namespace:

```
kubectl debug -it -n <namespace> <dps-pod> \
  --image=curlimages/curl \
  --target=dps-server -- \
  curl -sf -H "Authorization: Token <token>" \
   https://nautobot.company.com/
```

A non-200 response indicates a network configuration preventing DPS from reaching the Nautobot REST API

4. Check DPS server logs for startup connectivity errors

```
kubectl logs -n <namespace> <dps-pod> | grep -i nautobot
```

3.3.2.1.2.88 Power Compliance

The Dynamic Power Software (DPS) ensures power compliance through a validation system that operates at two critical stages: topology activation and resource group creation. This guide explains how DPS maintains power compliance by validating constraints and automatically adjusting policies when necessary.

3.3.2.1.2.89 Overview

DPS power compliance is achieved through:

1. **Topology Constraint Validation** - Ensuring the power topology can support the defined constraints before activation
2. **Dynamic Resource Group Validation** - Validating power requirements when new resource groups are created
3. **Automatic Policy Adjustment** - Falling back to lower power policies when constraints cannot be met
4. **Continuous Monitoring** - Ongoing validation of power consumption against defined limits

3.3.2.1.2.90 Topology Activation Compliance

3.3.2.1.2.91 Initial Validation Process

When a topology is activated, DPS performs validation to ensure all power constraints can be satisfied:

3.3.2.1.2.92 1. Power Budget Verification

- **Total Power Calculation**: DPS calculates the total power requirements across all entities in the topology
- **Constraint Checking**: Validates that power domains have sufficient capacity to support their child entities
- **Hierarchy Validation**: Ensures power flows correctly through the entire power distribution hierarchy

3.3.2.1.2.93 2. Power Factor Validation

- **Loss Calculations**: Accounts for power losses through power distribution components using defined power factors
- **Efficiency Modeling**: Validates that power supply efficiency ratings are realistic for the expected loads

3.3.2.1.2.94 3. Activation Failure Scenarios

The topology activation will fail if:

- Total power demand exceeds available power budget in any power domain
- Power distribution components cannot handle the calculated power flows
- Constraint violations are detected at any level of the power hierarchy

3.3.2.1.2.95 Resource Group Compliance

3.3.2.1.2.96 Dynamic Validation During Creation

When resource groups are created, DPS performs real-time compliance validation:

3.3.2.1.2.97 1. Policy Application Validation

- **Resource Allocation**: DPS calculates power requirements for the specific nodes allocated to the resource group
- **Policy Integration**: Applies the requested power policy to the allocated resources
- **Constraint Verification**: Validates that the modified topology still satisfies all power constraints

3.3.2.1.2.98 2. Power Tree Constraint Checking

DPS validates the entire power tree with the new policy applied:

```
PowerDomain (Root)
    PDU-A
        Node-01 [Applied Policy: High-Performance]
        Node-02 [Applied Policy: High-Performance]
    PDU-B
        Node-03 [Default Policy]
        Node-04 [Default Policy]
```

The validation ensures:

- The root power domain can supply the total demand
- Each PDU can handle its assigned load
- No thermal or electrical limits are exceeded

3.3.2.1.2.99 3. Resource Group Creation Process

1. **Initial Request**: Resource group creation request with desired power policy
2. **Power Calculation**: DPS calculates total power requirements with the requested policy

3. **Constraint Validation**: Validates against all topology constraints
4. **Success Path**: If validation passes, resource group is created with the requested policy
5. **Fallback Path**: If validation fails, DPS attempts policy adjustment

3.3.2.1.2.100 Automatic Policy Adjustment

3.3.2.1.2.101 Policy Fallback Mechanism

When the requested power policy cannot be satisfied, DPS implements an intelligent fallback system:

3.3.2.1.2.102 1. Policy Hierarchy

DPS maintains a hierarchy of power policies from highest to lowest power consumption, for example:

```
High-Performance   Balanced   Power-Saver   Emergency
```

3.3.2.1.2.103 2. Automatic Adjustment Process

1. **Validation Failure**: Initial policy validation fails due to power constraints
2. **Policy Downgrade**: DPS automatically tries the next lower power policy
3. **Re-validation**: Performs constraint validation with the lower policy
4. **Iteration**: Continues until a compliant policy is found or all options are exhausted

3.3.2.1.2.104 3. Fallback Example

```
# Resource group creation with automatic policy adjustment
Initial Request: Policy "High-Performance" for nodes [node-01, node-02]
Validation Result: FAILED - Insufficient power budget (deficit: 2500W)

Attempting Fallback: Policy "Balanced"
Validation Result: FAILED - Insufficient power budget (deficit: 800W)

Attempting Fallback: Policy "Power-Saver"
Validation Result: SUCCESS - Resource group created with "Power-Saver" policy

Resource Group ID: rg-12345
Applied Policy: Power-Saver
Allocated Nodes: node-01, node-02
Power Consumption: 12500W (within budget)
```

3.3.2.1.2.105 Best Practices for Compliance

3.3.2.1.2.106 1. Topology Design

- **Conservative Budgeting**: Allocate 10-15% headroom in power domains for unexpected demands
- **Realistic Power Factors**: Use measured power factors rather than theoretical values
- **Hierarchical Constraints**: Implement constraints at multiple levels for better granularity

3.3.2.1.2.107 2. Policy Management

- **Graduated Policies**: Create policies with gradual power differences to enable smooth fallbacks
- **Emergency Policies**: Always maintain ultra-low power emergency policies

3.3.2.1.2.108 3. Monitoring and Maintenance

- **Regular Validation**: Periodically re-validate topology constraints as hardware changes
- **Trend Analysis**: Monitor power consumption trends to predict future capacity needs

3.3.2.1.2.109 Troubleshooting Compliance Issues

3.3.2.1.2.110 Common Compliance Problems

3.3.2.1.2.111 1. Topology Activation Failures

Problem: Topology fails to activate due to power budget violations **Solution**:

- Review power domain capacities and increase if necessary
- Adjust default topology entity policies
- Validate entity power specifications

3.3.2.1.2.112 2. Resource Group Creation Failures

Problem: All policy fallbacks fail during resource group creation **Solution**:

- Review current power allocation across existing resource groups
- Implement more aggressive power-saving policies

3.3.2.2 Partner Integration

This section provides integration guides for DPS partners, covering grid solutions, optimization systems, and workload management integrations.

3.3.2.2.1 Partner Categories

3.3.2.2.1.1 Grid Integration Partners

Partners who provide grid intelligence solutions that interface between electrical utility grids and datacenter power management systems.

- ***Grid Integrator SDK*** - Grid intelligence solutions for datacenter power integration using the DPS NvGrid gRPC API

3.3.2.2.1.2 Power Optimization Partners

Partners who develop optimization algorithms and systems for datacenter power efficiency and performance optimization.

- ***Optimization Developer SDK*** - Tools for developing power and performance optimization solutions within NVIDIA's datacenter infrastructure

3.3.2.2.1.3 Workload Scheduling Partners

Partners who integrate workload schedulers so that placement decisions and DPS power management stay consistent.

- ***Workload Scheduling Integrator SDK*** - Integrating a workload scheduler with DPS

3.3.2.2.2 Integration Approach

3.3.2.2.2.1 Standards-Based APIs

All partner integrations use standardized gRPC APIs with protobuf message definitions for communication between partner systems and DPS.

3.3.2.2.2.2 Simulator-First Development

Partner integrations are developed and tested using DPS simulators, providing consistent testing environments independent of specific hardware configurations or utility connections.

3.3.2.2.2.3 Event-Driven Architecture

Partner systems receive real-time notifications through webhook systems for grid events, power changes, and system state transitions.

3.3.2.2.3 Getting Started

3.3.2.2.3.1 Prerequisites

Before beginning partner integration development:

1. **DPS Environment**: Access to DPS development environment with simulators
2. **API Access**: Credentials configured for your server's setup through *Configure Server Authentication*
3. **Development Tools**: gRPC development tools for chosen programming language
4. **Documentation Access**: Access to API specifications and integration guides

3.3.2.2.3.2 Development Process

1. **Environment Setup**: Deploy DPS development environment with simulators
2. **API Client Development**: Implement gRPC client using DPS API definitions
3. **Integration Testing**: Test integration using simulator environment
4. **Production Validation**: Validate integration in production-like environment

3.3.2.2.3.3 Support and Resources

- **API Documentation**: gRPC API specifications and message definitions
- **Simulator Configuration**: Testing environment configurations
- **Integration Examples**: Code examples for common integration patterns

3.3.2.2.4 Partner Integrations

Partner integrations with DPS enable:

- **Grid Intelligence**: Grid integration capabilities for demand response and load management
- **Power Efficiency**: Power optimization algorithms integrated with DPS power management

3.3.2.2.4.1 Grid Integrator SDK

3.3.2.2.4.2 Overview

The Grid Integrator SDK provides tools for integrating grid intelligence solutions with DPS datacenter power management. NvGrid operates at the power feed level, providing granular control over individual electrical circuits or datacenter-wide power consumption.

Grid integrators use the NvGrid gRPC API to schedule power constraints, receive power event notifications, and coordinate datacenter operations with utility grid requirements.

Integration scenarios include utility curtailment and demand response, emergency load shedding during grid instability, dynamic pricing optimization, renewable energy coordination, and grid stability support.

3.3.2.2.4.3 Target Persona

Grid Integration Engineer: Developer creating solutions that connect electrical utility grids with datacenter power management systems.

Capabilities:

- Schedule power constraint changes, such as planned power reductions, based on grid signals
- Monitor power utilization and target achievement through webhooks
- Query current power status and feed metadata
- Implement demand response programs
- Manage load curtailment events

3.3.2.2.4.4 SDK Components

- ***Prerequisites*** - Infrastructure, development tools, and testing environment setup
- ***Core Concepts*** - Architectural principles, power management strategy, and feed management
- ***API Specification*** - Complete NvGrid gRPC API reference with methods, messages, and data structures
- ***Managing Power Constraints*** - Schedule load targets and manage power constraints across datacenter feeds
- ***Monitoring Power Events*** - Register webhooks and receive power event notifications

3.3.2.2.4.5 Getting Started

Integrate grid solutions with DPS using the NvGrid gRPC API. Begin by reviewing prerequisites and core concepts, then explore the API specification before implementing power constraint management and event monitoring.

Security Note: Webhook URLs must be configured in the DPS server whitelist for event notifications. See Prerequisites for webhook security configuration details.

3.3.2.2.4.6 Integration Architecture

3.3.2.2.4.7 Prerequisites

Before implementing grid integration with DPS NvGrid, ensure the following are in place.

3.3.2.2.4.8 DPS Infrastructure

Required infrastructure:

- DPS server running and accessible with NvGrid service enabled
- Active topology with datacenter entities configured
- Power policies and resource groups defined
- API credentials configured through *Configure Server Authentication*

Verification:

In DPS 0.9, the topology must be active before running `dpsctl check connection`. Confirm the active topology before testing BMC connectivity:

```
# List topologies
dpsctl topology list

# Verify NvGrid service
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify nvgrid get-
↪metadata

# Test API connectivity
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify check connection

# Test with grpcurl (optional)
grpcurl -insecure api.dps:443 nvidia.dcpower.v1.NvGrid/GetStatus
```

3.3.2.2.4.9 Webhook Security Configuration

Configure webhook URL whitelist to control which endpoints can receive power event notifications.

Server Configuration:

Add authorized webhook URLs to DPS Helm values:

```
# values.yaml
dps:
  nvgrid:
    webhook:
      whitelist:
        - "https://grid-solution.example.com/nvgrid/webhook"
        - "https://monitoring.example.com/api/"
        - "http://localhost:8080/"
```

(continues on next page)

(continued from previous page)

```
        - "http://host.docker.internal:8181"
      concurrency: 32
      callTimeout: 1m
```

Apply configuration:

```
helm upgrade dps ./helm/dps \
  --namespace dps \
  --values values.yaml \
  --reuse-values
```

Whitelist Validation:

- RegisterWebhook API validates URLs against whitelist during registration
- Non-whitelisted URLs are rejected with error: "webhook URL not whitelisted"
- Failed validation attempts are logged for security auditing

Whitelist Matching Rules:

The whitelist supports flexible URL matching patterns:

```
dps:
  nvgrid:
    webhook:
      whitelist:
        # Exact URL match
        - "https://grid.example.com/webhook"
        # Path prefix match (allows any path under /api/)
        - "https://grid.example.com/api/"
        # Development endpoints
        - "http://localhost:8080/"
        - "http://127.0.0.1:8080/"
        - "http://host.docker.internal:8181"
```

Matching Components:

- **Scheme**: Must match (http vs https)
- **Hostname**: Must match exactly
- **Port**: Must match (443 vs 8080)
- **Path**: Prefix matching supported (trailing `/` matches all sub-paths)
- **Query Parameters**: Whitelisted params must be present in request URL
- **Fragment**: Must match if specified

Example:

Whitelist: `https://grid.example.com/api/`

- ✓ Matches: `https://grid.example.com/api/webhooks/nvgrid`
- ✓ Matches: `https://grid.example.com/api/v1/power`
- ✗ Rejected: `https://grid.example.com/other/path`
- ✗ Rejected: `http://grid.example.com/api/` (scheme mismatch)

- ✗ Rejected: `https://grid.example.com:8080/api/` (port mismatch)

3.3.2.2.4.10 Development Environment

Required development tools:

- gRPC development tools for chosen programming language
- Protocol buffer compiler for message generation
- DPS API definitions accessible from `api/v1/nvgrid.proto`

Client code generation:

```
# Python
python3 -m grpc_tools.protoc \
  -I ./api/v1 \
  --python_out=./generated \
  --grpc_python_out=./generated \
  ./api/v1/nvgrid.proto

# Go
protoc --go_out=. --go-grpc_out=. api/v1/nvgrid.proto
```

3.3.2.2.4.11 Testing Infrastructure

Testing environment requirements:

- DPS simulator environment running
- Test topology with representative hardware configuration
- Webhook endpoint for receiving power event notifications

Simulator Environment:

The DPS simulator environment includes BMC and BCM simulators that emulate hardware responses for testing.

3.3.2.2.4.12 Next Steps

With prerequisites in place, proceed to:

1. ***Core Concepts*** - Understand architectural principles
2. ***API Specification*** - Review API methods and messages
3. ***Managing Power Constraints*** - Schedule load targets
4. ***Monitoring Power Events*** - Implement webhook receivers

3.3.2.2.4.13 Managing Power Constraints

3.3.2.2.4.14 Overview

This use case demonstrates how to schedule and manage datacenter power constraints using the NvGrid API. Grid integrators can schedule future power changes, query scheduled targets, and retrieve feed metadata to respond to grid signals.

3.3.2.2.4.15 Goal

Schedule datacenter power constraint changes in response to grid signals such as curtailment events, demand response programs, or dynamic pricing signals.

3.3.2.2.4.16 Persona: Grid Integration Engineer

Role: Developer integrating commercial grid solutions with DPS for datacenter power management.

Objective: Implement grid signal response by scheduling power constraints and monitoring load target achievement.

3.3.2.2.4.17 Use Case Workflow

Scenario: Grid solution receives a curtailment signal and needs to reduce datacenter power consumption

Workflow Steps:

1. **Discover Feed Configuration** - Query feed metadata to understand power thresholds and current defaults
2. **Calculate Target Load** - Determine appropriate power constraint based on grid signal
3. **Schedule Load Target** - Set power constraint with start/end times
4. **Query Schedule** - Retrieve current and future scheduled targets
5. **Monitor Status** - Track actual power consumption vs target

---

3.3.2.2.4.18 Theory of Operation

3.3.2.2.4.19 Load Target Scheduling

Load targets define time-based power constraints for datacenter feeds. Each target specifies:

- **Interval**: Start and end times for the constraint
- **Load Constraint**: Target power limit (watts, kilowatts, or megawatts), or default (zero) to reset
- **Strategy**: Algorithm for achieving the target (currently `best_effort`)
- **Feed Tags**: Specific feeds to constrain (or all if omitted)
- **Correlation ID**: Unique identifier for tracking

Setting Constraints:

- Use `--value` and `--unit` to set a specific power limit
- Use `--default` to reset to the default (zero) constraint

- Alternatively, `--value=0` with any unit also sets the default constraint
- `--default` and `--value` are mutually exclusive

3.3.2.2.4.20 Schedule Conflict Resolution

Resolution Rule: For overlapping intervals, the most recently scheduled target takes precedence.

- Default constraint: 10MW
- Set 6MW from 4pm-7pm Active 4pm-7pm
- Set 8MW from 5pm-6pm **Overrides** to 8MW from 5pm-6pm
- Set 5MW from 6pm-7pm **Reverts** to 6MW from 6pm-7pm
- After 7pm Reverts to 10MW default

3.3.2.2.4.21 Power Reduction Strategy

The `best_effort` strategy:

- Reduces power from DPM-enabled workloads
- Does not terminate running jobs
- Target may not be achieved if constraint requires job termination
- Restores power when constraints are relaxed

3.3.2.2.4.22 Python Client Setup

This section shows one-time setup required for all of the following Python examples. Complete these steps once before using any Python code examples.

3.3.2.2.4.23 1. Install the DPS Python API

```
# Navigate to the Python API directory
cd /path/to/dcpower/api/python3

# Install the package
pip3 install -e .

# Verify installation
python3 -c "from dpsapi.v1 import nvgrid_pb2, nvgrid_pb2_grpc; print('Success
↪')"
```

3.3.2.2.4.24 2. Extract the CA Certificate

```
# Extract CA certificate from your DPS deployment
kubectl get secret -n <namespace> dps-api-tls -o jsonpath='{.data.ca\.crt}' |
↪base64 -d > /tmp/dps-ca.crt

# Security Note: In production, store certificates in a secure location
```

(continues on next page)

(continued from previous page)

```
# with appropriate file permissions (e.g., 0600) and consider using
# a dedicated certificate directory like /etc/ssl/certs/dps/

# Verify the certificate
ls -lh /tmp/dps-ca.crt

# Example for dps-test namespace:
kubectl get secret -n dps-test dps-api-tls -o jsonpath='{.data.ca\.crt}' |
↪base64 -d > /tmp/dps-ca.crt
```

3.3.2.2.4.25 3. Setup Authentication Helper

Save this reusable helper for all examples:

```
# nvgrid_client.py - Reusable authentication helper
import grpc
from dpsapi.v1 import auth_pb2, auth_pb2_grpc

def create_tls_channel(target, authority, ca_cert_path):
    """Create TLS channel with CA certificate"""
    with open(ca_cert_path, 'rb') as f:
        ca_cert = f.read()

    credentials = grpc.ssl_channel_credentials(root_certificates=ca_cert)
    options = [
        ('grpc.default_authority', authority),
        ('grpc.ssl_target_name_override', authority.split(':')[0]),
    ]

    return grpc.secure_channel(target, credentials, options=options)

def get_auth_token(username, password, host='api.dps', port=443, resolve_
↪to=None, ca_cert_path='/tmp/dps-ca.crt'):
    """Login and get access token"""
    target = f'{resolve_to or host}:{port}'
    authority = f'{host}:{port}'

    channel = create_tls_channel(target, authority, ca_cert_path)
    stub = auth_pb2_grpc.AuthServiceStub(channel)

    try:
        req = auth_pb2.TokenRequest(
            password_credential=auth_pb2.PasswordCredential(
                username=username,
                password=password
            )
        )
        resp = stub.Token(req)
        return resp.access_token
    finally:
        channel.close()
```

(continues on next page)

(continued from previous page)

```
def create_nvgrid_client(token, host='api.dps', port=443, resolve_to=None, ca_
↪cert_path='/tmp/dps-ca.crt'):
    """Create authenticated NvGrid client stub"""
    from dpsapi.v1 import nvgrid_pb2_grpc

    target = f'{resolve_to or host}:{port}'
    authority = f'{host}:{port}'

    channel = create_tls_channel(target, authority, ca_cert_path)
    stub = nvgrid_pb2_grpc.NvGridStub(channel)

    # Return both channel and stub so caller can close channel
    return channel, stub

def create_auth_metadata(token):
    """Create gRPC metadata with auth token"""
    return [('authorization', f'Bearer {token}')]

def format_load_value(load_value):
    """Format LoadValue for display"""
    from dpsapi.v1 import nvgrid_pb2

    if not load_value or not load_value.value:
        return "not set"

    unit_names = {
        nvgrid_pb2.LoadValue.UNIT_WATT: 'W',
        nvgrid_pb2.LoadValue.UNIT_KILOWATT: 'kW',
        nvgrid_pb2.LoadValue.UNIT_MEGAWATT: 'MW'
    }
    unit = unit_names.get(load_value.unit, 'W')
    return f"{load_value.value:.2f} {unit}"
```

Usage in all examples below:

```
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

# Login once
token = get_auth_token('admin', 'password', host='api.dps-test', resolve_to=
↪'127.0.0.1')

# Create client
channel, stub = create_nvgrid_client(token, host='api.dps-test', resolve_to=
↪'127.0.0.1')
metadata = create_auth_metadata(token)

# Use stub with metadata in API calls
# ... your code here ...

# Close when done
```

(continues on next page)

(continued from previous page)

```
channel.close()
```

3.3.2.2.4.26 Step-by-Step Integration Guide

3.3.2.2.4.27 Step 1: Discover Feed Configuration

Query feed metadata to understand available feeds and their power thresholds.

3.3.2.2.4.28 Python Example

```
#!/usr/bin/env python3
"""
Step 1: Discover feed configuration and power thresholds
"""
from google.protobuf import empty_pb2
from dpsapi.v1 import nvgrid_pb2
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

def discover_feeds(username, password):
    """Discover feed configuration from NvGrid API"""

    # Login
    token = get_auth_token(username, password, host='api.dps', port=443)

    # Create NvGrid client
    channel, stub = create_nvgrid_client(token, host='api.dps', port=443)
    metadata = create_auth_metadata(token)

    try:
        # Get feed metadata
        response = stub.GetMetadata(empty_pb2.Empty(), metadata=metadata)

        print(f"Found {len(response.metadata)} power feed(s):\n")

        feeds = {}
        for feed_tag, entry in response.metadata.items():
            feeds[feed_tag] = {
                'power_minimum': format_load_value(entry.power_minimum),
                'power_maximum': format_load_value(entry.power_maximum),
                'default_constraint': format_load_value(entry.default_
↪constraint)
            }

            print(f"Feed: {feed_tag}")
            print(f"  Minimum: {feeds[feed_tag]['power_minimum']}")
            print(f"  Maximum: {feeds[feed_tag]['power_maximum']}")
            print(f"  Default: {feeds[feed_tag]['default_constraint']}\n")
```

(continues on next page)

(continued from previous page)

```
        return feeds

    finally:
        channel.close()

def format_load_value(load_value):
    """Format LoadValue for display"""
    if not load_value or not load_value.value:
        return "not set"

    unit_names = {
        nvgrid_pb2.LoadValue.UNIT_WATT: 'W',
        nvgrid_pb2.LoadValue.UNIT_KILOWATT: 'kW',
        nvgrid_pb2.LoadValue.UNIT_MEGAWATT: 'MW'
    }
    unit = unit_names.get(load_value.unit, 'W')
    return f"{load_value.value:.2f} {unit}"

# Usage
if __name__ == '__main__':
    # Use environment variables for credentials in production
    # e.g.
    # username = os.getenv('DPS_USERNAME', 'admin')
    # password = os.getenv('DPS_PASSWORD')
    feeds = discover_feeds('admin', 'password')
```

3.3.2.2.4.29 Step 2: Calculate Target Load

Determine the appropriate power constraint based on the grid signal received.

3.3.2.2.4.30 Python Example

```
#!/usr/bin/env python3
"""
Step 2: Calculate target load based on grid signal
"""

def calculate_target_load(target_mw, feeds):
    """Calculate target load for all feeds

    Args:
        target_mw: Target load in megawatts (e.g., 5.0 for 5MW)
        feeds: Dict of feed metadata from Step 1

    Returns:
        Dict with target load per feed in watts
    """
    targets = {}

    for feed_tag in feeds.keys():
        target_watts = target_mw * 1_000_000
```

(continues on next page)

(continued from previous page)

```
        targets[feed_tag] = target_watts
        print(f"Feed {feed_tag}: {target_mw:.2f} MW")

    return targets

# Usage
target_mw = 5.0  # Set load target to 5 megawatts
targets = calculate_target_load(target_mw, feeds)
```

3.3.2.2.4.31 Step 3: Schedule Load Target

Set the power constraint with start and end times.

3.3.2.2.4.32 Python Example

```
#!/usr/bin/env python3
"""
Step 3: Schedule load target
"""
from datetime import datetime, timedelta
from google.protobuf.timestamp_pb2 import Timestamp
from dpsapi.v1 import nvgrid_pb2
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

def schedule_load_target(stub, metadata, targets, start_time, duration_
↪minutes, correlation_id=None):
    """Schedule load target on NvGrid

    Args:
        stub: NvGrid client stub
        metadata: Auth metadata
        targets: Dict of feed_tag -> target_watts
        start_time: datetime when constraint starts
        duration_minutes: How long constraint is active
        correlation_id: Optional tracking ID
    """
    end_time = start_time + timedelta(minutes=duration_minutes)
    request = nvgrid_pb2.LoadTargetRequest()

    for feed_tag, target_watts in targets.items():
        target = request.targets.add()
        target.interval.start_time.CopyFrom(Timestamp(seconds=int(start_time.
↪timestamp())))
        target.interval.end_time.CopyFrom(Timestamp(seconds=int(end_time.
↪timestamp())))
        target.load_constraint.value = target_watts / 1_000_000  # Convert to
↪MW
        target.load_constraint.unit = nvgrid_pb2.LoadValue.UNIT_MEGAWATT
        target.load_target_strategy.best_effort = True
        target.feed_tags.append(feed_tag)
```

(continues on next page)

(continued from previous page)

```
        if correlation_id:
            target.correlation_id = f"{correlation_id}-{feed_tag}"

        print(f"Feed {feed_tag}: {target_watts/1_000_000:.2f}MW "
              f"({start_time.strftime('%H:%M')} - {end_time.strftime('%H:%M')}
↪)")

    response = stub.SetLoadTarget(request, metadata=metadata)

    if response.code == nvgrid_pb2.LoadTargetResponse.STATUS_CODE_SUCCESS:
        print("  Load target scheduled successfully")
        for detail in response.details:
            print(f"  Correlation ID: {detail.correlation_id}")
        return response
    else:
        print(f"  Failed: {response.diag_msg}")
        return None

# Usage
token = get_auth_token('admin', 'password', host='api.dps')
channel, stub = create_nvgrid_client(token, host='api.dps')
metadata = create_auth_metadata(token)

start_time = datetime.now() + timedelta(minutes=5)
schedule_load_target(stub, metadata, targets, start_time, 60, "curtailment-001
↪")

channel.close()
```

3.3.2.2.4.33 Step 4: Query Schedule

Retrieve current and future scheduled load targets.

3.3.2.2.4.34 Python Example

```
#!/usr/bin/env python3
"""
Step 4: Query load schedule
"""
from datetime import datetime, timedelta
from google.protobuf.timestamp_pb2 import Timestamp
from dpsapi.v1 import nvgrid_pb2
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

def get_load_schedule(stub, metadata, start_time, end_time, feed_tags=None):
    """Query load schedule for a time interval"""

    request = nvgrid_pb2.GetLoadScheduleRequest()
    request.interval.start_time.CopyFrom(Timestamp(seconds=int(start_time.
```

(continues on next page)

(continued from previous page)

```
↪timestamp())))
    request.interval.end_time.CopyFrom(Timestamp(seconds=int(end_time.
↪timestamp())))

    if feed_tags:
        request.feed_tags.extend(feed_tags)

    schedule = stub.GetLoadSchedule(request, metadata=metadata)

    print(f"Found {len(schedule.targets)} scheduled targets:")
    for i, target in enumerate(schedule.targets, 1):
        start_dt = datetime.fromtimestamp(target.interval.start_time.seconds)
        end_dt = datetime.fromtimestamp(target.interval.end_time.seconds) if
↪target.interval.HasField('end_time') else None

        print(f"\n{i}. {start_dt.strftime('%H:%M')} - {end_dt.strftime('%H:%M
↪') if end_dt else '∞'}")
        print(f"   Load: {target.load_constraint.value:.2f} MW")
        print(f"   Feeds: {', '.join(target.feed_tags)}")
        print(f"   ID: {target.correlation_id}")

    return schedule

# Usage
token = get_auth_token('admin', 'password', host='api.dps')
channel, stub = create_nvgrid_client(token, host='api.dps')
metadata = create_auth_metadata(token)

start = datetime.now()
end = start + timedelta(hours=4)
schedule = get_load_schedule(stub, metadata, start, end)

channel.close()
```

3.3.2.2.4.35 Step 5: Monitor Status

Track current power status and target achievement.

3.3.2.2.4.36 Python Example

```
#!/usr/bin/env python3
"""
Step 5: Monitor current status
"""
from google.protobuf import empty_pb2
from dpsapi.v1 import nvgrid_pb2
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

def get_current_status(stub, metadata):
    """Get current power status and target achievement"""
```

(continues on next page)

(continued from previous page)

```
    # Get detailed status
    status_response = stub.GetStatus(empty_pb2.Empty(), metadata=metadata)

    print(f"Current status for {len(status_response.statuses)} feeds:\n")

    for feed_tag, feed_status in status_response.statuses.items():
        # Check if target is met
        target_watts = load_value_to_watts(feed_status.current_load_target)
        calculated_watts = load_value_to_watts(feed_status.current_calculated_
↪load)
        target_met = calculated_watts <= target_watts if target_watts > 0
↪else True

        status_str = "ADJUSTING" if feed_status.in_flight else "STABLE"
        status_icon = " " if target_met else " "

        print(f"Feed: {feed_tag} - {status_str}")
        print(f"  Target: {target_watts/1_000_000:.2f} MW")
        print(f"  Actual: {calculated_watts/1_000_000:.2f} MW")
        print(f"  Status: {status_icon} {'Met' if target_met else 'Not Met'}")
        if feed_status.correlation_id:
            print(f"  ID: {feed_status.correlation_id}")
        print()

    return status_response.statuses

def load_value_to_watts(load_value):
    """Convert LoadValue to watts"""
    if not load_value or not load_value.value:
        return 0

    multipliers = {
        nvgrid_pb2.LoadValue.UNIT_MEGAWATT: 1_000_000,
        nvgrid_pb2.LoadValue.UNIT_KILOWATT: 1_000,
        nvgrid_pb2.LoadValue.UNIT_WATT: 1
    }
    return load_value.value * multipliers.get(load_value.unit, 1)

# Usage
token = get_auth_token('admin', 'password', host='api.dps')
channel, stub = create_nvgrid_client(token, host='api.dps')
metadata = create_auth_metadata(token)

status = get_current_status(stub, metadata)

channel.close()
```

3.3.2.2.4.37 Configuration and Setup

Choose the testing approach that fits your needs:

3.3.2.2.4.38 Option 1: Automated Testing (Recommended for Comprehensive Testing)

```
# Deploy environment and run grid simulation
task deploy:sdk                    # Creates cluster with simulated datacenter
task sim:grid END_AFTER=600 # Runs 10-minute grid integration test
```

Choose Your Simulation Type:

Command	What It Tests	Use When
`task sim:grid`	Grid integration only	Testing grid API and load scheduling in isolation
`task sim`	Grid + resource groups	Testing grid under realistic workload churn

Grid-only simulation:

```
task sim:grid END_AFTER=600  # 10 minutes, default load patterns
```

Schedules 2-5 concurrent load targets, randomizes patterns across power feeds.

Combined simulation:

```
task sim END_AFTER=600       # 10 minutes, grid + workload lifecycle
```

Tests grid integration while resource groups are continuously created and deleted.

Advanced Configuration:

```
# Extended test with high load range
task sim:grid END_AFTER=3600 MIN_LOAD_PERCENT=70 MAX_LOAD_PERCENT=95

# Frequent schedule changes for rapid testing
task sim:grid INTERVAL=60 MIN_DURATION=120 MAX_DURATION=300

# Combined test with aggressive workload churn
task sim END_AFTER=1800 MAX_RGS=25 MIN_DURATION=60 MAX_DURATION=180
```

Documentation: *Grid Simulation* • *Combined Simulation*

3.3.2.2.4.39 Option 2: Manual Testing (Useful for Learning and Debugging)

Step-by-step setup with manual API control:

```
# 1. Deploy DPS SDK
task deploy:sdk

# 2. Verify NvGrid API connection
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify nvgrid get-
```

(continues on next page)

(continued from previous page)

```
↪metadata

# 3. Query current status
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify nvgrid get-status
```

Manual API Commands:

```
# Get feed metadata
dpsctl nvgrid get-metadata

# Schedule a single load target (6MW for 1 hour)
dpsctl nvgrid set-load-target \
  --value=6 \
  --unit=megawatt \
  --start-time="2025-10-24T15:00:00Z" \
  --end-time="2025-10-24T16:00:00Z" \
  --best-effort=true

# Reset to default constraint (using --default)
dpsctl nvgrid set-load-target \
  --default \
  --start-time="2025-10-24T18:00:00Z" \
  --end-time="2025-10-24T20:00:00Z"

# Reset to default constraint (using --value=0, equivalent)
dpsctl nvgrid set-load-target \
  --value=0 \
  --unit=watt \
  --start-time="2025-10-24T18:00:00Z" \
  --end-time="2025-10-24T20:00:00Z"

# Query load schedule
dpsctl nvgrid get-schedule \
  --start-time="2025-10-24T14:00:00Z" \
  --end-time="2025-10-24T18:00:00Z"

# Get current load target
dpsctl nvgrid get-current

# Get current status
dpsctl nvgrid get-status
```

3.3.2.2.4.40 Observable Metrics

NvGrid exposes the following Prometheus metrics:

3.3.2.2.4.41 API Performance

Metric	Type	Labels	Description
`grpc_duration`	histogram (ms)	`method`, `status_code`	API response time
`grpc_requests`	counter	`method`, `status_code`	Total API request count

3.3.2.2.4.42 Power Management

Metric	Type	Labels	Description
`nvgrid_system_status_l`	gauge (W)	`feed_tag`, `load_unit`	Current load target
`nvgrid_system_status_c`	gauge (W)	`feed_tag`, `load_unit`	Actual power consumption
`nvgrid_system_status_i`	gauge	`feed_tag`	Power event status (0/1)
`nvgrid_feed_schedule_l`	gauge (W)	`feed_tag`, `start_time`, `end_time`, `load_unit`, `status`	Scheduled targets (status: `active`, `scheduled`, `expired`)

Derived Metrics: Compare target vs calculated load for constraint accuracy. Monitor schedule status transitions for adherence tracking.

Event Correlation: Query `grid_feed_schedule` and `nvgrid_system_status` database tables for `correlation_id` fields to track grid signals through the power management pipeline.

This power constraint management integration provides grid solutions with precise control over datacenter power consumption in response to utility grid signals.

3.3.2.2.4.43 NvGrid API Specification

The NvGrid gRPC API provides methods for scheduling power constraints, querying power status, and receiving power event notifications. This specification details all available endpoints, message types, and data structures.

3.3.2.2.4.44 Service Definition

```
service NvGrid {
  rpc SetLoadTarget(LoadTargetRequest) returns (LoadTargetResponse);
  rpc GetLoadSchedule(GetLoadScheduleRequest) returns (LoadSchedule);
```

(continues on next page)

(continued from previous page)

```
  rpc GetCurrentLoadTarget(google.protobuf.Empty)
↪returns(GetCurrentLoadTargetResponse);
  rpc GetMetadata(google.protobuf.Empty) returns (FeedMetadata);
  rpc RegisterWebhook(RegisterWebhookRequest) returns (LoadTargetResponse);
  rpc UnregisterWebhook(UnregisterWebhookRequest) returns
↪(LoadTargetResponse);
  rpc GetStatus(google.protobuf.Empty) returns (LoadTargetStatus);
}
```

3.3.2.2.4.45 Enums

3.3.2.2.4.46 StatusCode

Status codes indicate success or failure of API operations.

```
enum StatusCode {
  STATUS_CODE_UNSPECIFIED = 0;
  STATUS_CODE_SUCCESS = 1;
  STATUS_CODE_ERROR = 2;
}
```

Values:

- `STATUS_CODE_UNSPECIFIED` - Status code was not set
- `STATUS_CODE_SUCCESS` - Operation completed successfully
- `STATUS_CODE_ERROR` - Operation encountered an error

3.3.2.2.4.47 PowerEvent

Power events are used in webhook registration and webhook notifications.

```
enum PowerEvent {
  POWER_EVENT_UNSPECIFIED = 0;
  POWER_EVENT_START_RAMP_UP = 1;
  POWER_EVENT_END_RAMP_UP = 2;
  POWER_EVENT_START_RAMP_DOWN = 3;
  POWER_EVENT_END_RAMP_DOWN = 4;
}
```

Values:

- `POWER_EVENT_UNSPECIFIED` - Subscribe to all events when used with RegisterWebhook
- `POWER_EVENT_START_RAMP_UP` - Scheduled LoadTarget ramp up event has started
- `POWER_EVENT_END_RAMP_UP` - Scheduled LoadTarget ramp up event has completed
- `POWER_EVENT_START_RAMP_DOWN` - Scheduled LoadTarget ramp down event has started
- `POWER_EVENT_END_RAMP_DOWN` - Scheduled LoadTarget ramp down event has completed

3.3.2.2.4.48 Unit

Units of measurement for power values.

```
enum Unit {
  UNIT_UNSPECIFIED = 0;
  UNIT_WATT = 1;
  UNIT_KILOWATT = 2;
  UNIT_MEGAWATT = 3;
}
```

Values:

- `UNIT_UNSPECIFIED` - Unit not specified
- `UNIT_WATT` - Power measured in watts
- `UNIT_KILOWATT` - Power measured in kilowatts
- `UNIT_MEGAWATT` - Power measured in megawatts

3.3.2.2.4.49 RPC Methods

3.3.2.2.4.50 SetLoadTarget

Schedule power constraint changes with specified start and end times.

Request: `LoadTargetRequest` **Response:** `LoadTargetResponse`

Updates the future load target schedule using the provided list of LoadTarget objects. Each LoadTarget defines starting time, optional ending time, and target load. Setting target load to null resets to datacenter default. For overlapping time intervals, the most recently scheduled LoadTarget becomes effective.

Use cases:

- Schedule power reductions for demand response events
- Set power constraints based on utility curtailment signals
- Define power limits for planned maintenance windows

3.3.2.2.4.51 GetLoadSchedule

Retrieve scheduled load targets for a specified time interval and feeds.

Request: `GetLoadScheduleRequest` **Response:** `LoadSchedule`

Returns load schedules for specified power feeds within a given time interval. The schedule includes the LoadTarget active at the interval start and all future LoadTargets scheduled before the interval end.

Use cases:

- Query upcoming power constraint schedule
- Verify scheduled load targets
- Audit power constraint history

3.3.2.2.4.52 GetCurrentLoadTarget

Query the currently active load target for each power feed.

Request: `google.protobuf.Empty` **Response:** `GetCurrentLoadTargetResponse`

Returns the current load target for all configured power feeds, including target values, feed tags, and correlation identifiers.

Use cases:

- Check current power constraints
- Verify active load targets
- Monitor current constraint status

3.3.2.2.4.53 GetMetadata

Discover feed configurations, power thresholds, and default constraints.

Request: `google.protobuf.Empty` **Response:** `FeedMetadata`

Returns metadata for all configured power feeds, including minimum/maximum power thresholds and default constraints. Use this API to discover available feeds and their operational limits.

Use cases:

- Discover available power feeds
- Query power threshold limits
- Determine default power constraints

3.3.2.2.4.54 RegisterWebhook

Subscribe to power event notifications via HTTP webhook.

Request: `RegisterWebhookRequest` **Response:** `LoadTargetResponse`

Registers a webhook URL to receive notifications for specified power events. The webhook receives HTTP POST requests with event payloads when power state changes occur.

URL Whitelisting:

For security, DPS validates webhook URLs against a configured whitelist. RegisterWebhook will reject non-whitelisted URLs with error `STATUS_CODE_ERROR` and diagnostic message "webhook URL not whitelisted".

Whitelist configuration supports:

- Exact URL matches
- Path prefix matching (e.g., `https://example.com/api/` matches `https://example.com/api/webhooks/nvgrid`)
- Query parameter validation

- Scheme, hostname, and port enforcement

Configure the webhook whitelist in server configuration. See Prerequisites for detailed configuration instructions.

Use cases:

- Receive notifications of power ramp events
- Track in real-time
- Integrate with external monitoring systems

3.3.2.2.4.55 UnregisterWebhook

Unsubscribe from power event notifications.

Request: `UnregisterWebhookRequest` **Response:** `LoadTargetResponse`

Removes a previously registered webhook URL from power event notifications.

Use cases:

- Clean up webhook subscriptions
- Stop receiving event notifications
- Decommission integration endpoints

3.3.2.2.4.56 GetStatus

Query current power status including targets, calculated loads, and in-flight transitions.

Request: `google.protobuf.Empty` **Response:** `LoadTargetStatus`

Returns comprehensive status for all power feeds, including current targets, calculated loads, target achievement status, and whether power transitions are in progress.

Use cases:

- Monitor current power consumption
- Check target achievement status
- Track power transition progress

3.3.2.2.4.57 Message Types

3.3.2.2.4.58 LoadTargetRequest

Request message containing one or more load targets to schedule.

```
message LoadTargetRequest {
  repeated LoadTarget targets = 1;
}
```

Fields:

- `targets` - List of LoadTarget objects to schedule

3.3.2.2.4.59 LoadTarget

Defines a time-based power constraint with strategy and feed assignments.

```
message LoadTarget {
  Interval interval = 1;
  LoadValue load_constraint = 2;
  LoadTargetStrategy load_target_strategy = 3;
  repeated string feed_tags = 4;
  string correlation_id = 5;
}
```

Fields:

- `interval` - Start and end times for the constraint
- `load_constraint` - Target power limit (null removes constraint)
- `load_target_strategy` - Strategy for achieving target
- `feed_tags` - Power feeds to apply constraint (empty = all feeds)
- `correlation_id` - Unique identifier for tracking (auto-generated if omitted, max 36 bytes)

3.3.2.2.4.60 LoadTargetResponse

Response indicating success or failure with diagnostic information.

```
message LoadTargetResponse {
  message Detail {
    StatusCode code = 1;
    google.protobuf.Any details = 2;
    string correlation_id = 3;
  }

  StatusCode code = 1;
  string diag_msg = 2;
  repeated Detail details = 3;
}
```

Fields:

- `code` - Overall response status (refer to the StatusCode enum)
- `diag_msg` - Diagnostic message describing the failure
- `details` - List of detailed status messages per target

Detail Fields:

- `code` - Status code for this specific target
- `details` - Additional error details

- `correlation_id` - If the request is a load target request, the correlation IDs of the requests in the same order as the requests. If correlation IDs are not specified in the request, this will return the generated UUIDs for each request.

3.3.2.2.4.61 LoadTargetStrategy

Specifies the strategy for achieving power targets.

```
message LoadTargetStrategy {
  oneof strategy {
    bool best_effort = 1;
  }
}
```

Strategies:

`best_effort` - If power constraints require shedding load, DPS will reduce power from DPM-enabled jobs until the constraint is satisfied. If the constraint cannot be satisfied, DPS will reduce as much power as possible, but may not achieve the target until jobs terminate.

3.3.2.2.4.62 Interval

Time interval with optional start and end times.

```
message Interval {
  google.protobuf.Timestamp start_time = 1;
  google.protobuf.Timestamp end_time = 2;
}
```

Fields:

- `start_time` - Interval start (omitted = current time)
- `end_time` - Interval end (omitted = indefinite duration)

3.3.2.2.4.63 LoadValue

Power value with unit specification.

```
message LoadValue {
  double value = 1;
  Unit unit = 2;
}
```

Fields:

- `value` - Numeric power value
- `unit` - Power unit (refer to the Unit enum). This must be explicitly set.

3.3.2.2.4.64 GetLoadScheduleRequest

Request to query load schedule for specific feeds and time interval.

```
message GetLoadScheduleRequest {
  repeated string feed_tags = 1;
  Interval interval = 2;
}
```

Fields:

- `feed_tags` - Feed tags to query (empty = all feeds)
- `interval` - Time interval for schedule query

3.3.2.2.4.65 LoadSchedule

Ordered list of scheduled load targets.

```
message LoadSchedule {
  repeated LoadTarget targets = 1;
}
```

Fields:

- `targets` - Chronologically ordered LoadTarget objects

3.3.2.2.4.66 GetCurrentLoadTargetResponse

Current load target for each feed.

```
message GetCurrentLoadTargetResponse {
  map<string, LoadTarget> load_targets = 1;
}
```

Fields:

- `load_targets` - Map of feed tag to current LoadTarget

3.3.2.2.4.67 FeedMetadata

Metadata for all configured power feeds.

```
message FeedMetadata {
  message Entry {
    string feed_tag = 1;
    LoadValue power_minimum = 2;
    LoadValue power_maximum = 3;
    LoadValue default_constraint = 4;
  }
```

(continues on next page)

(continued from previous page)

```
  map<string, Entry> metadata = 1;
}
```

Fields:

- `metadata` - Map of feed tag to metadata entry

Entry fields:

- `feed_tag` - Feed identifier
- `power_minimum` - Minimum allowable power cap
- `power_maximum` - Maximum allowable power cap
- `default_constraint` - Default topology power constraint

3.3.2.2.4.68 RegisterWebhookRequest

Request to register webhook for power event notifications.

```
message RegisterWebhookRequest {
  repeated PowerEvent for_events = 1;
  string url = 2;
}
```

Fields:

- `for_events` - List of events to subscribe to (refer to the PowerEvent enum). If unspecified, subscribes to all power events.
- `url` - URL to register for notifications

3.3.2.2.4.69 UnregisterWebhookRequest

Request to unregister webhook.

```
message UnregisterWebhookRequest {
  string url = 1;
}
```

Fields:

- `url` - Webhook URL to unregister

3.3.2.2.4.70 LoadTargetStatus

Current system status including targets and calculated loads.

```
message LoadTargetStatus {
  message FeedStatus {
    LoadValue current_load_target = 1;
    LoadValue current_calculated_load = 2;
    bool in_flight = 3;
    google.protobuf.Timestamp power_event_start_time = 4;
    string correlation_id = 5;
  }

  map<string, FeedStatus> statuses = 1;
}
```

Fields:

- `statuses` - Map of feed tag to feed status

FeedStatus fields:

- `current_load_target` - Active power target from LoadTarget
- `current_calculated_load` - Calculated load by DPS (may differ from target and actual)
- `in_flight` - Power transition in progress
- `power_event_start_time` - Scheduled start time of current power event
- `correlation_id` - Correlation ID of active load target

3.3.2.2.4.71 Webhook Payload

Power event webhooks receive HTTP POST requests with JSON payloads:

```
{
  "event": "start_ramp_down",
  "start_time": "2025-01-24T17:00:00Z",
  "end_time": "2025-01-24T18:00:00Z",
  "strategy": "best_effort",
  "target_load": {
    "value": 6.5,
    "unit": "mw"
  },
  "prev_target_load": {
    "value": 10.0,
    "unit": "mw"
  },
  "calculated_load": {
    "value": 9.2,
    "unit": "mw"
  },
  "compliant": false,
  "correlation_id": "curtailment-20250101-170000"
}
```

Payload fields:

- `event` - Event type (start_ramp_up, end_ramp_up, start_ramp_down, end_ramp_down)

- `start_time` - LoadTarget scheduled start time
- `end_time` - LoadTarget scheduled end time (may be omitted)
- `strategy` - Strategy used to achieve target
- `target_load` - Target power constraint
- `prev_target_load` - Previous target before this change
- `calculated_load` - Current calculated load
- `compliant` - Whether calculated load meets target (calculated_load target_load). If true, the target was met. If false, the requested LoadTarget could not be met.
- `correlation_id` - Correlation ID from LoadTarget

3.3.2.2.4.72 API Usage Patterns

3.3.2.2.4.73 Schedule Constraint Pattern

```
1. GetMetadata() - Discover feeds and limits
2. Calculate target based on grid signal
3. SetLoadTarget() - Schedule constraint
4. GetLoadSchedule() - Verify schedule
```

3.3.2.2.4.74 Webhook Integration Pattern

```
1. RegisterWebhook() - Subscribe to events
2. Receive START_RAMP_DOWN event
3. Track power reduction progress
4. Receive END_RAMP_DOWN event
5. Verify target achievement status
6. UnregisterWebhook() - Optionally, clean up when done
```

3.3.2.2.4.75 Monitoring Pattern

```
1. GetCurrentLoadTarget() - Check active constraints
2. GetStatus() - Query current power state
3. Compare target vs calculated load
4. Verify target achievement
```

3.3.2.2.4.76 Error Handling

API methods return `LoadTargetResponse` with status codes:

- `STATUS_CODE_SUCCESS` - Operation completed successfully
- `STATUS_CODE_ERROR` - Operation failed (refer to diag_msg for details)

Check the `details` field for per-target status information when scheduling multiple load targets.

3.3.2.2.4.77 Protocol Buffer Definition

The complete API specification is defined in:

```
api/v1/nvgrid.proto
```

Generate client code using protocol buffer compiler:

```
# Python
python3 -m grpc_tools.protoc \
  -I./api/v1 \
  --python_out=./generated \
  --grpc_python_out=./generated \
  ./api/v1/nvgrid.proto

# Go
protoc --go_out=. --go-grpc_out=. api/v1/nvgrid.proto
```

3.3.2.2.4.78 Related Documentation

- ***Prerequisites*** - Setup requirements before using the API
- ***Managing Power Constraints*** - Schedule and manage load targets
- ***Monitoring Power Events*** - Implement webhook receivers

3.3.2.2.4.79 Monitoring Power Events

3.3.2.2.4.80 Overview

This use case demonstrates how to receive real-time notifications of power events using webhooks. Grid integrators can register webhooks to receive notifications when power ramp up/down events start and complete, enabling real-time tracking of datacenter power changes.

3.3.2.2.4.81 Goal

Receive real-time notifications when datacenter power changes occur, enabling grid solutions to track target achievement, report status to utilities, and coordinate with other grid services.

3.3.2.2.4.82 Persona: Grid Integration Engineer

Role: Developer integrating commercial grid solutions with DPS for datacenter power management.

Objective: Implement webhook receivers to track power events and maintain real-time awareness of datacenter power state.

3.3.2.2.4.83 Use Case Workflow

Scenario: Grid solution needs to track when power constraints are applied and report status to utility

Workflow Steps:

1. **Implement Webhook Receiver** - Create HTTP endpoint to receive power event notifications
2. **Register Webhook** - Subscribe to power events via NvGrid API

3. **Process Events** - Parse webhook payloads and extract power event data
4. **Unregister Webhook** - Clean up webhook subscription when no longer needed

---

3.3.2.2.4.84 Theory of Operation

3.3.2.2.4.85 Power Events

DPS NvGrid sends webhook notifications for four power events:

- **START_RAMP_UP**: Power constraint is relaxing (increasing power limit)
- **END_RAMP_UP**: Power ramp up complete, new higher limit reached
- **START_RAMP_DOWN**: Power constraint is tightening (decreasing power limit)
- **END_RAMP_DOWN**: Power ramp down complete, new lower limit reached

3.3.2.2.4.86 Webhook Lifecycle

3.3.2.2.4.87 Event Payload

Webhooks receive HTTP POST requests with JSON payloads:

```
{
  "event": "start_ramp_down",
  "feed_tag": "main_power_feed",
  "start_time": "2025-10-24T17:00:00Z",
  "end_time": "2025-10-24T18:00:00Z",
  "strategy": "best_effort",
  "target_load": {
    "value": 6.5,
    "unit": "mw"
  },
  "prev_target_load": {
    "value": 10.0,
    "unit": "mw"
  },
  "calculated_load": {
    "value": 9.2,
    "unit": "mw"
  },
  "compliant": false,
  "correlation_id": "curtailment-20251024-170000"
}
```

3.3.2.2.4.88 Webhook Integration Guide

3.3.2.2.4.89 Step 1: Implement Webhook Receiver

Create an HTTP endpoint that accepts POST requests from DPS. Your endpoint must:

- Accept HTTP POST requests with JSON payload

- Return 200 OK response for successful receipt
- Parse and process the event payload (structure shown in the preceding example)

Minimal Example:

```
# Minimal webhook endpoint example (Flask)
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/nvgrid/webhook', methods=['POST'])
def receive_power_event():
    """Receive power event from DPS NvGrid"""
    event_data = request.get_json()

    # Extract key fields
    event_type = event_data.get('event')
    correlation_id = event_data.get('correlation_id')
    target_load = event_data.get('target_load', {})
    calculated_load = event_data.get('calculated_load', {})

    # Your processing logic here
    print(f"Received {event_type}: {target_load.get('value')} {target_load.
↪get('unit')}")

    return jsonify({'status': 'success'}), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

Implementation is grid integrator's responsibility. Process events according to your grid integration requirements.

3.3.2.2.4.90 Step 2: Register Webhook

Subscribe to power events by registering the webhook URL.

3.3.2.2.4.91 Python Example

Note: Complete Python setup (installation, CA cert, authentication) is documented in *Managing Power Constraints - Python Client Setup*.

```
#!/usr/bin/env python3
"""
Register webhook for NvGrid power events
"""
from dpsapi.v1 import nvgrid_pb2
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

def register_webhook(stub, metadata, webhook_url, events=None):
    """Register webhook for power event notifications
```

(continues on next page)

(continued from previous page)

```
    Args:
        stub: NvGrid client stub
        metadata: Auth metadata
        webhook_url: URL to receive webhooks
        events: List of event names to subscribe to (None = all)

    Returns:
        True if successful
    """
    request = nvgrid_pb2.RegisterWebhookRequest(url=webhook_url)

    # Map event names to proto enums
    event_map = {
        'start_ramp_up': nvgrid_pb2.RegisterWebhookRequest.POWER_EVENT_START_
↪RAMP_UP,
        'end_ramp_up': nvgrid_pb2.RegisterWebhookRequest.POWER_EVENT_END_RAMP_
↪UP,
        'start_ramp_down': nvgrid_pb2.RegisterWebhookRequest.POWER_EVENT_
↪START_RAMP_DOWN,
        'end_ramp_down': nvgrid_pb2.RegisterWebhookRequest.POWER_EVENT_END_
↪RAMP_DOWN,
    }

    if events:
        request.for_events.extend([event_map[e] for e in events])
        print(f"Subscribing to: {', '.join(events)}")
    else:
        print("Subscribing to all power events")

    response = stub.RegisterWebhook(request, metadata=metadata)

    if response.code == nvgrid_pb2.LoadTargetResponse.STATUS_CODE_SUCCESS:
        print(f"  Webhook registered: {webhook_url}")
        return True
    else:
        print(f"  Failed: {response.diag_msg}")
        return False

# Usage
token = get_auth_token('admin', 'password', host='api.dps')
channel, stub = create_nvgrid_client(token, host='api.dps')
metadata = create_auth_metadata(token)

# Register for all events
register_webhook(stub, metadata, 'http://grid-solution:8080/nvgrid/webhook')

# Or register for specific events
register_webhook(stub, metadata, 'http://grid-solution:8080/nvgrid/webhook',
                 events=['start_ramp_down', 'end_ramp_down'])

channel.close()
```

3.3.2.2.4.92 Step 3: Process Webhook Events

Your webhook receiver should process the JSON payload to extract relevant information for your grid integration needs. Key fields available in the payload:

- **`event`**: Event type (`start_ramp_down`, `end_ramp_down`, `start_ramp_up`, `end_ramp_up`)
- **`correlation_id`**: Tracking ID linking related events
- **`target_load`**: Target power level (`value`, `unit`)
- **`calculated_load`**: Actual power consumption (`value`, `unit`)
- **`prev_target_load`**: Previous target before this change
- **`compliant`**: Whether calculated load meets target (boolean)
- **`start_time`**, **`end_time`**: Schedule times
- **`strategy`**: Load targeting strategy (e.g., `best_effort`)

Process these fields according to your utility reporting and compliance requirements.

3.3.2.2.4.93 Step 4: Unregister Webhook

Clean up webhook subscription when no longer needed.

3.3.2.2.4.94 Python Example

Note: Refer to *Managing Power Constraints - Python Client Setup* for installation and authentication setup.

```
#!/usr/bin/env python3
"""
Unregister webhook from NvGrid
"""
from dpsapi.v1 import nvgrid_pb2
from nvgrid_client import get_auth_token, create_nvgrid_client, create_auth_
↪metadata

def unregister_webhook(stub, metadata, webhook_url):
    """Unregister webhook

    Args:
        stub: NvGrid client stub
        metadata: Auth metadata
        webhook_url: URL to unregister

    Returns:
        True if successful
    """
    request = nvgrid_pb2.UnregisterWebhookRequest(url=webhook_url)
    response = stub.UnregisterWebhook(request, metadata=metadata)

    if response.code == nvgrid_pb2.LoadTargetResponse.STATUS_CODE_SUCCESS:
        print(f"  Webhook unregistered: {webhook_url}")
```

(continues on next page)

(continued from previous page)

```
        return True
    else:
        print(f"  Failed: {response.diag_msg}")
        return False

# Usage
token = get_auth_token('admin', 'password', host='api.dps')
channel, stub = create_nvgrid_client(token, host='api.dps')
metadata = create_auth_metadata(token)

unregister_webhook(stub, metadata, 'http://grid-solution:8080/nvgrid/webhook')

channel.close()
```

3.3.2.2.4.95 Testing Webhooks

You can test webhook delivery using curl:

```
# Send test webhook payload to your endpoint
curl -X POST http://localhost:8080/nvgrid/webhook \
  -H "Content-Type: application/json" \
  -d '{
    "event": "start_ramp_down",
    "start_time": "2025-10-24T17:00:00Z",
    "end_time": "2025-10-24T18:00:00Z",
    "strategy": "best_effort",
    "target_load": {"value": 6.5, "unit": "mw"},
    "prev_target_load": {"value": 10.0, "unit": "mw"},
    "calculated_load": {"value": 9.8, "unit": "mw"},
    "compliant": false,
    "correlation_id": "test-001"
  }'
```

This webhook integration enables real-time awareness of datacenter power changes, allowing grid solutions to track compliance and report status to utilities effectively.

3.3.2.2.4.96 Core Concepts

This section covers fundamental architectural principles and operational behaviors of the NvGrid system for grid integration.

3.3.2.2.4.97 Architectural Principles

3.3.2.2.4.98 Standards-Based APIs

NvGrid integrations use standardized gRPC APIs with protobuf message definitions. This approach provides language-agnostic integration, strong typing, and efficient serialization for communication between grid solutions and DPS.

3.3.2.2.4.99 Event-Driven Architecture

Grid solutions receive real-time power event notifications through webhook systems. This enables immediate awareness of power state changes without polling, supporting responsive integration with utility grid operations.

3.3.2.2.4.100 Best Effort Power Management

DPS manages power constraints without terminating running workloads. The system reduces power from Dynamic Power Management (DPM) enabled jobs while maintaining operational continuity, pursuing power targets without disrupting datacenter operations.

3.3.2.2.4.101 Multi-Feed Architecture

NvGrid supports granular control over individual power feeds or entire datacenters. Load targets can be applied to specific electrical circuits or across all feeds, providing flexible power management aligned with utility infrastructure.

3.3.2.2.4.102 Simulator-First Development

Grid integrations are developed and tested using DPS simulators. This provides consistent testing environments independent of actual utility connections, enabling development and validation without requiring live grid access.

3.3.2.2.4.103 Power Management Strategy

The NvGrid system implements a best effort strategy for power constraint enforcement.

3.3.2.2.4.104 Constraint Application

DPM-Enabled Jobs

Power reduction targets Dynamic Power Management (DPM) enabled workloads first. These workloads can have their power limits adjusted without termination.

Non-Terminating Approach

Running jobs are never terminated to meet power constraints. The system prioritizes operational continuity over strict constraint adherence.

Gradual Reduction

Power is reduced incrementally rather than abruptly:

1. System identifies DPM-enabled workloads with available power headroom
2. Power limits are reduced in stages across identified workloads
3. Reduction continues until constraint is satisfied or workload minimums are reached
4. System monitors actual power consumption throughout the process

Target Achievement Tracking

System continuously tracks whether actual power consumption meets target constraints. Non-achievement is acceptable under the best effort model when constraints require job termination.

3.3.2.2.4.105 Power Restoration

Automatic Recovery

When power constraints are relaxed or expire, DPS automatically attempts to restore power to DPM-enabled jobs. Power may be distributed to different workloads than those originally reduced.

Power Distribution

Power restoration follows DPS power management policies:

- DPM-enabled jobs that had power reduced receive increased allocations
- Power distribution respects workload power policies and limits
- Restoration occurs incrementally to maintain system stability
- Topology and resource group constraints govern power allocation

Restoring Default Schedule

To explicitly return to default constraints before a scheduled target expires, schedule a zero watt target for the time period. This cancels any active targets in that time range, allowing the system to revert to default constraints.

3.3.2.2.4.106 Feed Management

Datacenters are powered by one or more electrical feeds, each representing a distinct circuit. NvGrid provides independent control over each feed.

3.3.2.2.4.107 Power Feeds

Each feed has operational boundaries defined by metadata:

- **Feed Tag**: Unique identifier (e.g., “feed-a”)
- **Power Limits**: Minimum and maximum power caps
- **Default Constraint**: Power limit when no targets are active

3.3.2.2.4.108 Targeting and Control

Load targets can be applied to specific feeds or all feeds:

- **Specific targeting**: Apply constraints to selected feeds only
- **Global targeting**: Apply constraints across all feeds
- **Independent operation**: Each feed operates according to its active target

For overlapping time periods, the most recently scheduled load target takes precedence.

3.3.2.2.4.109 Next Steps

With understanding of these core concepts, proceed to:

- ***API Specification*** - Detailed API reference
- ***Managing Power Constraints*** - Constraint scheduling workflows
- ***Monitoring Power Events*** - Webhook implementation

3.3.2.2.4.110 Optimization Developer SDK

3.3.2.2.4.111 Overview

The Optimization Developer SDK provides tools for developing power and performance optimization solutions within NVIDIA's datacenter infrastructure. This SDK enables developers to create systems that optimize power distribution and performance characteristics across datacenter resources.

3.3.2.2.4.112 Target Persona

Optimization Developer: Developer creating power/performance optimization solutions.

Capabilities:

- Entity configuration for performance and power optimization
- Power distribution algorithms
- Performance monitoring and tuning
- Integration with NVIDIA optimization frameworks

3.3.2.2.4.113 SDK Components

3.3.2.2.4.114 Core Use Cases

- ***Power Distribution Optimization*** - Test optimization integration - Utilize DPS interfaces - Observe utilization at aggregate/node/GPU levels

3.3.2.2.4.115 Getting Started

Use the DPS API to create resource groups with selected policy bundles and monitor power consumption at aggregate, node, and GPU levels.

3.3.2.2.4.116 Power Distribution Optimization Development

3.3.2.2.4.117 Overview

This use case demonstrates power distribution optimization integration with DPS.

3.3.2.2.4.118 Goal

Test integration of algorithms that distribute power across datacenter resources.

3.3.2.2.4.119 Workflow Steps

1. **Enable Optimization Integration** - Utilize DPS interfaces for power limit changes
2. **Implement Power Distribution Algorithm** - Distribute power across datacenter equipment
3. **Observe Aggregate Power Utilization** - Monitor total datacenter power consumption
4. **Observe Node-Level Power Utilization** - Monitor individual node power patterns
5. **Observe GPU-Level Power Utilization** - Monitor detailed GPU power and compare results

3.3.2.2.4.120 SDK Configuration

Configure your DPS environment with:

- Topology file with your datacenter nodes
- DPS configuration (dps-values.yaml)
- BMC simulator configuration (bmc-sim-values.yaml)

3.3.2.2.4.121 Environment Setup

```
# Deploy DPS with your configuration
helm upgrade --reuse-values dps -f <your-dps-values.yaml>

# Login to DPS
printf '%s\n' '<password>' | dpsctl --host api.dps --port 443 --insecure-tls-
↪skip-verify login --username <username> --password-stdin

# Import and activate your topology
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify topology import
↪<topology-file.json>
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify topology activate
↪--topology <topology-name>
```

3.3.2.2.4.122 Observable Metrics

3.3.2.2.4.123 Aggregate Power Distribution Metrics

- **Total Power Consumption**: Aggregate power across all managed nodes
- **Power Distribution Variance**: Variance in power allocation across nodes
- **Power Utilization Rate**: Actual vs available power capacity utilization
- **Distribution Balance**: Power distribution balance based on workload requirements

3.3.2.2.4.124 Node-Level Power Metrics

- **Per-Node Power Consumption**: Individual node power usage patterns
- **Power Policy Compliance**: Adherence to assigned power policies
- **Node Power Stability**: Consistency of power consumption over time
- **Inter-Node Power Balance**: Power distribution balance across nodes

3.3.2.2.4.125 GPU-Level Power Metrics

- **Per-GPU Power Utilization**: Individual GPU power consumption
- **Intra-Node GPU Balance**: Power distribution among GPUs within nodes
- **GPU Power Stability**: Stability of GPU power consumption

3.3.2.2.4.126 Command Line Example

```
# Create resource group with a policy bundle
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify resource-group
↪create --resource-group opt-test --external-id 1 --policy-bundle MaxP

# Add nodes to the resource group
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify resource-group
↪add --resource-group opt-test --entities <node1>,<node2>

# Activate resource group to apply power policy
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify resource-group
↪activate --sync --resource-group opt-test

# Monitor power consumption
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify check metrics --
↪nodes <node1>,<node2>

# Clean up when done
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify resource-group
↪delete --resource-group opt-test
```

3.3.2.2.4.127 API Integration Example (Python)

```
from dpsapi.api import DpsApi

# Initialize DPS API
api = DpsApi(
    dps_host="api.dps",
    dps_port=443,
    dps_username="<username>",
    dps_password="<password>",
)

# Create resource group with a policy bundle
api.resource_groups.create_resource_group(
    external_id=1,
    resource_group_name="opt-test",
    policy_bundle="MaxP",
)

# Add nodes to resource group
api.resource_groups.add_resources_to_group(
    resource_group_name="opt-test",
    entity_names=["<node-name>"],
)

# Activate resource group to apply power policy
api.resource_groups.activate_resource_group("opt-test")

# Monitor power consumption
metrics = api.metrics.request_metrics(requested_gpus=[("<node-name>", 0)])
```

(continues on next page)

(continued from previous page)

```
# Clean up when done
api.resource_groups.delete_resource_group("opt-test")
```

3.3.2.2.4.128 Testing and Validation

Test the workflow by:

1. Creating resource groups with different policy bundles
2. Monitoring power consumption at aggregate, node, and GPU levels
3. Observing power distribution across your datacenter resources

3.3.2.2.4.129 Workload Scheduling Integrator SDK

3.3.2.2.4.130 Overview

The Workload Scheduling Integrator SDK provides tools for integrating workload scheduling solutions into datacenter power management. This SDK enables developers to create power management workflows that coordinate between job schedulers (like Slurm) and DPS power policies.

3.3.2.2.4.131 Target Persona

Workload Scheduling Integrator: Integration engineer developing solutions that connect workload schedulers with datacenter power management systems.

Capabilities:

- Resource allocation and power policy coordination
- Workload lifecycle management with power awareness
- Priority-based power allocation handling
- Power policy application and cleanup

3.3.2.2.4.132 SDK Components

3.3.2.2.4.133 Core Use Cases

- ***Scheduling Workload Integration*** - Create integration components (Slurm prolog/epilog) - Transform node lists - Enable integration - Monitor dashboards

3.3.2.2.4.134 Getting Started

Implement scheduler integration by creating prolog/epilog scripts that interact with DPS resource group APIs to apply power policies to scheduled workloads.

3.3.2.2.4.135 Scheduling Workload Integration

3.3.2.2.4.136 Overview

This use case demonstrates workload scheduling integration with DPS. Integrators can test their scheduling solutions using the simulator environment to verify operation, observe power utilization compliance, and identify job scheduling issues.

3.3.2.2.4.137 Goal

Test workload scheduling capabilities with DPS using a simulator environment for validation independent of specific hardware configurations.

3.3.2.2.4.138 Persona: Workload Scheduling Integrator

Role: Integration engineer developing solutions to integrate workload schedulers with DPS power management system.

Objective: Integrate scheduling solution into DPS for power-aware workload management using DPS simulator.

3.3.2.2.4.139 Use Case Workflow

Scenario: Workload Scheduling Integrator plans to integrate their scheduling solution into DPS

Workflow Steps:

1. **Create Integration Components** - Create Slurm prolog/epilog scripts to utilize DPS interfaces
2. **Transform Simulated Node List** - Transform the simulated node list in the DPS simulator into scheduler resource pool
3. **Enable Integration** - Enable integration and verify operation
4. **Observe Job Scheduling Failures** - Observe DPS simulator dashboards for indications of job scheduling failures
5. **Observe Power Utilization Compliance** - Observe DPS simulator dashboards for power utilization and compliance status

---

3.3.2.2.4.140 Theory of Operation

3.3.2.2.4.141 Scheduler Integration Architecture

3.3.2.2.4.142 Resource Group Lifecycle

Scheduler integration implements a resource group lifecycle:

1. **Job Submission** - Scheduler receives job and allocates nodes
2. **Prolog Execution** - Create a DPS resource group with allocated nodes and the selected policy bundle
3. **Job Activation** - Activate the resource group to apply its effective policies to hardware
4. **Job Monitoring** - Monitor power consumption and compliance during job execution
5. **Job Completion** - Execute epilog to clean up resource group and restore default policies

---

3.3.2.2.4.143 Simulator Configuration

Configure your DPS environment with:

- Topology file defining your datacenter nodes and power distribution
- BMC simulator configuration for your hardware
- Policy bundles appropriate for your workload types

Names such as `Node-High`, `ML-Training-High`, and `GB200-High` in the following examples are site-defined policy bundle names, not built-in named policies. Provision the corresponding bundles and verify them with `dpsctl policy bundle list`, or replace the names with installed bundles. The released Go CLI selects the result with `--policy-bundle`.

3.3.2.2.4.144 Step-by-Step Integration Guide

3.3.2.2.4.145 Step 1: Create Integration Components

Create Slurm prolog and epilog scripts that interface with DPS APIs.

3.3.2.2.4.146 Prolog Script Implementation

```
#!/bin/bash
# slurm_dps_prolog.sh
# Slurm Prolog script for DPS integration

set -euo pipefail

# Configuration
DPS_API_ENDPOINT="${DPS_API_ENDPOINT:-api.dps:443}"
DPS_USERNAME="${DPS_USERNAME:-slurm-integrator}"
DPS_PASSWORD="${DPS_PASSWORD:-slurm-password}"
DPSCTL_PATH="${DPSCTL_PATH:-dpsctl}"

# Slurm job environment variables (automatically set by Slurm)
JOB_ID="${SLURM_JOB_ID}"
JOB_NAME="${SLURM_JOB_NAME:-job-$JOB_ID}"
NODE_LIST="${SLURM_JOB_NODELIST}"
JOB_PARTITION="${SLURM_JOB_PARTITION}"
JOB_QOS="${SLURM_JOB_QOS:-normal}"

# Log file for debugging
LOG_FILE="/var/log/slurm/dps_prolog.log"

log() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - PROLOG - JOB $JOB_ID - $1" >> "$LOG_
↪FILE"
}

log "Starting DPS integration for job $JOB_ID"
log "Job details: name=$JOB_NAME, partition=$JOB_PARTITION, qos=$JOB_QOS"
log "Allocated nodes: $NODE_LIST"
```

(continues on next page)

(continued from previous page)

```
# Function to determine power policy based on job characteristics
determine_power_policy() {
    local job_name="$1"
    local partition="$2"
    local qos="$3"

    # Policy selection logic
    case "$partition" in
        "gpu-high"|"training")
            if [[ "$job_name" =~ ml-training|training|llm ]]; then
                echo "ML-Training-High"
            else
                echo "Node-High"
            fi
            ;;
        "gpu-standard"|"inference")
            echo "Inference-Medium"
            ;;
        "gpu-dev"|"development"|"debug")
            echo "Development-Low"
            ;;
        *)
            # Default policy based on QOS
            case "$qos" in
                "high"|"urgent")
                    echo "Node-High"
                    ;;
                "low"|"preempt")
                    echo "Development-Low"
                    ;;
                *)
                    echo "Inference-Medium"
                    ;;
            esac
            ;;
    esac
}

# Function to expand Slurm node list format
expand_node_list() {
    local node_list="$1"

    # Use Slurm's scontrol to expand compressed node list
    if command -v scontrol >/dev/null 2>&1; then
        scontrol show hostnames "$node_list" | tr '\n' ',' | sed 's/,$//'
    else
        # Fallback: assume simple comma-separated list
        echo "$node_list" | tr ' ' ','
    fi
}
```

(continues on next page)

(continued from previous page)

```
# Function to create DPS resource group
create_dps_resource_group() {
    local job_id="$1"
    local policy="$2"
    local nodes="$3"

    local group_name="slurm-job-$job_id"

    log "Creating DPS resource group: $group_name"
    log "Policy: $policy, Nodes: $nodes"

    # Build dpsctl command
    local cmd=(
        "$DPSCTL_PATH" "resource-group" "create"
        "--resource-group=$group_name"
        "--external-id=$job_id"
        "--policy-bundle=$policy"
    )

    # Execute resource group creation
    if "${cmd[@]}" 2>&1 | tee -a "$LOG_FILE"; then
        log "Resource group $group_name created successfully"

        # Add allocated nodes
        if ! "$DPSCTL_PATH" resource-group add \
            --resource-group "$group_name" \
            --entities "$nodes" 2>&1 | tee -a "$LOG_FILE"; then
            log "ERROR: Failed to add nodes to resource group $group_name"
            return 1
        fi

        # Activate the resource group
        if "$DPSCTL_PATH" resource-group activate --resource-group "$group_
↪name" --sync 2>&1 | tee -a "$LOG_FILE"; then
            log "Resource group $group_name activated successfully"
            return 0
        else
            log "ERROR: Failed to activate resource group $group_name"
            return 1
        fi
    else
        log "ERROR: Failed to create resource group $group_name"
        return 1
    fi
}

# Main execution
main() {
    # Validate required environment
    if [[ -z "$JOB_ID" || -z "$NODE_LIST" ]]; then
        log "ERROR: Missing required Slurm environment variables"
        exit 1
```

(continues on next page)

(continued from previous page)

```
    fi

    # Check dpsctl availability
    if ! command -v "$DPSCTL_PATH" >/dev/null 2>&1; then
        log "ERROR: dpsctl not found at $DPSCTL_PATH"
        exit 1
    fi

    # Determine appropriate power policy
    POWER_POLICY=$(determine_power_policy "$JOB_NAME" "$JOB_PARTITION" "$JOB_
↪QOS")
    log "Selected power policy: $POWER_POLICY"

    # Expand node list
    EXPANDED_NODES=$(expand_node_list "$NODE_LIST")
    log "Expanded node list: $EXPANDED_NODES"

    # Create and activate DPS resource group
    if create_dps_resource_group "$JOB_ID" "$POWER_POLICY" "$EXPANDED_NODES";
↪then
        log "DPS integration completed successfully for job $JOB_ID"
        exit 0
    else
        log "ERROR: DPS integration failed for job $JOB_ID"
        exit 1
    fi
}

# Execute main function
main "$@"
```

3.3.2.2.4.147 Epilog Script Implementation

```
#!/bin/bash
# slurm_dps_epilog.sh
# Slurm Epilog script for DPS cleanup

set -euo pipefail

# Configuration
DPSCTL_PATH="${DPSCTL_PATH:-dpsctl}"

# Slurm job environment variables
JOB_ID="${SLURM_JOB_ID}"
JOB_NAME="${SLURM_JOB_NAME:-job-$JOB_ID}"

# Log file for debugging
LOG_FILE="/var/log/slurm/dps_epilog.log"

log() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - EPILOG - JOB $JOB_ID - $1" >> "$LOG_
```

(continues on next page)

(continued from previous page)

```
↪FILE"
}

log "Starting DPS cleanup for job $JOB_ID"

# Function to delete DPS resource group
cleanup_dps_resource_group() {
    local job_id="$1"
    local group_name="slurm-job-$job_id"

    log "Cleaning up DPS resource group: $group_name"

    # Check if resource group exists
    if $DPSCTL_PATH resource-group list 2>/dev/null | grep -q "$group_name";
↪then
        # Delete the resource group
        if "$DPSCTL_PATH" resource-group delete --resource-group "$group_name
↪" 2>&1 | tee -a "$LOG_FILE"; then
            log "Resource group $group_name deleted successfully"
            return 0
        else
            log "WARNING: Failed to delete resource group $group_name"
            return 1
        fi
    else
        log "Resource group $group_name not found (may have already been
↪cleaned up)"
        return 0
    fi
}

# Main execution
main() {
    # Validate required environment
    if [[ -z "$JOB_ID" ]]; then
        log "ERROR: Missing required SLURM_JOB_ID environment variable"
        exit 1
    fi

    # Check dpsctl availability
    if ! command -v "$DPSCTL_PATH" >/dev/null 2>&1; then
        log "ERROR: dpsctl not found at $DPSCTL_PATH"
        exit 1
    fi

    # Cleanup DPS resource group
    if cleanup_dps_resource_group "$JOB_ID"; then
        log "DPS cleanup completed successfully for job $JOB_ID"
        exit 0
    else
        log "WARNING: DPS cleanup had issues for job $JOB_ID (continuing
↪anyway)"
```

(continues on next page)

(continued from previous page)

```
        exit 0  # Do not fail the job due to cleanup issues
    fi
}

# Execute main function
main "$@"
```

3.3.2.2.4.148 Integration Helper (Python)

```
#!/usr/bin/env python3
"""
Slurm-DPS Integration Helper
Integration functions for Slurm scheduler with DPS
"""

import os
import json
import logging
import subprocess
import time
from typing import Dict, List, Optional
from dpsapi.api import DpsApi

class SlurmDPSIntegrator:
    def __init__(self,
                 dps_endpoint: str = "api.dps:443",
                 dpsctl_path: str = "dpsctl"):

        self.logger = logging.getLogger(__name__)
        self.dpsctl_path = dpsctl_path

        # Initialize DPS API connection
        endpoint_parts = dps_endpoint.split(':')
        self.dps_api = DpsApi(
            dps_host=endpoint_parts[0],
            dps_port=int(endpoint_parts[1]),
            dps_username="slurm-integrator",
            dps_password="slurm-password",
            dps_insecure_tls_skip_verify=True,
        )

        # Job type to policy mapping
        self.policy_mapping = {
            'ml-training': 'GB200-High',
            'inference': 'GB200-Med',
            'development': 'GB200-Low',
            'simulation': 'GB200-High',
            'data-processing': 'GB200-Med'
        }

    def analyze_job_requirements(self, job_info: Dict) -> Dict:
```

(continues on next page)

(continued from previous page)

```
        """Analyze job requirements to determine power policy"""

        job_name = job_info.get('name', '').lower()
        partition = job_info.get('partition', '').lower()
        qos = job_info.get('qos', 'normal').lower()
        time_limit = job_info.get('time_limit', '01:00:00')

        # Analyze job characteristics
        analysis = {
            'job_id': job_info.get('job_id'),
            'job_name': job_name,
            'partition': partition,
            'qos': qos,
            'time_limit': time_limit,
            'nodes': job_info.get('nodes', []),
            'estimated_duration_hours': self._parse_time_limit(time_limit)
        }

        # Determine job type from name and partition
        job_type = 'development'  # default

        if any(keyword in job_name for keyword in ['training', 'train', 'llm',
↪ 'gpt']):
            job_type = 'ml-training'
        elif any(keyword in job_name for keyword in ['inference', 'infer',
↪'serving']):
            job_type = 'inference'
        elif any(keyword in job_name for keyword in ['simulation', 'sim', 'hpc
↪']):
            job_type = 'simulation'
        elif any(keyword in job_name for keyword in ['data', 'process', 'etl
↪']):
            job_type = 'data-processing'
        elif partition in ['gpu-high', 'training'] or qos in ['high', 'urgent
↪']:
            job_type = 'ml-training'

        analysis['job_type'] = job_type
        analysis['recommended_policy'] = self.policy_mapping.get(job_type,
↪'GB200-Med')

        # Adjust policy based on duration for long-running jobs
        if analysis['estimated_duration_hours'] > 12:
            if job_type in ['ml-training', 'simulation']:
                analysis['recommended_policy'] = 'GB200-High'
            else:
                analysis['recommended_policy'] = 'GB200-Med'

        self.logger.info(f"Job analysis complete: {analysis['job_type']} ->
↪{analysis['recommended_policy']}")
        return analysis
```

(continues on next page)

(continued from previous page)

```
    def _parse_time_limit(self, time_limit: str) -> float:
        """Parse Slurm time limit format to hours"""
        try:
            # Handle formats like "01:00:00", "2-12:00:00", "60"
            if '-' in time_limit:
                days, time_part = time_limit.split('-')
                hours_from_days = int(days) * 24
            else:
                hours_from_days = 0
                time_part = time_limit

            if ':' in time_part:
                parts = time_part.split(':')
                if len(parts) == 3:  # HH:MM:SS
                    hours = int(parts[0]) + int(parts[1])/60 + int(parts[2])/
↪3600
                elif len(parts) == 2:  # HH:MM
                    hours = int(parts[0]) + int(parts[1])/60
                else:
                    hours = int(parts[0])
            else:
                # Assume minutes if no colons
                hours = int(time_part) / 60

            return hours_from_days + hours
        except:
            return 1.0  # default to 1 hour if parsing fails

    def create_resource_group_for_job(self, job_analysis: Dict) -> bool:
        """Create resource group for job"""

        job_id = job_analysis['job_id']
        policy = job_analysis['recommended_policy']
        nodes = job_analysis['nodes']

        group_name = f"slurm-job-{job_id}"

        try:
            self.logger.info(f"Creating resource group: {group_name}")

            # Build dpsctl command
            cmd = [
                self.dpsctl_path, "resource-group", "create",
                f"--resource-group={group_name}",
                f"--external-id={job_id}",
                f"--policy-bundle={policy}"
            ]

            # Execute command
            result = subprocess.run(cmd, capture_output=True, text=True)

            if result.returncode == 0:
```

(continues on next page)

(continued from previous page)

```
                self.logger.info(f"Resource group {group_name} created
↪successfully")

                add_cmd = [
                    self.dpsctl_path, "resource-group", "add",
                    f"--resource-group={group_name}",
                    "--entities=" + ",".join(nodes)
                ]
                add_result = subprocess.run(
                    add_cmd, capture_output=True, text=True
                )
                if add_result.returncode != 0:
                    self.logger.error(
                        f"Failed to add nodes to resource group: {add_result.
↪stderr}"
                    )
                    return False

                # Activate with monitoring
                return self._activate_with_monitoring(group_name, job_
↪analysis)
            else:
                self.logger.error(f"Failed to create resource group: {result.
↪stderr}")
                return False

        except Exception as e:
            self.logger.error(f"Error creating resource group: {e}")
            return False

    def _activate_with_monitoring(self, group_name: str, job_analysis: Dict) -
↪> bool:
        """Activate resource group with power monitoring"""

        try:
            # Activate resource group
            cmd = [
                self.dpsctl_path, "resource-group", "activate",
                f"--resource-group={group_name}", "--sync"
            ]
            result = subprocess.run(cmd, capture_output=True, text=True)

            if result.returncode != 0:
                self.logger.error(f"Failed to activate resource group:
↪{result.stderr}")
                return False

            self.logger.info(f"Resource group {group_name} activated
↪successfully")

            # Monitor initial power consumption
            time.sleep(5)  # Allow activation to take effect
```

(continues on next page)

(continued from previous page)

```
            power_metrics = self._get_initial_power_metrics(job_analysis[
↪'nodes'])
            if power_metrics:
                self.logger.info(f"Initial power consumption: {power_metrics[
↪'total_power']:.1f}W "
                                 f"across {len(job_analysis['nodes'])} GB200
↪nodes")

            return True

        except Exception as e:
            self.logger.error(f"Error during activation monitoring: {e}")
            return False

    def _get_initial_power_metrics(self, nodes: List[str]) -> Optional[Dict]:
        """Get initial power metrics for monitoring"""
        try:
            gpu_list = []
            for node in nodes:
                for gpu_id in range(4):  # DGX_GB200 has 4 GPUs
                    gpu_list.append([node, gpu_id])

            metrics_response = self.dps_api.metrics.request_metrics(requested_
↪gpus=gpu_list)

            if 'metrics' in metrics_response:
                total_power = sum(metric['usage'] for metric in metrics_
↪response['metrics'])
                return {
                    'total_power': total_power,
                    'node_count': len(nodes),
                    'gpu_count': len(gpu_list),
                    'metrics': metrics_response['metrics']
                }

        except Exception as e:
            self.logger.warning(f"Could not retrieve initial power metrics:
↪{e}")

        return None

    def monitor_job_power_usage(self, job_id: str, nodes: List[str],
                                duration_minutes: int = 60) -> Dict:
        """Monitor power usage during job execution"""

        self.logger.info(f"Starting power monitoring for job {job_id} (
↪{duration_minutes} minutes)")

        gpu_list = []
        for node in nodes:
            for gpu_id in range(4):  # DGX_GB200 has 4 GPUs
```

(continues on next page)

(continued from previous page)

```
                gpu_list.append([node, gpu_id])

        monitoring_data = {
            'job_id': job_id,
            'start_time': time.time(),
            'samples': [],
            'duration_minutes': duration_minutes
        }

        end_time = time.time() + (duration_minutes * 60)
        sample_interval = 30  # 30 second intervals

        while time.time() < end_time:
            try:
                metrics_response = self.dps_api.metrics.request_
↪metrics(requested_gpus=gpu_list)

                if 'metrics' in metrics_response:
                    total_power = sum(metric['usage'] for metric in metrics_
↪response['metrics'])

                    sample = {
                        'timestamp': time.time(),
                        'total_power': total_power,
                        'node_metrics': {}
                    }

                    # Organize by node
                    for metric in metrics_response['metrics']:
                        node = metric['node']
                        if node not in sample['node_metrics']:
                            sample['node_metrics'][node] = []
                        sample['node_metrics'][node].append(metric['usage'])

                    monitoring_data['samples'].append(sample)

                    self.logger.info(f"Power sample: {total_power:.1f}W total
↪")

                time.sleep(sample_interval)

            except Exception as e:
                self.logger.error(f"Error during power monitoring: {e}")
                time.sleep(sample_interval)

        # Calculate summary statistics
        if monitoring_data['samples']:
            power_values = [s['total_power'] for s in monitoring_data['samples
↪']]
            monitoring_data['summary'] = {
                'average_power': sum(power_values) / len(power_values),
                'min_power': min(power_values),
```

(continues on next page)

(continued from previous page)

```
                'max_power': max(power_values),
                'sample_count': len(power_values)
            }

            self.logger.info(f"Monitoring complete - Avg: {monitoring_data[
↪'summary']['average_power']:.1f}W, "
                             f"Min: {monitoring_data['summary']['min_power']:.
↪1f}W, "
                             f"Max: {monitoring_data['summary']['max_power']:.
↪1f}W")

        return monitoring_data

    def cleanup_with_reporting(self, job_id: str) -> Dict:
        """Cleanup resource group with final power reporting"""

        group_name = f"slurm-job-{job_id}"
        cleanup_report = {
            'job_id': job_id,
            'cleanup_time': time.time(),
            'success': False,
            'final_metrics': None
        }

        try:
            # Get final power metrics before cleanup
            resource_groups = self.dps_api.resource_groups.list_resource_
↪groups()
            for rg in resource_groups.get('resource_groups', []):
                if rg.get('group_name') == group_name:
                    nodes = rg.get('resource_names', [])
                    if nodes:
                        cleanup_report['final_metrics'] = self._get_initial_
↪power_metrics(nodes)
                    break

            # Delete resource group
            cmd = [self.dpsctl_path, "resource-group", "delete", f"--resource-
↪group={group_name}"]
            result = subprocess.run(cmd, capture_output=True, text=True)

            if result.returncode == 0:
                cleanup_report['success'] = True
                self.logger.info(f"Resource group {group_name} cleaned up
↪successfully")
            else:
                self.logger.warning(f"Resource group cleanup had issues:
↪{result.stderr}")
                cleanup_report['error'] = result.stderr

        except Exception as e:
            self.logger.error(f"Error during cleanup with reporting: {e}")
```

(continues on next page)

(continued from previous page)

```
            cleanup_report['error'] = str(e)

        return cleanup_report
```

3.3.2.2.4.149 Step 2: Transform Simulated Node List into Scheduler Resource Pool

Transform the simulated node list in the DPS simulator into your scheduler resource pool.

3.3.2.2.4.150 Node Transformation (Python)

```
#!/usr/bin/env python3
"""
Transform DPS Simulator Nodes into Scheduler Resource Pool
"""
import subprocess
import json
from typing import List, Dict

class DPSSimulatorNodeTransformer:
    def __init__(self):
        # DPS simulator GB200 nodes
        self.simulator_nodes = []
        for rack in range(1, 9):  # Racks 01-08
            for node in range(1, 13):  # Nodes 0001-0012
                self.simulator_nodes.append(f"gb200-r{rack:02d}-{node:04d}")

        # Node characteristics
        self.node_specs = {
            "DGX_GB200": {
                "gpus": 4,          # GB200 has 4 GPUs per node
                "cpus": 2,          # Grace CPUs
                "min_power": 900,   # Watts (from pkg/device/specs/devices.
↪yaml)
                "max_power": 5640,  # Watts
                "gpu_model": "GB200",
                "cpu_model": "Grace"
            }
        }

    def get_dps_simulator_topology(self) -> Dict:
        """Retrieve current topology from DPS simulator"""
        try:
            result = subprocess.run(
                ["dpsctl", "topology", "list-entities", "dps-simulator"],
                capture_output=True, text=True
            )

            if result.returncode == 0:
                # Parse dpsctl output to get node information
                lines = result.stdout.strip().split('\n')
                nodes = []
```

(continues on next page)

(continued from previous page)

```
                for line in lines:
                    if 'ComputerSystem' in line and any(node in line for node
↪in self.simulator_nodes):
                        # Extract node name from dpsctl output
                        for node in self.simulator_nodes:
                            if node in line:
                                nodes.append({
                                    "name": node,
                                    "type": "ComputerSystem",
                                    "model": "DGX_GB200",
                                    "status": "available",
                                    "power_policy": self._extract_policy_from_
↪line(line)
                                })
                                break

                return {"nodes": nodes, "total_count": len(nodes)}
            else:
                print(f"Failed to get topology: {result.stderr}")
                return {"nodes": [], "total_count": 0}

        except Exception as e:
            print(f"Error retrieving DPS topology: {e}")
            return {"nodes": [], "total_count": 0}

    def _extract_policy_from_line(self, line: str) -> str:
        """Extract policy name from dpsctl output line"""
        # Extract policy names
        if "GB200-High" in line:
            return "GB200-High"
        elif "GB200-Med" in line:
            return "GB200-Med"
        elif "GB200-Low" in line:
            return "GB200-Low"
        else:
            return "Default"

    def transform_to_slurm_nodes_conf(self) -> str:
        """Transform DPS simulator nodes into Slurm nodes.conf format"""

        topology = self.get_dps_simulator_topology()
        slurm_config = []

        for node in topology["nodes"]:
            node_name = node["name"]
            model = node.get("model", "DGX_GB200")
            spec = self.node_specs.get(model, self.node_specs["DGX_GB200"])

            # Generate Slurm node configuration for GB200
            slurm_node_config = (
                f"NodeName={node_name} "
```

(continues on next page)

(continued from previous page)

```
                f"CPUs=144 "  # Grace CPU configuration for GB200
                f"Gres=gpu:gb200:{spec['gpus']} "  # 4 GB200 GPUs per node
                f"State=IDLE "
                f"Features=dps_managed,power_aware,gb200 "
                f"# DPS Power: {spec['min_power']}-{spec['max_power']}W"
            )
            slurm_config.append(slurm_node_config)

        # Add partition configuration
        node_list = ",".join([node["name"] for node in topology["nodes"]])
        partition_config = f"PartitionName=dps_simulator Nodes={node_list}
↪Default=YES MaxTime=INFINITE State=UP"

        full_config = "\n".join(slurm_config + ["", partition_config])

        print("Generated Slurm nodes.conf configuration:")
        print(full_config)

        return full_config

    def create_scheduler_resource_pool(self) -> Dict:
        """Create a generic scheduler resource pool from DPS simulator nodes"""

        topology = self.get_dps_simulator_topology()

        resource_pool = {
            "pool_name": "dps_simulator_pool",
            "total_nodes": topology["total_count"],
            "available_nodes": topology["total_count"],
            "resources": []
        }

        for node in topology["nodes"]:
            spec = self.node_specs.get(node.get("model", "DGX_GB200"), self.
↪node_specs["DGX_GB200"])

            resource = {
                "node_id": node["name"],
                "node_type": "gpu_compute",
                "capabilities": {
                    "gpu_count": spec["gpus"],        # 4 GB200 GPUs
                    "cpu_count": spec["cpus"],        # 2 Grace CPUs
                    "gpu_model": spec["gpu_model"],   # GB200
                    "cpu_model": spec["cpu_model"],   # Grace
                    "power_management": True,
                    "power_range": f"{spec['min_power']}-{spec['max_power']}W
↪",
                    "dps_managed": True,
                    "architecture": "grace_blackwell"
                },
                "current_policy": node.get("power_policy", "Default"),
                "status": "available",
```

(continues on next page)

(continued from previous page)

```
                "scheduler_metadata": {
                    "partition": "dps_simulator",
                    "priority": "normal",
                    "exclusive": False,
                    "rack_id": node["name"][:10]  # Extract rack ID (e.g.,
↪"gb200-r01")
                }
            }
            resource_pool["resources"].append(resource)

        print(f"Created scheduler resource pool with {len(resource_pool[
↪'resources'])} nodes")
        return resource_pool

## Usage example
def transform_simulator_nodes_example():
    """Example of transforming DPS simulator nodes"""

    transformer = DPSSimulatorNodeTransformer()

    print("=== Step 2: Transform Simulated Node List ===")

    # Get current DPS simulator topology
    print("1. Retrieving DPS simulator topology...")
    topology = transformer.get_dps_simulator_topology()
    print(f"   Found {topology['total_count']} nodes in simulator")

    # Transform to Slurm format
    print("\n2. Transforming to Slurm configuration...")
    slurm_config = transformer.transform_to_slurm_nodes_conf()

    # Create generic resource pool
    print("\n3. Creating scheduler resource pool...")
    resource_pool = transformer.create_scheduler_resource_pool()

    return resource_pool

if __name__ == "__main__":
    transform_simulator_nodes_example()
```

3.3.2.2.4.151 Step 3: Enable Integration and Verify Operation

Deploy and test the integration components.

3.3.2.2.4.152 Integration Deployment and Testing

```
#!/bin/bash
## enable_integration.sh
## Enable scheduler integration with DPS and test functionality

set -euo pipefail
```

(continues on next page)

(continued from previous page)

```
echo "=== Step 3: Enable Integration and Observe Functionality ==="

## 1. Deploy integration scripts
echo "1. Deploying integration scripts..."
sudo cp slurm_dps_prolog.sh /usr/local/bin/
sudo cp slurm_dps_epilog.sh /usr/local/bin/
sudo chmod +x /usr/local/bin/slurm_dps_*.sh

## 2. Update Slurm configuration
echo "2. Updating Slurm configuration..."
sudo tee -a /etc/slurm/slurm.conf << 'EOF'

## DPS Integration Configuration
Prolog=/usr/local/bin/slurm_dps_prolog.sh
Epilog=/usr/local/bin/slurm_dps_epilog.sh
PrologFlags=Alloc

## DPS Simulator Partition
PartitionName=dps_simulator Nodes=gb200-r01-[0001-0012],gb200-r02-[0001-0012]
 ↪Default=YES MaxTime=INFINITE State=UP
EOF

## 3. Restart Slurm services
echo "3. Restarting Slurm services..."
sudo systemctl restart slurmctld
sudo systemctl restart slurmd

## 4. Verify integration functionality
echo "4. Testing integration functionality..."

## Test job submission
echo "   Submitting test job..."
TEST_JOB_ID=$(sbatch --parsable --job-name=dps-integration-test --
 ↪partition=dps_simulator --nodes=2 --time=5:00 << 'EOF'
#!/bin/bash
echo "DPS Integration Test Job"
echo "Job ID: $SLURM_JOB_ID"
echo "Nodes: $SLURM_JOB_NODELIST"
sleep 60
echo "Job completed"
EOF
)

echo "   Test job submitted with ID: $TEST_JOB_ID"

## Monitor job and DPS integration
echo "   Monitoring job execution..."
sleep 10

        # Check if resource group was created
        echo "   Checking DPS resource group creation..."
```

(continues on next page)

(continued from previous page)

```
        if dpsctl resource-group list | grep -q "slurm-job-$TEST_JOB_ID"; then
            echo "   [SUCCESS] Resource group created successfully"
        else
            echo "   [FAILED] Resource group creation failed"
        fi

## Wait for job completion
echo "   Waiting for job completion..."
while squeue -j $TEST_JOB_ID | grep -q $TEST_JOB_ID; do
    sleep 5
done

        # Check if resource group was cleaned up
        echo "   Checking DPS resource group cleanup..."
        sleep 5
        if dpsctl resource-group list | grep -q "slurm-job-$TEST_JOB_ID"; then
            echo "   [FAILED] Resource group cleanup failed"
        else
            echo "   [SUCCESS] Resource group cleaned up successfully"
        fi

echo "=== Integration functionality test complete ==="
```

3.3.2.2.4.153 Step 4: Observe Job Scheduling Failures

Observe DPS simulator dashboards for indications of job scheduling failures.

3.3.2.2.4.154 Dashboard Monitoring (Python)

```
#!/usr/bin/env python3
"""
Monitor DPS Simulator Dashboards for Job Scheduling Failures
"""
import time
import json
import subprocess
from typing import Dict, List
from dpsapi.api import DpsApi

class JobSchedulingFailureMonitor:
    def __init__(self):
        self.dps_api = DpsApi(
            dps_host="api.dps",
            dps_port=443,
            dps_username="monitoring",
            dps_password="monitor-password",
            dps_insecure_tls_skip_verify=True,
        )

        self.failure_indicators = []
```

(continues on next page)

(continued from previous page)

```
    def monitor_scheduling_failures(self, duration_minutes: int = 30):
        """Monitor DPS dashboards for job scheduling failure indicators"""

        print(f"=== Step 4: Monitor Job Scheduling Failures ({duration_
↪minutes} minutes) ===")

        end_time = time.time() + (duration_minutes * 60)
        failure_count = 0

        while time.time() < end_time:
            try:
                # Check for common scheduling failure indicators
                failures = self._check_failure_indicators()

                if failures:
                    failure_count += len(failures)
                    print(f"[WARNING] Detected {len(failures)} scheduling
↪issues:")
                    for failure in failures:
                        print(f"   - {failure['type']}: {failure['message']}")
                        print(f"     Time: {time.strftime('%H:%M:%S', time.
↪localtime(failure['timestamp']))}")
                        print(f"     Affected: {failure.get('affected_
↪resources', 'Unknown')}")

                time.sleep(30)  # Check every 30 seconds

            except Exception as e:
                print(f"Error monitoring failures: {e}")
                time.sleep(30)

        print(f"\n=== Monitoring Summary ===")
        print(f"Total scheduling issues detected: {failure_count}")
        print(f"Monitoring duration: {duration_minutes} minutes")

        if failure_count == 0:
            print("[SUCCESS] No scheduling failures detected - Integration
↪working correctly")
        else:
            print("[WARNING] Scheduling issues detected - Review failure
↪details above")

        return self.failure_indicators

    def _check_failure_indicators(self) -> List[Dict]:
        """Check various indicators of job scheduling failures"""

        current_failures = []
        timestamp = time.time()

        # 1. Check for resource groups in error state
        try:
```

(continues on next page)

(continued from previous page)

```
            resource_groups = self.dps_api.resource_groups.list_resource_
↪groups()
            for rg in resource_groups.get('resource_groups', []):
                if not rg.get('is_active', False) and rg.get('group_name', '
↪').startswith('slurm-job-'):
                    current_failures.append({
                        'type': 'resource_group_inactive',
                        'message': f"Resource group {rg['group_name']} is not
↪active",
                        'timestamp': timestamp,
                        'affected_resources': rg.get('resource_names', [])
                    })
        except Exception as e:
            current_failures.append({
                'type': 'api_error',
                'message': f"Failed to query resource groups: {e}",
                'timestamp': timestamp,
                'affected_resources': 'API'
            })

        # 2. Check for nodes with failed policy application
        try:
            nodes = ["gb200-r01-0001", "gb200-r01-0002", "gb200-r01-0003",
↪"gb200-r01-0004"]  # Sample nodes
            gpu_list = [[node, gpu_id] for node in nodes for gpu_id in
↪range(4)]  # DGX_GB200 has 4 GPUs

            metrics_response = self.dps_api.metrics.request_metrics(requested_
↪gpus=gpu_list)

            if 'metrics' not in metrics_response:
                current_failures.append({
                    'type': 'metrics_unavailable',
                    'message': "GPU metrics not available from DPS",
                    'timestamp': timestamp,
                    'affected_resources': nodes
                })
            else:
                # Check for nodes with zero power (indicating potential
↪failure)
                node_powers = {}
                for metric in metrics_response['metrics']:
                    node = metric['node']
                    if node not in node_powers:
                        node_powers[node] = 0
                    node_powers[node] += metric['usage']

                for node, power in node_powers.items():
                    if power < 100:  # Low power might indicate issues
                        current_failures.append({
                            'type': 'low_power_node',
                            'message': f"Node {node} showing unexpectedly low
```

(continues on next page)

(continued from previous page)

```
↪power: {power:.1f}W",
                            'timestamp': timestamp,
                            'affected_resources': [node]
                        })

        except Exception as e:
            current_failures.append({
                'type': 'power_monitoring_error',
                'message': f"Failed to monitor node power: {e}",
                'timestamp': timestamp,
                'affected_resources': 'Power monitoring'
            })

        # 3. Check Slurm job queue for stuck jobs
        try:
            result = subprocess.run(
                ["squeue", "--format=%i,%j,%t,%r", "--noheader"],
                capture_output=True, text=True, timeout=10
            )

            if result.returncode == 0:
                for line in result.stdout.strip().split('\n'):
                    if line:
                        job_id, job_name, state, reason = line.split(',')

                        # Look for problematic job states
                        if state in ['PD'] and 'Power' in reason:
                            current_failures.append({
                                'type': 'job_power_pending',
                                'message': f"Job {job_id} pending due to
↪power constraint: {reason}",
                                'timestamp': timestamp,
                                'affected_resources': [job_id]
                            })
                        elif state in ['F', 'NF', 'TO']:
                            current_failures.append({
                                'type': 'job_failed',
                                'message': f"Job {job_id} ({job_name}) failed
↪with state {state}",
                                'timestamp': timestamp,
                                'affected_resources': [job_id]
                            })
        except Exception as e:
            # Slurm might not be available in all environments
            pass

        # Store failures for reporting
        self.failure_indicators.extend(current_failures)

        return current_failures

## Usage example
```

(continues on next page)

(continued from previous page)

```
def monitor_job_failures_example():
    """Example monitoring for job scheduling failures"""

    monitor = JobSchedulingFailureMonitor()
    failures = monitor.monitor_scheduling_failures(duration_minutes=10)

    return failures

if __name__ == "__main__":
    monitor_job_failures_example()
```

3.3.2.2.4.155 Step 5: Observe Power Utilization Compliance

Observe DPS simulator dashboards for power utilization and successful compliance integration.

3.3.2.2.4.156 Power Utilization Monitoring (Python)

```
#!/usr/bin/env python3
"""
Monitor DPS Simulator Dashboards for Power Utilization and Compliance
"""
import time
import numpy as np
from typing import Dict, List, Tuple
from dpsapi.api import DpsApi

class PowerUtilizationComplianceMonitor:
    def __init__(self):
        self.dps_api = DpsApi(
            dps_host="api.dps",
            dps_port=443,
            dps_username="compliance-monitor",
            dps_password="monitor-password",
            dps_insecure_tls_skip_verify=True,
        )

        # Compliance thresholds
        self.compliance_config = {
            'power_variance_threshold': 0.15,  # 15% variance allowed
            'policy_compliance_threshold': 0.95,  # 95% compliance required
            'monitoring_interval': 30  # seconds
        }

        self.compliance_data = []

    def monitor_power_utilization_compliance(self, duration_minutes: int =
↪30):
        """Monitor power utilization and compliance through DPS dashboards"""

        print(f"=== Step 5: Monitor Power Utilization Compliance ({duration_
↪minutes} minutes) ===")
```

(continues on next page)

(continued from previous page)

```
        end_time = time.time() + (duration_minutes * 60)
        sample_count = 0
        compliance_violations = []

        while time.time() < end_time:
            try:
                # Collect utilization and compliance data
                utilization_data = self._collect_utilization_data()
                compliance_status = self._assess_compliance(utilization_data)

                sample_count += 1
                self.compliance_data.append({
                    'timestamp': time.time(),
                    'utilization': utilization_data,
                    'compliance': compliance_status
                })

                # Report current status
                print(f"\nSample {sample_count} - {time.strftime('%H:%M:%S')}
↪")
                self._report_current_status(utilization_data, compliance_
↪status)

                # Check for violations
                violations = self._identify_violations(compliance_status)
                if violations:
                    compliance_violations.extend(violations)
                    print("[WARNING] Compliance violations detected:")
                    for violation in violations:
                        print(f"   - {violation['type']}: {violation['message
↪']}")

                time.sleep(self.compliance_config['monitoring_interval'])

            except Exception as e:
                print(f"Error monitoring compliance: {e}")
                time.sleep(self.compliance_config['monitoring_interval'])

        # Generate compliance summary
        self._generate_compliance_summary(sample_count, compliance_violations)

        return self.compliance_data

    def _collect_utilization_data(self) -> Dict:
        """Collect power utilization data"""

        # Get sample nodes
        nodes = ["gb200-r01-0001", "gb200-r01-0002", "gb200-r01-0003", "gb200-
↪r01-0004",
                 "gb200-r02-0001", "gb200-r02-0002"]  # Sample nodes
```

(continues on next page)

(continued from previous page)

```
        # Collect GPU metrics
        gpu_list = [[node, gpu_id] for node in nodes for gpu_id in range(4)]
↪# DGX_GB200 has 4 GPUs
        metrics_response = self.dps_api.metrics.request_metrics(requested_
↪gpus=gpu_list)

        if 'metrics' not in metrics_response:
            raise Exception("No power metrics available")

        # Process metrics
        node_power = {}
        total_power = 0

        for metric in metrics_response['metrics']:
            node = metric['node']
            power = metric['usage']

            if node not in node_power:
                node_power[node] = {'gpus': [], 'total': 0}

            node_power[node]['gpus'].append(power)
            node_power[node]['total'] += power
            total_power += power

        # Calculate utilization statistics
        node_powers = [data['total'] for data in node_power.values()]
        gpu_powers = [metric['usage'] for metric in metrics_response['metrics
↪']]

        utilization_data = {
            'total_power': total_power,
            'node_count': len(nodes),
            'gpu_count': len(gpu_list),
            'node_power_distribution': node_power,
            'statistics': {
                'average_node_power': np.mean(node_powers) if node_powers
↪else 0,
                'node_power_std': np.std(node_powers) if len(node_powers) > 1
↪else 0,
                'average_gpu_power': np.mean(gpu_powers) if gpu_powers else 0,
                'gpu_power_std': np.std(gpu_powers) if len(gpu_powers) > 1
↪else 0,
                'power_distribution_variance': np.var(node_powers) if
↪len(node_powers) > 1 else 0
            }
        }

        return utilization_data

    def _assess_compliance(self, utilization_data: Dict) -> Dict:
        """Assess power utilization compliance against policies"""
```

(continues on next page)

(continued from previous page)

```
        compliance_status = {
            'overall_compliant': True,
            'compliance_score': 1.0,
            'assessments': {}
        }

        # 1. Power Distribution Compliance
        stats = utilization_data['statistics']
        avg_node_power = stats['average_node_power']
        node_power_std = stats['node_power_std']

        if avg_node_power > 0:
            power_variance = node_power_std / avg_node_power
            power_distribution_compliant = power_variance <= self.compliance_
↪config['power_variance_threshold']
        else:
            power_variance = 0
            power_distribution_compliant = True

        compliance_status['assessments']['power_distribution'] = {
            'compliant': power_distribution_compliant,
            'variance': power_variance,
            'threshold': self.compliance_config['power_variance_threshold'],
            'score': max(0, 1.0 - (power_variance / self.compliance_config[
↪'power_variance_threshold']))
        }

        # 2. Policy Limit Compliance
        policy_violations = 0
        total_gpus = 0

        for node, data in utilization_data['node_power_distribution'].items():
            for gpu_power in data['gpus']:
                total_gpus += 1
                # Site-configured example threshold: 1,250 W per GPU
                if gpu_power > 1250:
                    policy_violations += 1

        policy_compliance_rate = (total_gpus - policy_violations) / total_
↪gpus if total_gpus > 0 else 1.0
        policy_compliant = policy_compliance_rate >= self.compliance_config[
↪'policy_compliance_threshold']

        compliance_status['assessments']['policy_limits'] = {
            'compliant': policy_compliant,
            'compliance_rate': policy_compliance_rate,
            'violations': policy_violations,
            'total_gpus': total_gpus,
            'threshold': self.compliance_config['policy_compliance_threshold']
        }
```

(continues on next page)

(continued from previous page)

```
        # Calculate overall compliance
        assessment_scores = [assessment['score'] for assessment in compliance_
↪status['assessments'].values()
                             if 'score' in assessment]
        compliance_status['compliance_score'] = np.mean(assessment_scores) if
↪assessment_scores else 0

        compliance_status['overall_compliant'] = all(
            assessment['compliant'] for assessment in compliance_status[
↪'assessments'].values()
        )

        return compliance_status

    def _report_current_status(self, utilization_data: Dict, compliance_
↪status: Dict):
        """Report current power utilization and compliance status"""

        total_power = utilization_data['total_power']
        avg_node_power = utilization_data['statistics']['average_node_power']
        compliance_score = compliance_status['compliance_score']

        status_icon = "[OK]" if compliance_status['overall_compliant'] else
↪"[WARN]"

        print(f"   {status_icon} Total Power: {total_power:.1f}W | Avg Node:
↪{avg_node_power:.1f}W | Compliance: {compliance_score:.2f}")

        # Report assessment details
        for assessment_name, assessment in compliance_status['assessments'].
↪items():
            icon = "[OK]" if assessment['compliant'] else "[FAIL]"
            if assessment_name == 'power_distribution':
                print(f"   {icon} Power Distribution: {assessment['variance
↪']:.3f} variance (limit: {assessment['threshold']:.3f})")
            elif assessment_name == 'policy_limits':
                print(f"   {icon} Policy Compliance: {assessment['compliance_
↪rate']:.2f} rate ({assessment['violations']} violations)")

    def _identify_violations(self, compliance_status: Dict) -> List[Dict]:
        """Identify specific compliance violations"""

        violations = []

        for assessment_name, assessment in compliance_status['assessments'].
↪items():
            if not assessment['compliant']:
                if assessment_name == 'power_distribution':
                    violations.append({
                        'type': 'power_distribution_violation',
                        'message': f"Power distribution variance {assessment[
↪'variance']:.3f} exceeds threshold {assessment['threshold']:.3f}"
```

(continues on next page)

(continued from previous page)

```
                    })
                elif assessment_name == 'policy_limits':
                    violations.append({
                        'type': 'policy_limit_violation',
                        'message': f"{assessment['violations']} GPUs
↪exceeding policy limits ({assessment['compliance_rate']:.2f} compliance
↪rate)"
                    })

        return violations

    def _generate_compliance_summary(self, sample_count: int, violations:
↪List[Dict]):
        """Generate compliance monitoring summary"""

        print(f"\n=== Power Utilization Compliance Summary ===")
        print(f"Monitoring Duration: {len(self.compliance_data)} samples over
↪{sample_count} intervals")

        if self.compliance_data:
            # Calculate summary statistics
            compliance_scores = [data['compliance']['compliance_score'] for
↪data in self.compliance_data]
            avg_compliance = np.mean(compliance_scores)
            min_compliance = np.min(compliance_scores)

            total_powers = [data['utilization']['total_power'] for data in
↪self.compliance_data]
            avg_power = np.mean(total_powers)
            power_trend = "increasing" if total_powers[-1] > total_powers[0]
↪else "decreasing"

            print(f"Average Compliance Score: {avg_compliance:.3f}")
            print(f"Minimum Compliance Score: {min_compliance:.3f}")
            print(f"Average Total Power: {avg_power:.1f}W")
            print(f"Power Trend: {power_trend}")

            # Violation summary
            violation_types = {}
            for violation in violations:
                violation_type = violation['type']
                violation_types[violation_type] = violation_types.
↪get(violation_type, 0) + 1

            if violations:
                print(f"\nCompliance Violations ({len(violations)} total):")
                for violation_type, count in violation_types.items():
                    print(f"  - {violation_type}: {count} occurrences")
            else:
                print("\n[SUCCESS] No compliance violations detected")

            # Final assessment
```

(continues on next page)

(continued from previous page)

```
            if avg_compliance >= 0.95 and not violations:
                print(f"\n[SUCCESS] Scheduling integration compliance score:
↪{avg_compliance:.2f}")
            elif avg_compliance >= 0.85:
                print(f"\n[OK] Scheduling integration compliance score: {avg_
↪compliance:.2f}")
            elif avg_compliance >= 0.70:
                print(f"\n[WARNING] Scheduling integration compliance score:
↪{avg_compliance:.2f} - some issues detected")
            else:
                print(f"\n[FAILED] Scheduling integration compliance score:
↪{avg_compliance:.2f} - issues detected")

## Usage example
def monitor_compliance_example():
    """Example monitoring for power utilization compliance"""

    monitor = PowerUtilizationComplianceMonitor()
    compliance_data = monitor.monitor_power_utilization_compliance(duration_
↪minutes=15)

    return compliance_data

## Example usage functions
def prolog_integration():
    """Prolog integration with job analysis"""

    # Get Slurm job information
    job_info = {
        'job_id': os.environ.get('SLURM_JOB_ID'),
        'name': os.environ.get('SLURM_JOB_NAME', ''),
        'partition': os.environ.get('SLURM_JOB_PARTITION', ''),
        'qos': os.environ.get('SLURM_JOB_QOS', 'normal'),
        'time_limit': os.environ.get('SLURM_TIME_LIMIT', '01:00:00'),
        'nodes': os.environ.get('SLURM_JOB_NODELIST', '').split(',')
    }

    # Initialize integrator
    integrator = SlurmDPSIntegrator()

    # Analyze job requirements
    analysis = integrator.analyze_job_requirements(job_info)

    # Create resource group
    success = integrator.create_resource_group_for_job(analysis)

    if success:
        print(f"DPS integration successful for job {job_info['job_id']}")
        return 0
    else:
        print(f"DPS integration failed for job {job_info['job_id']}")
        return 1
```

(continues on next page)

(continued from previous page)

```
def epilog_integration():
    """Epilog integration with reporting"""

    job_id = os.environ.get('SLURM_JOB_ID')

    # Initialize integrator
    integrator = SlurmDPSIntegrator()

    # Cleanup with reporting
    report = integrator.cleanup_with_reporting(job_id)

    if report['success']:
        print(f"DPS cleanup successful for job {job_id}")
        if report['final_metrics']:
            print(f"Final power consumption: {report['final_metrics']['total_
↪power']:.1f}W")
    else:
        print(f"DPS cleanup completed with issues for job {job_id}")

    return 0  # Do not fail jobs due to cleanup issues

if __name__ == "__main__":
    import sys

    logging.basicConfig(level=logging.INFO)

    if len(sys.argv) > 1:
        if sys.argv[1] == "prolog":
            sys.exit(prolog_integration())
        elif sys.argv[1] == "epilog":
            sys.exit(epilog_integration())

    print("Usage: python3 slurm_dps_integration.py [prolog|epilog]")
    sys.exit(1)
```

3.3.2.2.4.157 Configuration and Setup

3.3.2.2.4.158 Slurm Configuration

Add the following to `/etc/slurm/slurm.conf`:

```
## DPS Integration Configuration
Prolog=/usr/local/bin/slurm_dps_prolog.sh
Epilog=/usr/local/bin/slurm_dps_epilog.sh

## Enable prolog/epilog for all partitions
PrologFlags=Alloc
```

3.3.2.2.4.159 Environment Setup Commands

```
## Deploy DPS SDK
task deploy:sdk

## Install integration scripts
sudo cp slurm_dps_prolog.sh /usr/local/bin/
sudo cp slurm_dps_epilog.sh /usr/local/bin/
sudo chmod +x /usr/local/bin/slurm_dps_*.sh

## Login to DPS
printf '%s\n' '<password>' | dpsctl --host api.dps --port 443 --insecure-tls-
↪skip-verify login --username <username> --password-stdin

## Import your topology
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify topology import
↪<topology-file.json>
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify topology activate
↪--topology <topology-name>

## Restart Slurm to pick up new configuration
sudo systemctl restart slurmctld
sudo systemctl restart slurmd
```

3.3.2.2.4.160 Testing Commands

```
## Submit test job to verify integration
sbatch --job-name=test-job --partition=dps_simulator --nodes=2 test_job.sh

## Monitor resource groups
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify resource-group
↪list
watch 'dpsctl --host api.dps --port 443 --insecure-tls-skip-verify resource-
↪group list'

## Check power consumption
dpsctl --host api.dps --port 443 --insecure-tls-skip-verify check metrics --
↪nodes=<node-name>

## View integration logs
tail -f /var/log/slurm/dps_prolog.log
tail -f /var/log/slurm/dps_epilog.log
```

3.3.2.2.4.161 Observable Metrics

3.3.2.2.4.162 Scheduler Integration Performance

Monitor these metrics during scheduler integration:

- **Resource Group Creation Time**: Time from job submission to DPS resource group activation
- **Policy Application Success**: Success rate for resource group creation and activation

- **Job Startup Overhead**: Additional time due to DPS integration
- **Cleanup Reliability**: Success rate for resource group cleanup in epilog

3.3.2.2.4.163 Power Management

- **Policy Compliance**: Applied power policies reflected in actual power consumption
- **Power Monitoring**: Real-time power consumption metrics
- **Policy Selection**: Power policies selected based on job characteristics
- **Resource Utilization**: Datacenter power capacity usage across scheduled workloads

3.3.2.2.4.164 Integration Health

- **Script Execution**: Prolog/epilog script execution success
- **API Reliability**: API call success during operations
- **Monitoring Coverage**: Power monitoring data availability for scheduled jobs
- **Error Recovery**: Handling of DPS service unavailability

This scheduling workload integration provides a solution for workload schedulers to manage datacenter power policies using the DPS simulator environment.

3.3.3. Developer Guides

Documentation and guides for DPS SDK developers.

3.3.3.1 Developer Guide

This guide provides a high-level overview of developing with and integrating against DPS using the DPS SDK. Whether you are building power optimization systems or grid integration solutions, this guide helps you get started.

3.3.3.1.1 Quick Start

3.3.3.1.1.1 Set Up Your Environment

Download the DPS SDK files from the NVIDIA NVOnline Portal and unarchive them.

Install Task (minimum version **v3.48.0**, required for `defer:` support):

```
# Linux
sh -c "$(curl --location https://taskfile.dev/install.sh)" -- -d -b /usr/
↪local/bin v3.48.0

# macOS
brew install go-task/tap/go-task
```

Install SDK dependencies (docker, kubectl, k3d, helm, helm-git, uv, `dpsctl`):

```
task setup
```

3.3.3.1.1.2 Deploy the SDK

The recommended starting point is the **Default profile** - a blank DPS environment with no topology or simulators loaded, ideal for hands-on `dpsctl` exercises and API exploration:

```
task deploy
```

Configure your shell once so `dpsctl` connects to the local SDK endpoint without repeating connection flags:

```
export DPSCTL_HOST=api.dps.sdk
export DPSCTL_PORT=80
export DPSCTL_INSECURE_TLS_SKIP_VERIFY=true
```

For automated resource-group, grid, and load-shedding simulations against the full 144-node GB300 topology, use the Hardware Emulation profile instead:

```
task sdk    # creates the k3d cluster and deploys DPS with the GB300 topology
task sim    # imports the topology and runs combined RG + grid simulation
```

To deploy with a custom Helm values file:

```
task deploy:custom VALUES_FILE=path/to/my-values.yaml
```

Verify the SDK is running properly:

```
kubectl get pods -n dps
```

Your output should look similar to the following:

```
alertmanager-dps-sdk-kube-prometheus-st-alertmanager-0   2/2     Running   0
 ↪        15s
bmc-gb300-simulator-b6b88c546-687n2                      1/1     Running   0
 ↪        15s
dps-sdk-alloy-0                                          2/2     Running   0
 ↪        15s
dps-sdk-grafana-d676f6f8f-hnkhd                          3/3     Running   0
 ↪        15s
dps-sdk-kube-prometheus-st-operator-6fd968d45c-hj4pl     1/1     Running   0
 ↪        15s
dps-sdk-kube-state-metrics-55d94c9858-b2hh5              1/1     Running   0
 ↪        15s
dps-sdk-loki-0                                           2/2     Running   0
 ↪        15s
dps-sdk-loki-chunks-cache-0                              2/2     Running   0
 ↪        15s
dps-sdk-loki-gateway-85c47bddc-972t7                     1/1     Running   0
 ↪        15s
dps-sdk-loki-results-cache-0                             2/2     Running   0
 ↪        15s
dps-sdk-postgresql-0                                     1/1     Running   0
 ↪        15s
```

(continues on next page)

(continued from previous page)

```
dps-sdk-prometheus-node-exporter-p4gtt                   1/1     Running   0
↪          15s
dps-sdk-promtail-dtbwc                                   1/1     Running   0
↪          15s
dps-sdk-pyroscope-0                                      1/1     Running   0
↪          15s
dps-sdk-tempo-0                                          1/1     Running   0
↪          15s
dps-server-0                                             1/1     Running   0
↪          15s
dps-ui-69cb67b98d-9fsqx                                  1/1     Running   0
↪          15s
loki-canary-6qwwm                                        1/1     Running   0
↪          15s
prometheus-dps-sdk-kube-prometheus-st-prometheus-0       2/2     Running   0
↪          15s
prs-65f6f75b8f-qnmct                                     1/1     Running   0
↪          15s
```

The `dpsctl` CLI provides its own verification command to verify the DPS application is running properly:

```
dpsctl verify
```

A successful deployment should contain output that looks similar to the following:


```
{
  "auth": {
    "details": "JWT secret key is configured",
    "healthy": true,
    "message": "Authentication service is configured with JWT",
    "name": "Authentication Service"
  },
  "bcm": {
    "message": "BCM check skipped - not configured",
    "name": "BCM",
    "skip_reason": "BCM not configured or not using BCM credentials store",
    "skipped": true
  },
  "database": {
    "details": "Database ping successful",
    "healthy": true,
    "message": "Database connectivity successful",
    "name": "Database"
  },
  "dps_server": {
    "details": "gRPC endpoint accessible, topology server responsive",
    "healthy": true,
    "message": "DPS Server is healthy",
    "name": "DPS Server"
  },
  "status": {
```


(continues on next page)

(continued from previous page)

```
    "diag_msg": "All requested DPS components are healthy",
    "ok": true
  },
  "ui": {
    "details": "URL: http://ui.dps.sdk. Note: You may need to update your /
↪etc/hosts file to resolve ui.dps.sdk to the cluster's IP address. The UI
↪should be tested from an external browser.",
    "healthy": true,
    "message": "UI ingress endpoint retrieved successfully",
    "name": "DPS UI"
  }
}
```

3.3.3.1.2 Access Services

Service	URL	Default Auth
DPS API (gRPC)	`api.dps.sdk`	`dps/dps`
DPS Web UI	`http://ui.dps.sdk`	`dps/dps`
Grafana	`http://grafana.dps.sdk`	`admin/dps`
Prometheus	`http://prometheus.dps.sdk`	-
Alertmanager	`http://alertmanager.dps.sdk`	-
Pyroscope	`http://pyroscope.dps.sdk`	-

3.3.3.1.3 Key Concepts for Developers

3.3.3.1.3.1 Topology

The topology represents the physical datacenter power distribution:

- **Entities** - Physical components (nodes, racks, PDUs, PSUs)
- **Relationships** - Power distribution hierarchy
- **Policy bundle selection** - Topology-wide effective power controls

The Hardware Emulation and Workload-Aware profiles include a pre-configured topology with 144 emulated DGX GB300 nodes across 8 racks (18 nodes per rack).

3.3.3.1.3.2 Resource Groups

Resource groups are the primary abstraction for managing power policies for workloads. They follow a lifecycle:

1. **CREATE** - Initialize an empty resource group with an optional policy bundle
2. **ADD** - Add compute nodes to the group
3. **ACTIVATE** - Apply power policies to the hardware

4. **UPDATE** (optional) - Change supported retained settings while the group is inactive; policy-bundle updates are not implemented in released 0.9
5. **DELETE** - Clean up and restore topology defaults

Resource groups are ideal for:

- **Workload orchestrators** - Map jobs to resource groups
- **Power optimization** - Apply custom policies to workload collections
- **Temporary overrides** - Override topology policies for specific workloads

3.3.3.1.3.3 Power Policies

Power policies define power constraints for hardware:

- **Node-level policies** - Overall node power limits
- **GPU-level policies** - Per-GPU power limits (watts or percentage)
- **CPU-level policies** - CPU power constraints
- **Memory-level policies** - Memory power constraints
- **Workload profiles** - Hardware-specific optimizations for GPU workloads

Older integrations can expose the `policy_name`/`PolicyName` compatibility fields, but these fields do not select a policy bundle.

3.3.3.1.3.4 Dynamic Power Management (DPM)

DPS supports dynamic power optimization:

- **Power Reservation Steering (PRS)** - Automatic power redistribution based on telemetry
- **Runtime adjustments** - Update policies during workload execution
- **Telemetry-driven** - React to actual power consumption patterns

3.3.3.1.4 Common Integration Patterns

3.3.3.1.4.1 1. Power Distribution Optimization - Optimize power distribution across topologies

3.3.3.1.4.2 3. Grid Integration - Implement demand response for grid events using the NvGrid API

3.3.3.1.5 Integration Options

DPS provides a gRPC API for integration, with pre-built libraries for Golang and Python.

3.3.3.1.5.1 Python Client

The Python Client provides a Pythonic interface to DPS:

```
from dpsapi import DpsApi

# Connect to DPS
api = DpsApi(
    dps_host="api.dps.sdk",
    dps_port=80,
```

(continues on next page)

(continued from previous page)

```
    dps_username="dps",
    dps_password="dps",
    dps_insecure_tls_skip_verify=True,
)

# Create resource group for workload
api.resource_groups.create_resource_group(
    external_id=12345,
    resource_group_name="job_12345",
    policy_bundle="MaxP",
)

# Add compute nodes
api.resource_groups.add_resources_to_group(
    resource_group_name="job_12345",
    entity_names=["gb300-r01-0001", "gb300-r01-0002"],
)

# Activate with power policies
api.resource_groups.activate_resource_group(
    resource_group_name="job_12345"
)

# Query metrics
metrics = api.metrics.request_metrics([
    ("gb300-r01-0001", 0),  # node, gpu_id
    ("gb300-r01-0001", 1)
])

# Cleanup when done
api.resource_groups.delete_resource_group(
    resource_group_name="job_12345"
)
```

Complete Example: See this *Power Distribution Optimization Demo* for a full power optimization workflow demonstrating:

- Power distribution algorithms across datacenter resources
- Multi-level power monitoring (aggregate, node-level, and GPU-level metrics)
- Resource group lifecycle with policy-bundle selection
- Integration patterns for power optimization systems

3.3.3.1.5.2 Go gRPC Client

For Go applications, use the gRPC client directly:

```
package main

import (
    "context"
    "log"
```

(continues on next page)

(continued from previous page)

```
    "google.golang.org/grpc"
    "nvidia.com/NVIDIA/dcpower/api/v1"
)

func main() {
    // Connect to DPS
    conn, err := grpc.Dial("api.dps.sdk:50051", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("Failed to connect: %v", err)
    }
    defer conn.Close()

    // Create resource group client
    client := api.NewResourceGroupManagementServiceClient(conn)

    // Create resource group
    createReq := &api.ResourceGroupCreateRequest{
        ExternalId:   12345,
        GroupName:    "job_12345",
        PolicyBundle: "MaxP",
    }

    _, err = client.ResourceGroupCreate(context.Background(), createReq)
    if err != nil {
        log.Fatalf("Failed to create resource group: %v", err)
    }

    // Add entities
    addReq := &api.ResourceGroupAddResourcesRequest{
        GroupName:   "job_12345",
        ResourceNames: []string{"gb300-r01-0001", "gb300-r01-0002"},
    }

    _, err = client.ResourceGroupAddResources(context.Background(), addReq)
    if err != nil {
        log.Fatalf("Failed to add entities: %v", err)
    }

    // Activate
    activateReq := &api.ActivateResourceGroupRequest{
        GroupName: "job_12345",
    }

    _, err = client.ActivateResourceGroup(context.Background(), activateReq)
    if err != nil {
        log.Fatalf("Failed to activate: %v", err)
    }

    log.Println("Resource group created and activated successfully")
}
```

Complete Example: See the *Grid Integrator guide* for complete integration examples demonstrating:

- NvGrid API client for grid integration
- Load target scheduling and management
- Webhook event handling
- Power feed metadata queries

Using the API:

The API reference can be found *here*.

3.3.3.1.5.3 gRPC API

For any language, use the gRPC API via `grpcurl`:

```
# Get Token
grpcurl -insecure -d '{
  "passwordCredential": {
    "username": "dps",
    "password": "dps"
  }
}' \
  api.dps.sdk:80 \
  nvidia.dcpower.v1.AuthService/Token

# Use the token to call APIs
export DPS_ACCESS_TOKEN="<token-from-above>"

# List resource groups
grpcurl -insecure -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
  api.dps.sdk:80 \
  nvidia.dcpower.v1.ResourceGroupManagementService/ResourceGroupList


# Create resource group
grpcurl -insecure -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
-d '{
    "external_id": 12345,
    "group_name": "job_12345",
    "policy_bundle": "MaxP"
    }' \
  api.dps.sdk:80 \
  nvidia.dcpower.v1.ResourceGroupManagementService/ResourceGroupCreate

# Add entities
grpcurl -insecure -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
-d '{
    "group_name": "job_12345",
    "resource_names": ["gb300-r01-0001", "gb300-r01-0002"]
    }' \
  api.dps.sdk:80 \
  nvidia.dcpower.v1.ResourceGroupManagementService/ResourceGroupAddResources

# Activate
grpcurl -insecure -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
```

(continues on next page)

(continued from previous page)

```
-d '{"group_name": "job_12345"}' \
  api.dps.sdk:80 \
  nvidia.dcpower.v1.ResourceGroupManagementService/ActivateResourceGroup

# Delete
grpcurl -insecure -H "authorization: Bearer $DPS_ACCESS_TOKEN" \
-d '{"group_name": "job_12345"}' \
  api.dps.sdk:80 \
  nvidia.dcpower.v1.ResourceGroupManagementService/ResourceGroupDelete
```

3.3.3.1.5.4 CLI (dpsctl)

For scripting and manual operations:

```
# Create resource group
dpsctl -H api.dps.sdk -p 80 --insecure-tls-skip-verify rg create \
  --external-id 12345 \
  --resource-group "job_12345" \
  --policy-bundle MaxP

# Add entities
dpsctl -H api.dps.sdk -p 80 --insecure-tls-skip-verify rg add \
  --resource-group "job_12345" \
  --entities "gb300-r01-0001,gb300-r01-0002"

# Activate
dpsctl -H api.dps.sdk -p 80 --insecure-tls-skip-verify rg activate \
  --resource-group "job_12345"

# List
dpsctl -H api.dps.sdk -p 80 --insecure-tls-skip-verify rg list

# Delete
dpsctl -H api.dps.sdk -p 80 --insecure-tls-skip-verify rg delete \
  --resource-group "job_12345"
```

3.3.3.1.6 Available APIs

DPS provides multiple gRPC services:

- **ResourceGroupManagementService** - Resource group lifecycle and management
- **TopologyManagementService** - Topology import, activation, entity generation from BCM or Nautobot (async `GenerateEntities` / `GetGenerateEntitiesStatus`), and queries
- **MetricsManagementService** - Power telemetry and metrics
- **PolicyManagementService** - Power policy management
- **NvGridService** - Grid integration and load management
- **AuthService** - Authentication and authorization
- **VersionService** - Version information

3.3.3.1.7 Development Workflow

3.3.3.1.7.1 Local Development With the DPS SDK

```
# Deploy the default (blank) profile
task deploy

# Or, deploy the Hardware Emulation profile with the full 144-node GB300
↪topology
task sdk
task sim

# Run your integration code
python my_integration.py

# Monitor with Grafana
open http://grafana.dps.sdk  # admin/dps

# View logs
kubectl logs -n dps -l app=dps-server -f

# Cleanup (tear down SDK and delete cluster)
task sdk:down
```

3.3.3.1.7.2 Testing Your Integration

Use the simulator to test integration scenarios:

```
# Run automated resource group simulation
task sim:rgs

# Custom simulation parameters
task sim:rgs MAX_RGS=10 MIN_DURATION=60 MAX_DURATION=180

# Monitor metrics in Grafana while testing
# - DPS Server Dashboard
# - DPS SDK Datacenter dashboard
# - DPS SDK Resource Groups dashboard
```

3.3.3.1.8 Troubleshooting

3.3.3.1.8.1 Common Issues

Connection refused:

```
# Verify DPS is running
kubectl get pods -n dps
kubectl get svc -n dps
dpsctl verify

# Check /etc/hosts entries
grep dps.sdk /etc/hosts
```

Authentication errors:

Use the login procedure for your server's setup in *Configure Server Authentication*. If access still fails, follow *Cannot Connect or Log In* to check the credentials and access requirements.

Resource group activation fails:

```
# View server logs
kubectl logs -n dps -l app=dps-server --tail=100
```

3.3.3.1.9 Additional Resources

- ***Administrator Documentation*** - Deployment and operations
- ***Partner Integration Guides*** - Integration patterns
- ***API Reference*** - Detailed API Reference

Chapter 4. DSX OS

Use this section to connect Dynamic Power Software (DPS) to DSX Exchange and operate the DSX Agent integration.

Start with *DSX Agent Deployment Guide* for deployment prerequisites, authentication, and connection settings.

4.1. DSX Agent Deployment Guide

Deploy the DSX Agent to connect Dynamic Power Software (DPS) NvGrid power management to DSX Exchange. This guide is for Kubernetes administrators configuring the bidirectional connection.

The configuration and topics in this guide match DPS `0.10.0-rc1`. Select the corresponding DSX Agent chart and container image from your release artifacts. Do not assume the agent chart version, image tag, and DPS version are identical.

4.1.1. Connection Overview

The DSX Agent runs as a separate Kubernetes Deployment. It connects to DPS over gRPC, DSX Exchange over Message Queuing Telemetry Transport (MQTT), and PostgreSQL for persistent message queues.

The agent implements the following message paths:

- **Load targets:** Subscribe to `grid/v1/isv/+/loadtarget/set`, validate incoming CloudEvents and JSON Web Signatures (JWS), queue the messages, and submit load schedules to the DPS NvGrid API.
- **Power state:** Read NvGrid metadata, schedules, and status, then publish signed `grid.powerstate.status.v1` CloudEvents on `grid/v1/dsx-flex-agent/<agent-id>/powerstate/status`. The default periodic interval is 30 seconds.
- **Power events:** Register the agent webhook with DPS. DPS calls the agent when ramp-up or ramp-down starts or ends, and the agent publishes the resulting power state to DSX Exchange.

The agent provides these load-target and power-state operations. It does not implement every DSX OS integration. For NvGrid concepts and feed configuration, refer to *NvGrid* and *Managing NvGrid*.

4.1.2. Prepare the Deployment

Before you install the agent, prepare the following resources:

- A configured DPS server, reachable from the agent over verified Transport Layer Security (TLS). Configure feed metadata and the power-management topology before testing load targets. Refer to *Deployment and Configuration Guide*.
- A DPS automation identity authorized to call the NvGrid API. The example uses a DPS password login. Use the authentication mode configured for your server.
- A PostgreSQL database and role for agent queues. The agent applies its database migrations at startup; the role must be able to create and use the required tables, type, function, and trigger.
- DSX Exchange broker addresses, credentials, and topic permissions from your Exchange administrator. Allow the agent to subscribe to `grid/v1/isv/+/loadtarget/set` and publish to its power-state topic. This release also subscribes to the legacy `grid.setloadtarget.v1` topic for migration compatibility; include that subscription in the broker permissions.
- A JWS signing private key for outbound messages and trusted public keys for inbound messages. Agree on the algorithm and key IDs with the Exchange participants. Register the agent public key with the participants that verify its outbound messages.
- Network access to DPS, PostgreSQL, and the MQTT broker, plus your identity provider if you use OAuth 2.0. Allow DPS to reach the agent webhook on TCP port `9100`.
- Helm, `kubectl`, release-matched `dpsctl`, and permission to create the chart resources and Secrets in the target namespace.

This procedure uses namespace `dps`, DPS release `dps`, and agent release `dps-dsx-agent`. The broker certificate must chain to a certificate authority (CA) trusted by the agent container. PostgreSQL and DPS use the CA files you supply. Replace example hostnames, identifiers, and file paths with values for your environment.

4.1.3. Select the Chart and Image

Obtain the versioned DSX Agent chart package and corresponding image tag from your release distribution. Inspect the package before installation:

```
export DSX_AGENT_CHART="/path/to/dps-dsx-agent-<chart-version>.tgz"
export DSX_AGENT_IMAGE_TAG="<release-image-tag>"
helm show chart "${DSX_AGENT_CHART}"
helm show values "${DSX_AGENT_CHART}" > dsx-agent-reference.yaml
```

The agent chart is installed separately from the DPS chart. Retain the reference values for your selected package. If you use a private image mirror, configure `image.repository` and `imagePullSecrets` for that mirror.

4.1.4. Create the Secrets

Create the `dps` namespace if it does not exist:

```
kubectl create namespace dps --dry-run=client -o yaml | kubectl apply -f -
```

Store credentials in local files protected by your organization's secret handling process. Create Kubernetes Secrets from those files. The following commands use an Exchange access token and a DPS login password:

Create Credential and Key Secrets

```
kubectl create secret generic dsx-agent-db --namespace dps \
  --from-file=password=/secure/path/postgres-password
kubectl create secret generic dsx-agent-mqtt --namespace dps \
  --from-file=token=/secure/path/exchange-token
kubectl create secret generic dsx-agent-dps-auth --namespace dps \
  --from-file=password=/secure/path/dps-password
kubectl create secret generic dsx-agent-dps-ca --namespace dps \
  --from-file=ca.crt=/secure/path/dps-ca.crt
kubectl create secret generic dsx-agent-db-ca --namespace dps \
  --from-file=ca.crt=/secure/path/postgres-ca.crt
kubectl create secret generic dsx-agent-jws --namespace dps \
  --from-file=signing-key.pem=/secure/path/agent-signing-key.pem \
  --from-file=exchange-public-key.pem=/secure/path/exchange-public-key.pem
```

The signing key stays with the agent. Share its public key, not the private key. The verification key in this example belongs to the participant that signs incoming load-target messages. If the message includes a JWS `kid` header, it must match the configured verification key ID.

Keep credentials out of Helm values and command-line literals. If your platform provisions Secrets, use the same Secret names and keys or adapt the values to your platform's resources.

4.1.5. Configure the Agent

Save the following overrides as `dsx-agent-values.yaml`. This example uses RSA signing with RS256, verified TLS, PostgreSQL server verification, and mounted credentials:

DSX Agent Values

```
fullnameOverride: dps-dsx-agent
agentConfig:
  agentName: dps
  logLevel: info
  database:
    hostname: postgres.example.com
    port: 5432
    name: dps
    user: dps
    existingSecret: dsx-agent-db
    existingSecretKey: password
    ssl_mode: verify-full
    ssl_root_cert: /secrets/db-ca/ca.crt
  mqtt:
    servers: ["tls://exchange.example.com:8883"]
    clientId: dps-dsx-agent
    username: oauthtoken
    tokenFile: /secrets/mqtt/token
```

(continues on next page)

(continued from previous page)

```
    secretRef:
      name: dsx-agent-mqtt
      key: token
    tls:
      insecure_skip_verify: false
  jws:
    signatureAlgorithm: RS256
    signatureKeyFile: /secrets/jws/signing-key.pem
    signatureKeyId: dps-agent-signing-key
    verificationPublicKeyFiles:
      exchange-signing-key: /secrets/jws/exchange-public-key.pem
  dps:
    serverAddr: dps-server-grpc:8080
    tls:
      transportSecurity: tls
      secretRef:
        name: dsx-agent-dps-ca
    auth:
      dps:
        username: dps-agent
      secretRef:
        name: dsx-agent-dps-auth
        key: password
  dpsWebhookBindAddress: ":9100"
  dpsWebhookAnnounceAddress: http://dps-dsx-agent-webhook:9100
  handlers:
    PowerState:
      enabled: true
      agentId: dps
      publishPeriod: 30s
networkPolicy:
  dps:
    releaseName: dps
    namespace: dps
secrets:
  - name: dsx-agent-db-ca
    secretKey: ca.crt
    mountPath: /secrets/db-ca/ca.crt
  - name: dsx-agent-jws
    secretKey: signing-key.pem
    mountPath: /secrets/jws/signing-key.pem
  - name: dsx-agent-jws
    secretKey: exchange-public-key.pem
    mountPath: /secrets/jws/exchange-public-key.pem
```

Set `handlers.PowerState.agentId` to the identifier assigned for your deployment; it determines the outbound MQTT topic. Use a unique MQTT `clientId` for each simultaneously connected agent instance. The example uses the default single replica.

The `dps.serverAddr` value uses the standard DPS Helm gRPC Service. Adapt its host and port if your release overrides them. The server certificate must cover the hostname you configure.

The chart supports the following authentication alternatives:

- **MQTT:** Replace `tokenFile` with `oauth2` settings if Exchange provides client credentials. Configure `tokenUrl`, `scopes`, and `credentialsFile`, and point `mqtt.secretRef` to the credentials Secret. The credential file defaults to JSON fields `client_id` and `client_secret`. Select only one MQTT credential source.
- **DPS:** Use a bearer-token Secret by omitting `auth.dps` and configuring `auth.secretRef`. For OAuth 2.0, replace `auth.dps` with `auth.oauth2` settings and use a credentials Secret. Select only one DPS credential source.
- **Mutual TLS:** Set `dps.tls.transportSecurity: mtls` and reference a Secret containing `ca.crt`, `tls.crt`, and `tls.key`.

For incoming-command-only deployments, set `handlers.PowerState.enabled: false`. Verification keys and the signature algorithm are still required; the agent does not initialize the outbound signer or register the power-state webhook in this mode.

4.1.6. Configure Webhook Delivery

The standard webhook URL is `http://dps-dsx-agent-webhook:9100`. The DPS chart in `0.10.0-rc1` includes this URL in `dps.nvgrid.webhook.whitelist`. Confirm that your server's deployed values permit it before starting the agent.

If you use another service name or namespace, update the announce URL and the DPS whitelist to match. For a separate agent namespace, use a service DNS name that resolves from the DPS namespace. Set `networkPolicy.dps.releaseName` and `networkPolicy.dps.namespace` to the server's release and namespace. Retain any other whitelist entries your deployment requires.

The webhook listener uses HTTP. Restrict its ingress to DPS. The chart enables an ingress NetworkPolicy for this purpose; your cluster network plugin must enforce NetworkPolicy. If your security policy requires encryption on this path, provide it through your platform's service mesh or gateway and configure the announced URL and allowed peers accordingly.

4.1.7. Install and Check Startup

Install the selected chart with your overrides and explicit image tag:

```
helm upgrade --install dps-dsx-agent "${DSX_AGENT_CHART}" \
  --namespace dps \
  --values dsx-agent-values.yaml \
  --set-string image.tag="${DSX_AGENT_IMAGE_TAG}" \
  --wait --timeout 5m
```

Check the Deployment and its logs:

```
kubectl rollout status deployment/dps-dsx-agent --namespace dps
kubectl logs deployment/dps-dsx-agent --namespace dps --tail=200
```

Confirm that the logs report `connected to mqtt` and, with power-state publication enabled, `starting DPS webhook server`. Resolve any database, JWS, subscription, or DPS authentication errors. Webhook registration happens asynchronously; a registration failure can leave the agent running without webhook delivery.

The health endpoints return HTTP `200` while their server is running. A ready pod establishes process health but does not establish successful message delivery, webhook registration, or power control.

4.1.8. Verify the Signed Message Flow

Use your configured `dpsctl` connection to confirm that NvGrid has feed metadata and readable status:

```
dpsctl nvgrid get-metadata
dpsctl nvgrid get-status
dpsctl nvgrid get-current
```

For client connection and authentication setup, refer to *Installing dpsctl*. For schedule queries, refer to *dpsctl nvgrid*.

Use the DPS `0.10.0-rc1` `DSX Flex AsyncAPI schema archive` when preparing the test command. Extract `dsx-flex.yaml` from the archive. The schema defines the envelope at `components.schemas.LoadTargetSetEvent` and the payload at `components.schemas.SetLoadTargetData`.

Set `type` to `grid.loadtarget.set.v1` and `specversion` to `"1.0.2"`. Sign the JSON payload and place its compact JWS serialization in the envelope's `data` string. The JWS protects only that payload; the envelope fields, including the correlation ID, are unsigned.

The shipped DSX schema requires the exact `specversion` value `"1.0.2"`. If your CloudEvents SDK emits `"1.0"`, configure it to emit `"1.0.2"` for this integration; `"1.0"` does not pass the release schema validation.

With `handlers.PowerState.enabled: true`, complete the following verification with your Exchange administrator in an approved test environment:

1. Receive a periodic message on `grid/v1/dsx-flex-agent/<agent-id>/powerstate/status`. Verify its JWS signature with the agent's public key and confirm the feed metadata, schedules, and status against the NvGrid API results.
2. Publish a schema-valid, signed load-target CloudEvent through Exchange on `grid/v1/isv/<isv-id>/loadtarget/set`. Use a unique event ID, a UUID correlation ID, known feed tags, and an approved target and time interval. Follow the Exchange message contract, including its tracing fields.
3. Read the schedule for the test interval with `dpsctl nvgrid get-schedule` and read status with `dpsctl nvgrid get-status`. Confirm that DPS accepted the target. Retain the correlation ID when comparing the schedule and subsequent power-state messages.
4. Confirm ramp events reach the agent webhook and produce signed power-state messages on Exchange. Check both the agent and DPS logs for delivery or processing errors.
5. Confirm the target's effects through DPS status and your test environment's device telemetry. A received message or accepted schedule alone does not prove hardware actuation.

For incoming-command-only deployments with `handlers.PowerState.enabled: false`, complete steps 2, 3, and 5. Check the accepted schedule's correlation ID and the agent's processing logs. Skip the outbound message and webhook checks; this mode does not publish power-state messages or register that webhook.

Do not send a production load-target command solely to test connectivity. Use the approved target, rollback, and telemetry procedure for your site.

4.1.9. Troubleshooting

Use the following checks to isolate deployment and message-flow failures:

Symptom	Check
Pod does not start	Check chart and image compatibility, image access, Secret names and keys, mounted-file permissions, and configuration parsing errors.
Database migration fails	Check database hostname, TLS CA and hostname verification, credentials, and migration permissions.
MQTT connection or subscription fails	Check broker DNS and egress, certificate trust, token validity or OAuth 2.0 settings, topic permissions, and duplicate client IDs.
JWS setup or message validation fails	Check the algorithm, PEM key formats, signature key ID, trusted sender key IDs, and the release-matched message schema.
Webhook registration fails	Check NvGrid authorization and the server whitelist. A non-network registration error stops webhook startup.
Periodic messages arrive but ramp events do not	Check webhook service DNS, port `9100`, NetworkPolicy enforcement, allowed peers, and server delivery logs.
Load target does not appear in NvGrid	Check signature or schema rejection, feed tags, time intervals, and agent queue-processing errors.
Pod is ready but messages stop	Check dependency connectivity and message-processing logs; the readiness endpoint does not inspect these dependencies.

4.1.10. Update Credentials and Configuration

Update Secrets through your platform's credential-management process. MQTT and DPS credential Secret references use directory mounts so Kubernetes can project changed files. Token caches and connection lifetimes can delay when a new credential takes effect.

The additional `secrets` entries use file mounts with `subPath`. Restart the agent after changing those files, including JWS keys or the database CA. Also restart after changing startup-loaded verification keys or database credentials. Coordinate JWS key changes with Exchange participants before the restart.

After applying your changes, restart the Deployment and repeat the message-flow verification:

```
kubectl rollout restart deployment/dps-dsx-agent --namespace dps
kubectl rollout status deployment/dps-dsx-agent --namespace dps
```

If you supply the full configuration through `config.existingSecret`, keep chart-managed ports and file mounts consistent with that file. Changes to an existing configuration Secret do not add a chart-generated configuration checksum; restart the Deployment to load the updated file.

4.2. Integration Architecture

The diagram shows the DSX Agent power-management integration, its dependencies, and the DPS control and hardware paths. Dashed connections show optional integrations or mounted configuration. In the diagram, MQTT means Message Queuing Telemetry Transport, and TLS means Transport Layer Security.

The DSX Agent and the DPS device agent have different roles. The DSX Agent connects NvGrid to Exchange and uses gRPC for load targets, status queries, and webhook registration. Hardware access belongs to the DPS server or its device agents. Ramp-event callbacks travel directly from the control service to the DSX Agent when power-state publication is enabled. A remote device agent initiates its gRPC connection to the control service.

PostgreSQL stores the agent queues and DPS control state. These can use separate databases or database servers. Configure the agent database independently of the DPS database.

Power Steering and the Redfish proxy are optional. Use direct Redfish access or a configured proxy for each hardware path. Power Steering requests GPU limits through DPS, which validates and applies them. Refer to *DPS Power Steering Controller* for its control loop.

The credential source depends on your deployment. OAuth 2.0 token acquisition is optional; mounted token files and DPS password login are alternatives. The mounted JSON Web Signature (JWS) keys sign or verify message payloads separately from connection authentication.

4.3. Integration Responsibilities

The integration connects three components:

- **DSX Exchange** carries MQTT messages between the DSX Agent and external grid applications.
- **DSX Agent** validates incoming messages, queues them in PostgreSQL, and translates load targets into DPS NvGrid API calls. It also signs and publishes power-state messages to DSX Exchange.
- **DPS control service** owns feed metadata, load-target schedules, and power management. The DSX Agent uses the gRPC API to request changes and read status.

The implemented bridge handles load-target requests and power-state status. Do not assume that every event defined in a DSX schema has a handler in the DSX Agent.

Refer to *NvGrid* for feed constraints, eligible resource groups, and power restoration. Use *Managing NvGrid* to inspect feed metadata, schedules, and status directly through DPS.

4.4. Load Targets and Status

A load-target request follows this path:

1. An external application publishes a load-target message to DSX Exchange.
2. The DSX Agent validates the message schema and signature, then stores the request for processing.
3. The DSX Agent calls `NvGrid.SetLoadTarget` to update the feed schedule.
4. DPS applies the schedule through its power-management controls. Check status to determine whether the requested target is met.

When power-state publication is enabled, the agent publishes periodic snapshots and updates after processing load targets. These snapshots include feed metadata, scheduled targets, calculated load, and transition status. Calculated load is a DPS value and can differ from measured data center consumption.

With power-state publication enabled, the agent registers its HTTP callback address through `NvGrid.RegisterWebhook` for the start and end of ramp-up and ramp-down events. DPS sends notifications to that address, and the agent uses them to publish power-state updates. Configure the advertised address for reachability from DPS and include it in the DPS webhook whitelist (`dps.nvgrid.webhook.whitelist`). The agent callback uses HTTP and does not authenticate its sender. Restrict inbound callback traffic to DPS and ensure your cluster network plugin enforces the ingress NetworkPolicy. Refer to *DSX Agent Deployment Guide*.

4.5. Deployment Boundaries and Verification

Deploy the DSX Agent with access to DSX Exchange, PostgreSQL, and the DPS gRPC API. Configure MQTT authentication, DPS API authentication, and message signing and verification for your environment. For connection settings, refer to *DSX Agent Deployment Guide*.

Hardware access remains part of the DPS deployment. If you use a baseboard management controller (BMC) proxy, including the NVIDIA Infra Controller proxy, configure it through *Redfish Proxy Configuration*.

The DSX Agent readiness endpoint reports that its health server responds; it does not verify the database, MQTT connection, DPS API, or message delivery. Before accepting the integration, submit an approved signed load target in a test environment and confirm its schedule and status through NvGrid. If power-state publication is enabled, receive the corresponding message on DSX Exchange and verify its signature with the agent public key. Check hardware telemetry when you need to establish the actual power response.

Chapter 5. Release Notes

- *DPS 0.10.x*
- *DPS 0.9.x*
- *DPS 0.8.x*

5.1. DPS 0.10 Release Notes

- **Previous release:** DPS 0.9.0

DPS 0.10.0 enables DSX Exchange integration including DSX Flex, DPS agent authentication, Nautobot import, and deployment routing. It also improves power-allocation recovery and component-policy handling.

5.1.1. Upgrade Information

5.1.1.1 Review Optional External Authentication

External authentication is opt-in through `dps.auth.external.enabled`. This mode delegates authentication and authorization to an independently installed proxy and binds the DPS gRPC backend to loopback. It disables DPS token issuance.

Install and configure the proxy, Service routing, certificates, and UI login before enabling this mode. The Helm chart does not install the authorization proxy for you. The HTTP gateway connects to the local proxy through `dps.auth.external.proxyPort`.

5.1.2. Added

This release candidate adds the following capabilities:

- Added support for DSX Exchange integration & DSX Flex integrations via DSX Exchange
- Added Nautobot manual import support for DSX
- Added shared DPS client authentication to the DPS agent and DSX Agent, including OAuth2 client credentials, DPS login, and bearer-token files to support DSX Launch Layer integration.
- Added OAuth2 client-credentials and token-file authentication for the DSX Agent MQTT connection to DSX Exchange.
- Added a configurable JWS signing key ID for DSX Agent messages.

- Added GraphQL transport for Nautobot entity imports, with REST fallback when the GraphQL endpoint is unavailable. Authentication errors and GraphQL query errors do not trigger fallback.
- Added support for an independently supplied authentication and authorization sidecar to support DSX Launch Layer integration.

5.1.3. Improved

This release candidate includes the following improvements:

- Allowed Kubernetes Ingress and Gateway API routes to coexist for DPS and the UI. Coordinate DNS, certificates, controllers, and authentication across both entry paths before enabling them together.
- Updated the embedded Power Steering Helm dependency to version 2.2.1.
- Improved DSX Agent Flex command handling and signed-message validation.
- Improved GPU-only Node Manager policy handling by completing required CPU and memory component policies from available hardware policy information.

5.1.4. Fixed

This release candidate includes the following fixes:

- Corrected power give-back recovery and allocation synchronization.
- Preserved shared-GPU recovery state and completed GPU and give-back state when an update is acknowledged.
- Corrected deadlocks in topology activation and deactivation, resource-group release, and GPU policy updates.
- Created the ServiceAccount required by agent-only Helm deployments.

5.1.5. Important DSX Configuration Information

5.1.5.1 Using DSX Agent Connection Settings

The DSX Agent uses the shared DPS client configuration. Update custom Helm values and externally supplied configuration files before upgrading.

The following settings replace the previous `agentConfig.dps` fields in Helm values. JSON application configuration uses the same field names under `dps`:

Previous Setting	DPS 0.10 Setting
`host` and `port`	`serverAddr`, in `host:port` form
`caCertPath`	`tls.certs.caFile`
`insecure: true`	`tls.transportSecurity: insecure`
`insecure: false`	`tls.transportSecurity: tls`
`insecureTLSSkipVerify`	`tls.insecureSkipVerify`

Standalone YAML application configuration uses `dps.server_addr`, `dps.tls.transport_security`, `dps.tls.insecure_skip_verify`, and `dps.tls.certs.ca_file` for the corresponding settings. These YAML field names differ from the Helm values, which the chart renders as JSON.

For mutual TLS, use `tls.transportSecurity: mtls` and set `tls.certs.caFile`, `tls.certs.certFile`, and `tls.certs.keyFile` to the mounted CA, client certificate, and client key. Their standalone YAML names are `tls.certs.ca_file`, `tls.certs.cert_file`, and `tls.certs.cert_key_file`. In Helm values, you can instead set `agentConfig.dps.tls.secretRef.name` to a Secret containing `ca.crt`, `tls.crt`, and `tls.key`. Authenticated connections require verified TLS. Plaintext authentication requires an explicit development override; authenticated TLS connections cannot skip certificate verification.

Configure DPS authentication under `agentConfig.dps.auth` and broker authentication under `agentConfig.mqtt`. Broker credentials and DPS credentials serve separate connections. Review JSON Web Signature (JWS) signing and verification keys independently of those connection credentials.

The chart removes `dpsCredentials.existingSecret` and `dpsCredentials.mountPath`. For DPS password login, set `agentConfig.dps.auth.dps.username`, and reference the password Secret through `agentConfig.dps.auth.secretRef.name` and `agentConfig.dps.auth.secretRef.key`. You can reuse the previous Secret's `password` key. The chart mounts the selected file and configures `auth.dps.passwordFile`; it no longer reads a `username` file from the previous credential directory.

For externally supplied JSON configuration, move `dps.username` to `dps.auth.dps.username` and replace `dps.secretMountPath` with `dps.auth.dps.passwordFile`, pointing to a mounted password file. Move an existing inline `dps.password` to `dps.auth.dps.password` if you retain that credential source. YAML configuration uses `password_file` instead of `passwordFile`. Configure exactly one password source.

5.1.5.2 DSX MQTT Topics

DPS 0.10 changes the Exchange MQTT topics for these message paths. The agent, publishers, and consumers need matching broker access-control permissions:

Message Path	Previous Topic	DPS 0.10 Topic
Outbound power state	`grid/v1/poweragent/<agent-id>/powerstate/status`	`grid/v1/dsx-flex-agent/<agent-id>/powerstate/status`
Incoming load targets	`grid.setloadtarget.v1`	`grid/v1/isv/<isv-id>/loadtarget/set`

Power-state consumers must subscribe to the new outbound topic, and the agent needs permission to publish to it. Load-target publishers use their concrete ISV topic. The agent subscribes to `grid/v1/isv/+/loadtarget/set` and retains the legacy `grid.setloadtarget.v1` subscription for migration compatibility. Permit both subscriptions in the broker configuration. Power state is published on the new outbound topic. Consumers still subscribed only to `grid/v1/poweragent/<agent-id>/powerstate/status` receive no new power-state updates after the upgrade.

Refer to *Managing NvGrid* for feed metadata, load targets, and status.

5.1.5.3 Nautobot Import Configuration

Nautobot entity generation selects GraphQL by default when the request omits the transport. REST is an explicit option. Review custom import clients and allowed query parameters before upgrading: parameters must map to supported Nautobot device-query arguments.

5.2. DPS 0.9.0 Release Notes

- **Release date:** September 11, 2026
- **Previous release:** DPS 0.8.3

DPS 0.9.0 introduces MaxLPS: policy-driven dynamic power management, an improved policy interface, expanded observability dashboards and alerting, Redfish proxy support, Vera Rubin platform support, improved dynamic power allocation, and new authentication and transport-security capabilities.

These notes cover DPS 0.9.0.

5.2.1. Important Upgrade Information for MaxLPS Power Pilots

5.2.1.1 Upgrading a MaxLPS Pilot Deployment From DPS 0.8.x

For MaxLPS pilot test deployments, it is generally safe to deactivate the DPS 0.8.x topology, uninstall the DPS 0.8.x Helm release, and install a DPS 0.9 deployment. Before deactivating the topology, deactivate every active resource group that contains entities in that topology. Treat this process as a clean replacement rather than an in-place upgrade. Retain the topology, policy, credential, database, and Helm values that the pilot must preserve before uninstalling. This procedure is not a substitute for an approved production backup and recovery plan.

Deploy DPS Core (the DPS server) and ensure that at least one DPS agent is connected. The chart deploys DPS Core and its ingress (`dps.enabled: true`, `dps.ingress.enabled: true`), the DPS UI and its ingress (`ui.enabled: true`, `ui.ingress.enabled: true`), bundled PostgreSQL with persistent storage (`postgresql.enabled: true`), and the Power Steering 2.1 subchart (`power-steering.enabled: true`) by default. To deploy an agent in the same cluster as DPS Core, set `agent.enabled: true` when you install or upgrade the chart. Unless overridden, the agent connects to the release's DPS gRPC service.

For configuration and client-transition details for the DPS 0.9 deployment, refer to *Further Upgrade Information*.

5.2.1.2 Review the Default Policy Bundle

New DPS 0.9.0 deployments set `global_policy_bundle` to `MaxLPS`. When a topology is created or imported, or a resource group is created, with an empty policy-bundle selection, DPS resolves that selection to the current global setting and persists the resolved bundle name. An explicit empty selection on a deactivated topology update resolves the same way.

For a retained DPS 0.8.x database, this resolution also occurs during 0.9.0 recovery for existing topologies and resource groups whose stored selection is empty. DPS persists the resolved name and recreates the policy binding for active objects. Existing nonempty policy-bundle selections are not rewritten; likewise, a persisted `global_policy_bundle` value, including an intentionally empty value, takes precedence over the shipped `MaxLPS` default.

The resolved topology bundle becomes the topology-wide policy when the topology is active. A resolved resource-group bundle is the resource group's higher-precedence policy binding, which composes with or supersedes topology policies by policy type. `MaxLPS` is therefore a default-selection mechanism, not a live global policy binding. Before retaining a DPS 0.8.x database, or creating or importing new objects, inspect the available bundles and the current setting. Test the resulting selections in a non-production copy, then set a different default only when it matches the deployment's intended policy:

```
dpsctl policy bundle list
dpsctl settings list
dpsctl settings update --set global_policy_bundle=MaxP
```

5.2.2. Added

5.2.2.1 Policy-Driven Power Management

- Introduced a typed policy framework based on policy bundles, bindings, scopes, composition, and effective-policy inspection.
- Added support for Computer System Limits, Dynamic Power Management, and Aggregate Power Limits policy models.
- Added `ComputerSystemLimits.v1` floor, target, and ceiling specifications for node, GPU, CPU, and memory power. Node ranges can use watts or a percentage of the hardware maximum. Component ranges can use watts or a percentage of the node ceiling. GPU watt values apply to the aggregate GPU budget of one ComputerSystem, not to each GPU separately.
- Enforced the operating-range contract `floor <= target <= ceiling`. A node target is a desired operating point within the hardware-clamped range. Component targets become requested GPU, CPU, or memory policy values when supplied. Floors and ceilings constrain policy composition, admission, and redistribution rather than independently programming component setpoints.
- When multiple limits apply, DPS uses the most restrictive valid range: the highest floor and the lowest non-zero target and ceiling. DPS rejects incompatible combinations rather than relaxing them.
- Added default policy bundles — MaxP, MaxQ, and MaxLPS — that provide model-specific Computer System Limits ranges.
- Improved the policy interface across the UI, CLI, and APIs with effective-policy inspection, policy-bundle management, and resource-group policy-bundle attachment.
- Enabled MaxLPS settings by default and changed the default Dynamic Power Management strategy to proportional.

5.2.2.2 MaxLPS Dynamic Power Management

- Delivered the first release of MaxLPS policy-driven dynamic power management, combining the MaxLPS policy bundle with dynamic power allocation for dynamic GPU-power redistribution.
- Upgraded the embedded PRS deployment with the independently versioned `power-steering` Helm subchart, version 2.1.0.

5.2.2.3 Hardware, BMC, and Agent Operations

- Added support for Vera Rubin NVL72, including catalog, power-plugin, and BMC-health support.
- Added durable GPU hardware-identity discovery and Redfish GPU UUID exposure through DPS APIs, metrics, resource groups, and policy-update results.
- Added the DPS agent control-plane service, remote device configuration, agent discovery API and CLI support, and node-compliance alerts.
- Added configurable per-node BMC-health timeouts.

5.2.2.4 Authentication, Security, and Observability

- Added OIDC/JWKS validation, `dpsctl` client-credentials and browser device-code login, and OIDC PKCE support in the UI.
- Added LDAP group profiles, browser-session logout and revocation, refresh-token rotation, and replay detection.
- Added BMC client CA and client-certificate support, per-connection TLS-verification controls, server TLS version and cipher validation, and optional secured Prometheus metrics access.
- Improved observability dashboards and alerting with Grafana handoffs for power compliance, BMC latency, traces, and server health; active-resource-group priority metrics; NvGrid unmet-budget reasons; and node-compliance alerts.

5.2.2.5 UI and SDKs

- Modernized the bundled UI with policy-bundle management, BMC-health workflows, agent views, topology and inventory workflows, and expanded NvGrid views.
- Added a Go client, typed C/C++ API error codes, and updated Python service APIs.

5.2.3. Improved

- Added Agent metrics scraping and trace propagation.
- Improved topology handling and GPU policy updates.
- Improved GPU policy application by staging reductions and increases, applying fixed GPU targets on supported platforms, and returning an explicit in-progress result when a `dpsctl` deadline expires while the server continues the update.
- Moved Redfish telemetry to the platform environment-metrics endpoint where applicable.
- Added Redfish proxy support, Redfish URL overrides, proxy-aware transport, and retries for proxy-connectivity failures.
- Improved BMC-policy persistence, telemetry-failure handling, partial-BMC-loss handling, tracing, and metric-cardinality control.
- Changed BMC health checks to report telemetry sampling failures as check failures.

5.2.4. Fixed

- Corrected stale or invalid platform-telemetry handling and bounded Redfish operation latency.

- Corrected resource-group activation, deactivation, deletion, and topology-validation failure handling.
- Corrected BMC session and bearer-token lifecycle handling, GPU policy timeout reporting, proxy retry behavior, and fixed-target GPU actuation.
- Hardened JWT metadata parsing, LDAP authentication, BMC TLS verification, dependency versions, and UI runtime dependencies.

5.2.5. Deprecated

- Legacy single-policy fields are deprecated in favor of Policy Bundles.

5.2.6. Known Issues

- `ResourceGroupUpdate` cannot change a resource group's policy bundle in DPS 0.9. Support is planned for a future release.

5.2.7. Further Upgrade Information

5.2.7.1 Update Telemetry Settings

If your Helm values use `dps.telemetry.provider`, remove it. Use `dps.telemetry.metrics.source` instead.

5.2.7.2 Review Power Steering Defaults

The MaxLPS power steering controller implements dynamic power allocation and is enabled by default and uses a 5-second allocation interval.

If dynamic power allocation MUST remain disabled, explicitly set both `dps.prs.enabled` and `power-steering.enabled` values to `false` before you deploy the replacement release.

The embedded `prs` subchart is replaced by the independently versioned `power-steering` subchart. Migrate custom `prs.*` values to their `power-steering.*` equivalents. In-cluster service names change from `*-prs-config` and `*-prs-controller` to `*-power-config` and `*-power-controller`. Preserve explicit `dps.prs.hostPort` and `dps.prs.controllerHostPort` overrides where needed.

The updated power steering controller changes v2 Prometheus metric families from `prs_*` to `power_steering_*`. Update dashboards, recording rules, and alerts accordingly. Some metric names require more than a mechanical prefix replacement.

5.2.7.3 Configure Production Authentication and TLS

DPS 0.9 does not generate gRPC TLS material. Server `tls` and `mtls` require `dps.serverTLS.existingSecret` with `tls.crt`, `tls.key`, and `ca.crt`. Agent `tls` requires `agent.tls.existingSecret` with `ca.crt`. Agent `mtls` requires `ca.crt`, `tls.crt`, and `tls.key`.

The default chart is intended for bring-up only. Server and agent transport default to `insecure`, and an empty `dps.auth.authenticatorType` resolves to `alwaysAllow` unless LDAP is enabled. Production deployments must configure TLS or mTLS, LDAP or OIDC authentication, and an HTTPS UI ingress.

5.2.7.4 Upgrade Clients and Build Environments

Regenerate or upgrade gRPC, Python, Go, and C/C++ clients for the DPS 0.9 API. Device-controller legacy single-policy fields are deprecated. Use complete effective-policy snapshots instead.

Revalidate UI automation and custom UI integrations because the bundled application uses UI v2 and a Node-based `/api` proxy.

5.3. 0.8.3 Release Notes

These release notes cover changes from DPS 0.8.2 to DPS 0.8.3.

5.3.1. Added

- Added support for DGX B200 EM systems through the new `DGX_B200EM` Redfish device plugin.
- Added support for DGX B300 EM systems through the new `DGX_B300EM` Redfish device plugin.
- Added Redfish GPU index detection for platforms whose GPU endpoint numbering differs from the default.
- Added configurable GPU endpoint index base support through the `indexBase` device specification field.
- Added Helm configuration for Redfish per-operation timeouts.

5.3.2. Improved

- Improved Redfish GPU power-limit actuation performance by parallelizing GPU write operations.
- Improved Redfish connection tuning with separate controls for metrics and actuation paths.
- Improved telemetry resilience by returning partial topology telemetry when one or more BMC telemetry requests fail.
- Improved BMC simulator timing handling by migrating workload timing fields to `time.Duration`.

5.3.3. Fixed

- Fixed async resource group activation task status lookup.
- Fixed Redfish operation behavior by bounding per-operation latency.
- Fixed Redfish telemetry handling to reject invalid `0xFFFFFFFF` sentinel values.
- Fixed stale platform environment metric handling in Redfish telemetry.
- Fixed high-cardinality Prometheus metric labels to improve observability scalability.
- Backported disable-node behavior to prevent node power controls on the 0.8.x release branch.

5.3.4. Security

No security-specific changes are included in this release.

5.3.5. Deprecated

No features are deprecated in this release.

5.3.6. Removed

No features are removed in this release.

5.3.7. Compatibility and Support Notes

- DPS version: `0.8.3`
- DPS Helm chart version: `0.8.3`
- PRS version remains `1.0.12`.
- Operators should review Redfish timeout and concurrency settings before rollout, especially in environments with BMCs sensitive to concurrent actuation requests.

Chapter 6. NVIDIA SOFTWARE LICENSE AGREEMENT

IMPORTANT NOTICE – PLEASE READ AND AGREE BEFORE USING THE SOFTWARE

This software license agreement ("Agreement") is a legal agreement between you, whether an individual or entity, ("you") and NVIDIA Corporation ("NVIDIA") and governs the use of the NVIDIA Dynamic Power Software software and materials that NVIDIA delivers to you under this Agreement ("Software"). NVIDIA and you are each a "party" and collectively the "parties."

By using the Software, you are affirming that you have read and agree to this Agreement.

If you don't accept all the terms and conditions below, do not use the Software.

6.1. 1. License Grants

6.1.1. 1.1 License Grant to You

The Software made available by NVIDIA to you is licensed, not sold. Subject to the terms of this Agreement, NVIDIA grants you a limited, non-exclusive, revocable, non-transferable, and non-sublicensable (except as expressly granted in this Agreement), license to:

(a) install and use copies of the Software,

(b) configure the Software using configuration files provided (if applicable), and

(c) modify and create derivative works of any sample or example source code NVIDIA delivers to you as part of the Software ("Derivatives") (if applicable).

All the foregoing (a) through (c) are only for your internal use in systems with NVIDIA GPUs and NVIDIA CPUs ("Purpose").

6.1.2. 1.2 License Grant to NVIDIA

Subject to the terms of this Agreement, you grant NVIDIA and its affiliates a non-exclusive, perpetual, irrevocable, sublicensable, worldwide, royalty-free, fully paid-up and transferable license, under your intellectual property rights, to publicly perform, publicly display, reproduce, use, make, have made, sell, offer for sale, distribute (through multiple tiers of distribution), import, create derivative works of and otherwise commercialize and exploit at NVIDIA's discretion any Derivatives created by or for you. You may, but are not required to, deliver any Derivatives to NVIDIA.

6.2. 2. License Restrictions

Your license to use the Software and Derivatives is restricted as stated in this Section 2 (“License Restrictions”). You will cooperate with NVIDIA and, upon NVIDIA’s written request, you will confirm in writing and provide reasonably requested information to verify your compliance with the terms of this Agreement. You may not:

1. Use the Software or Derivatives for any purpose other than the Purpose;
2. Sell, rent, sublicense, transfer, distribute or otherwise make available to others (except authorized users as stated in Section 3 (“Authorized Users”)) any portion of the Software or Derivatives, except as expressly granted in Section 1.1 (“License Grant to You”);
3. Reverse engineer, decompile, or disassemble the Software components provided in binary form, nor attempt in any other manner to obtain source code of such Software;
4. Modify or create derivative works of the Software, except as expressly granted in Section 1.1 (“License Grant to You”);
5. Change or remove copyright or other proprietary notices in the Software;
6. Bypass, disable, or circumvent any technical limitation, encryption, security, digital rights management or authentication mechanism in the Software;
7. Use the Software or Derivatives in any manner that would cause them to become subject to an open source software license, subject to the terms in Section 7 (“Components Under Other Licenses”);
8. Use the Software or Derivatives for the purpose of developing competing products or technologies or assist a third party in such activities; or
9. Use the Software or Derivatives in violation of any applicable law or regulation in relevant jurisdictions.

6.3. 3. Authorized Users

You may allow employees and contractors of your entity or of your subsidiary(ies), and for educational institutions also enrolled students, to internally access and use the Software as authorized by this Agreement from your secure network to perform the work authorized by this Agreement on your behalf. You are responsible for the compliance with the terms of this Agreement by your authorized users. Any act or omission that if committed by you would constitute a breach of this Agreement will be deemed to constitute a breach of this Agreement if committed by your authorized users.

6.4. 4. Pre-Release

Software versions identified as alpha, beta, preview, early access or otherwise as pre-release (“Pre-Release”) may not be fully functional, may contain errors or design flaws, and may have reduced or different security, privacy, availability and reliability standards relative to NVIDIA commercial offerings. You use Pre-Release Software at your own risk. NVIDIA did not design or test the Software for use in production or business-critical systems. NVIDIA may choose not to make available a commercial version of Pre-Release Software. NVIDIA may also choose to abandon development and terminate the availability of Pre-Release Software at any time without liability.

6.5. 5. Third Party Privacy Practices

The Software may contain links to third party websites and services. NVIDIA encourages you to review the privacy statements on those sites and services that you choose to visit to understand how they may collect, use and share your data. NVIDIA is not responsible for the privacy statements or practices of third-party sites or services.

6.6. 6. Updates

NVIDIA may at any time and at its option, change, discontinue, or deprecate any part, or all, of the Software, or change or remove features or functionality, or make available patches, workarounds or other updates to the Software. Unless the updates are provided with their separate governing terms, they are deemed part of the Software licensed to you under this Agreement, and your continued use of the Software is deemed acceptance of such changes.

6.7. 7. Components Under Other Licenses

The Software may include or be distributed with components provided with separate legal notices or terms that accompany the components, such as open source software licenses and other license terms ("Other Licenses"). The components are subject to the applicable Other Licenses, including any proprietary notices, disclaimers, requirements and extended use rights; except that this Agreement will prevail regarding the use of third-party open source software, unless a third-party open source software license requires its license terms to prevail. Open source software license means any software, data or documentation subject to any license identified as an open source license by the Open Source Initiative (http://opensource.org), Free Software Foundation (http://www.fsf.org) or other similar open source organization or listed by the Software Package Data Exchange (SPDX) Workgroup under the Linux Foundation (http://www.spdx.org).

6.8. 8. Ownership

6.8.1. 8.1 NVIDIA Ownership

The Software, including all intellectual property rights, is and will remain the sole and exclusive property of NVIDIA or its licensors. Except as expressly granted in this Agreement, (a) NVIDIA reserves all rights, interests and remedies in connection with the Software, and (b) no other license or right is granted to you by implication, estoppel or otherwise.

6.8.2. 8.2 Your Ownership

Subject to the rights of NVIDIA and its suppliers in the Software, which continue to be licensed as stated in this Agreement, even when incorporated in your products or services, and the extent permitted by applicable law, as between you and NVIDIA, you hold all rights, title and interest in and to your products, services and Derivatives you develop as permitted in this Agreement including their respective intellectual property rights.

6.9. 9. Feedback

You may, but you are not obligated to, provide suggestions, requests, fixes, modifications, enhancements, or other feedback regarding the Software (collectively, "Feedback"). Feedback, even if designated as confidential by you, will not create any confidentiality obligation for NVIDIA or its affiliates. If you provide Feedback, you grant NVIDIA, its affiliates and its designees a non-exclusive, perpetual, irrevocable, sublicensable, worldwide, royalty-free, fully paid-up and transferable license, under your intellectual property rights, to publicly perform, publicly display, reproduce, use, make, have made, sell, offer for sale, distribute (through multiple tiers of distribution), import, create derivative works of and otherwise commercialize and exploit the Feedback at NVIDIA's discretion.

6.10. 10. No Publicity

You may not issue any public statements about this Agreement, or any information or results related to your use of the Software, without NVIDIA's prior written approval.

6.11. 11. Termination

6.11.1. 11.1 Termination for Convenience

Either party may terminate this Agreement at any time with thirty (30) days' advance written notice to the other party.

6.11.2. 11.2 Termination for Cause

If you commence or participate in any legal proceeding against NVIDIA with respect to the Software, this Agreement will terminate immediately without notice. Either party may terminate this Agreement for cause if:

(a) The other party fails to cure a material breach of this Agreement within ten (10) days of the non-breaching party's written notice of the breach; or

(b) the other party breaches its confidentiality obligations or license rights under this Agreement, which termination will be effective immediately upon written notice.

6.11.3. 11.3 Effect of Termination

Upon any expiration or termination of this Agreement, you will promptly (a) stop using and return, delete or destroy NVIDIA confidential information and all Software received under this Agreement, and (b) delete or destroy Derivatives created under this Agreement, unless an authorized NVIDIA representative provides prior written approval that you may keep a copy of the Derivatives solely for archival purposes. Upon written request, you will certify in writing that you have complied with your obligations under this Section 11.3 ("Effect of Termination").

6.11.4. 11.4 Survival

Section 1.2 (“License Grant to NVIDIA”), Section 6 (“Updates”), Section 7 (“Components Under Other Licenses”), Section 8 (“Ownership”), Section 9 (“Feedback”), Section 10 (“No Publicity”), Section 11.3 (“Effect of Termination”), Section 11.4 (“Survival”), Section 12 (“Disclaimer of Warranties”), Section 13 (“Limitation of Liability”), Section 14 (Use in Mission Critical Applications), Section 15 (“Governing Law and Jurisdiction”) and Section 16 (“General”) will survive any expiration or termination of this Agreement.

6.12. 12. Disclaimer of Warranties

THE SOFTWARE IS PROVIDED BY NVIDIA AS-IS AND WITH ALL FAULTS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, NVIDIA DISCLAIMS ALL WARRANTIES AND REPRESENTATIONS OF ANY KIND, WHETHER EXPRESS, IMPLIED OR STATUTORY, RELATING TO OR ARISING UNDER THIS AGREEMENT, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF TITLE, NONINFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, USAGE OF TRADE AND COURSE OF DEALING. NVIDIA DOES NOT WARRANT OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR COMPLETENESS OF ANY THIRD-PARTY INFORMATION, TEXT, GRAPHICS, LINKS CONTAINED IN THE SOFTWARE. WITHOUT LIMITING THE FOREGOING, NVIDIA DOES NOT WARRANT THAT THE SOFTWARE WILL MEET YOUR REQUIREMENTS, ANY DEFECTS OR ERRORS WILL BE CORRECTED, ANY CERTAIN CONTENT WILL BE AVAILABLE; OR THAT THE SOFTWARE IS FREE OF VIRUSES OR OTHER HARMFUL COMPONENTS. NO INFORMATION OR ADVICE GIVEN BY NVIDIA WILL IN ANY WAY INCREASE THE SCOPE OF ANY WARRANTY EXPRESSLY PROVIDED IN THIS AGREEMENT.

6.13. 13. Limitations of Liability

6.13.1. 13.1 EXCLUSIONS

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY (I) INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, OR (ii) DAMAGES FOR (a) THE COST OF PROCURING SUBSTITUTE GOODS, OR (b) LOSS OF PROFITS, REVENUES, USE, DATA OR GOODWILL ARISING OUT OF OR RELATED TO THIS AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, OR OTHERWISE, AND EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND EVEN IF A PARTY’S REMEDIES FAIL THEIR ESSENTIAL PURPOSE.

6.13.2. 13.2 DAMAGES CAP

ADDITIONALLY, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, NVIDIA’S TOTAL CUMULATIVE AGGREGATE LIABILITY FOR ANY AND ALL LIABILITIES, OBLIGATIONS OR CLAIMS ARISING OUT OF OR RELATED TO THIS AGREEMENT WILL NOT EXCEED FIVE U.S. DOLLARS (US$5).

6.14. 14. Use in Mission Critical Applications

You acknowledge that the Software provided under this Agreement is not designed or tested by NVIDIA for use in any system or application where the use or failure of such system or application

developed with NVIDIA's Software could result in injury, death or catastrophic damage (each, a "Mission Critical Application"). Examples of Mission Critical Applications include use in avionics, navigation, autonomous vehicle applications, AI solutions for automotive products, military, medical, life support or other mission-critical or life-critical applications. NVIDIA will not be liable to you or any third party, in whole or in part, for any claims or damages arising from these uses. You are solely responsible for ensuring that systems and applications developed with the Software include sufficient safety and redundancy features and comply with all applicable legal and regulatory standards and requirements.

6.15. 15. Governing Law and Jurisdiction

This Agreement will be governed in all respects by the laws of the United States and the laws of the State of Delaware, without regard to conflict of laws principles or the United Nations Convention on Contracts for the International Sale of Goods. The state and federal courts residing in Santa Clara County, California will have exclusive jurisdiction over any dispute or claim arising out of or related to this Agreement, and the parties irrevocably consent to personal jurisdiction and venue in those courts; except that either party may apply for injunctive remedies or an equivalent type of urgent legal relief in any jurisdiction.

6.16. 16. General

6.16.1. 16.1 Indemnity

By using the Software you agree to defend, indemnify and hold harmless NVIDIA and its affiliates and their respective officers, directors, employees and agents from and against any claims, disputes, demands, liabilities, damages, losses, costs and expenses arising out of or in any way connected with your use of the Software in breach of the terms of this Agreement.

6.16.2. 16.2 Independent Contractors

The parties are independent contractors, and this Agreement does not create a joint venture, partnership, agency, or other form of business association between the parties. Neither party will have the power to bind the other party or incur any obligation on its behalf without the other party's prior written consent. Nothing in this Agreement prevents either party from participating in similar arrangements with third parties.

6.16.3. 16.3 No Assignment

NVIDIA may assign, delegate or transfer its rights or obligations under this Agreement by any means or operation of law. You may not, without NVIDIA's prior written consent, assign, delegate or transfer any of your rights or obligations under this Agreement by any means or operation of law, and any attempt to do so is null and void.

6.16.4. 16.4 No Waiver

No failure or delay by a party to enforce any term or obligation of this Agreement will operate as a waiver by that party, or prevent the enforcement of such term or obligation later.

6.16.5. 16.5 Trade Compliance

You agree to comply with all applicable export, import, trade and economic sanctions laws and regulations, as amended, including without limitation U.S. Export Administration Regulations and Office of Foreign Assets Control regulations. You confirm (a) your understanding that export or reexport of certain NVIDIA products or technologies may require a license or other approval from appropriate authorities and (b) that you will not export or reexport any products or technology, directly or indirectly, without first obtaining any required license or other approval from appropriate authorities, (i) to any countries that are subject to any U.S. or local export restrictions (currently including, but not necessarily limited to, Belarus, Cuba, Iran, North Korea, Russia, Syria, the Region of Crimea, Donetsk People's Republic Region and Luhansk People's Republic Region); (ii) to any end-user who you know or have reason to know will utilize them in the design, development or production of nuclear, chemical or biological weapons, missiles, rocket systems, unmanned air vehicles capable of a maximum range of at least 300 kilometers, regardless of payload, or intended for military end-use, or any weapons of mass destruction; (iii) to any end-user who has been prohibited from participating in the U.S. or local export transactions by any governing authority; or (iv) to any known military or military-intelligence end-user or for any known military or military-intelligence end-use in accordance with U.S. trade compliance laws and regulations.

6.16.6. 16.6 Government Rights

The Software, documentation and technology ("Protected Items") are "Commercial products" as this term is defined at 48 C.F.R. 2.101, consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in, respectively, 48 C.F.R. 12.212 and 48 C.F.R. 227.7202 & 252.227-7014(a)(1). Before any Protected Items are supplied to the U.S. Government, you will (i) inform the U.S. Government in writing that the Protected Items are and must be treated as commercial computer software and commercial computer software documentation developed at private expense; (ii) inform the U.S. Government that the Protected Items are provided subject to the terms of the Agreement; and (iii) mark the Protected Items as commercial computer software and commercial computer software documentation developed at private expense. In no event will you permit the U.S. Government to acquire rights in Protected Items beyond those specified in 48 C.F.R. 52.227-19(b)(1)-(2) or 252.227-7013(c) except as expressly approved by NVIDIA in writing.

6.16.7. 16.7 Notices

Please direct your legal notices or other correspondence to legalnotices@nvidia.com with a copy mailed to NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, California 95051, United States of America, Attention: Legal Department. If NVIDIA needs to contact you, you consent to receive the notices by email and agree that such notices will satisfy any legal communication requirements.

6.16.8. 16.8 Severability

If a court of competent jurisdiction rules that a provision of this Agreement is unenforceable, that provision will be deemed modified to the extent necessary to make it enforceable and the remainder of this Agreement will continue in full force and effect.

6.16.9. 16.9 Construction

The headings in the Agreement are included solely for convenience and are not intended to affect the meaning or interpretation of the Agreement. As required by the context of the Agreement, the singular of a term includes the plural and vice versa.

6.16.10. 16.10 Amendment

Any amendment to this Agreement must be in writing and signed by authorized representatives of both parties.

6.16.11. 16.11 Entire Agreement

Regarding the subject matter of this Agreement, the parties agree that (a) this Agreement constitutes the entire and exclusive agreement between the parties and supersedes all prior and contemporaneous communications and (b) any additional or different terms or conditions, whether contained in purchase orders, order acknowledgments, invoices or otherwise, will not be binding and are null and void.

(v. October 8, 2025)

Chapter 7. Look something up

These are reference destinations rather than guided paths.

- *gRPC API reference* — the `nvidia.dcpower.v1` service and message definitions
- *`dpsctl` command reference* — every command, flag, and example
- *Topology Schema Reference* — the schemas that validate topology files
- *Troubleshooting* — operator diagnosis and escalation criteria
- *Release notes* — what changed in each release

Copyright

©2025-2026, NVIDIA Corporation