DOCA Services
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA BlueField Container Deployment Guide</td>
<td>8</td>
</tr>
<tr>
<td>NVIDIA DOCA BlueMan Service Guide</td>
<td>24</td>
</tr>
<tr>
<td>NVIDIA DOCA Firefly Service Guide</td>
<td>32</td>
</tr>
<tr>
<td>NVIDIA DOCA Flow Inspector Service Guide</td>
<td>84</td>
</tr>
<tr>
<td>NVIDIA DOCA HBN Service Guide</td>
<td>100</td>
</tr>
<tr>
<td>HBN Service Release Notes</td>
<td>103</td>
</tr>
<tr>
<td>HBN Service Deployment</td>
<td>116</td>
</tr>
<tr>
<td>HBN Service Configuration</td>
<td>135</td>
</tr>
<tr>
<td>HBN Service Troubleshooting</td>
<td>232</td>
</tr>
<tr>
<td>NVIDIA DOCA Management Service Guide</td>
<td>240</td>
</tr>
<tr>
<td>NVIDIA OpenvSwitch Acceleration (OVS in DOCA)</td>
<td>255</td>
</tr>
<tr>
<td>OVS-Kernel Hardware Acceleration</td>
<td>257</td>
</tr>
<tr>
<td>OVS-DPDK Hardware Acceleration</td>
<td>301</td>
</tr>
<tr>
<td>OVS-DOCA Hardware Acceleration</td>
<td>320</td>
</tr>
<tr>
<td>OVS Metrics</td>
<td>347</td>
</tr>
<tr>
<td>OVS Inside BlueField</td>
<td>347</td>
</tr>
<tr>
<td>NVIDIA DOCA Telemetry Service Guide</td>
<td>351</td>
</tr>
<tr>
<td>NVIDIA DOCA UROM Service Guide</td>
<td>437</td>
</tr>
</tbody>
</table>
This is an overview of the set of services provided by DOCA and their purpose.

Introduction

DOCA services are DOCA-based products, wrapped in a container for fast and easy deployment on top of the NVIDIA® BlueField® DPU. DOCA services leverage DPU capabilities to offer telemetry, time synchronization, networking solutions, and more.

Services containers can be found under the official NGC catalog, labeled under the "DOCA" and "DPU" NGC labels, as well as the built-in NVIDIA platform option ("DOCA") on the container catalog.

Note

The following services are not available in the NGC catalog:

- DOCA Management Service
- NVIDIA OpenvSwitch Acceleration (OVS in DOCA)

For information on the deployment of the services, refer to the NVIDIA BlueField Container Deployment Guide.

Development Lifecycle

DOCA-based containers consist of two main categories:

- DOCA Base Images – containerized DOCA environments for both runtime and development. Used either by developers for their development environment or in the process of containerizing a DOCA-based solution.

- DOCA Services – containerized DOCA-based products

The process of developing and containerizing a DOCA-based product is described in the following sections.
Development

Before containerizing a product, users must first design and develop it using the same process for a bare-metal deployment on the BlueField DPU.

This process consists of the steps:

1. Identifying the requirements for the DOCA-based solution.

2. Reviewing the feature set offered by the DOCA SDK libraries, as shown in detail in their respective programming guides.

3. Starting the development process by following our Developer Guide to make the best use of our provided tips and tools.

4. Testing the developed solution.

Once the developed product is mature enough, it is time to start containerizing it.

Containerization

In this process, it is recommended to make use of DOCA's provided base-images, as available on DOCA's NGC page.

Three image flavors are provided:

- **base-rt** – includes the DOCA runtime, using the most basic runtime environment required by DOCA's SDK

- **full-rt** – builds on the previous image and includes the full list of runtime packages, which are all user-mode components that can be found under the doca-runtime package

- **devel** – builds on the previous image and adds headers and development tools for developing and debugging DOCA applications. This image is particularly useful for multi-stage builds.

All images are preconfigured to use to the DOCA repository of the matching DOCA version. This means that installing an additional DOCA package as part of a Dockerfile /
within the development container can be done using the following commands:

```
apt update
apt install <package name>
```

For DOCA and CUDA environments, there are similar flavors for these images combined with CUDA's images:

- base-rt (DOCA) + base (CUDA)
- full-rt (DOCA) + runtime (CUDA)
- devel (DOCA) + devel (CUDA)

Once the containerized solution is mature enough, users may start profiling it in preparation for a production-grade deployment.

**Note**

DOCA provides base images for both the DPU and the Host. For host-related DOCA base images, please refer to the image tag suffixed with "-host".

### Profiling

As mentioned in the NVIDIA BlueField Container Deployment Guide, the current deployment model of containers on top of the DPU is based on kubelet-standalone. And more specifically, this Kubernetes-based deployment makes use of YAML files to describe the resources required by the pod such as:

- CPU
- RAM
• Huge pages

It is recommended to profile your product so as to estimate the resources it requires (under regular deployments, as well as under stress testing) so that the YAML would contain an accurate "resources" section. This allows an administrator to better understand what the requirements are for deploying your service, as well as allow the k8s infrastructure to ensure that the service is not misbehaving once deployed.

Once done, the containerized DOCA-based product is ready for the final testing rounds, after which it will be ready for deployment in production environments.

Services

Container Deployment

The NVIDIA BlueField Container Deployment Guide provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField® DPU.

DOCA BlueMan

DOCA BlueMan service runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface. This friendly, easy-to-use web dashboard acts as a one-stop shop for all the information needed to monitor the DPU.

DOCA Firefly

DOCA Firefly service provides precision time protocol (PTP) based time syncing services to the BlueField DPU. PTP is used to synchronize clocks in a network which, when used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, which is far better than what is normally obtainable with network time protocol (NTP).
DOCA Flow Inspector

DOCA Flow Inspector service allows monitoring real-time data and extraction of telemetry components which can be utilized by various services for security, big data and more.

Specific mirrored packets can be transferred to Flow Inspector for parsing and analyzing. These packets are forwarded to DTS, which gathers predefined statistics determined by various telemetry providers.

DOCA HBN

DOCA Host-Based Networking service orchestrates network connectivity of dynamically created VMs/containers on cloud servers. HBN service is a BGP router that supports EVPN extension to enable multi-tenant clouds.

At its core, HBN is the Linux networking acceleration driver of the DPU, Netlink-to-DOCA daemon which seamlessly accelerates Linux networking using DOCA hardware programming APIs.

DOCA Management Service

DOCA Management service (DMS) is a one-stop shop for the user to configure and operate NVIDIA BlueField Networking Platforms and NVIDIA ConnectX Adapters (NICs). DMS governs all scripts/tools of NVIDIA with an easy open API created by the OpenConfig community. The user can configure BlueField or ConnectX for any mode whether locally (ssh) or remotely (grpc). It makes it easy to migrate and bootstrap any customer for any NVIDIA network device.

OpenvSwitch Acceleration (OVS in DOCA)

OVS-DOCA is a virtual switch service, designed to work with NVIDIA NICs and DPUs to utilize ASAP² (Accelerated Switching and Packet Processing) technology for data-path acceleration, providing the most efficient performance and feature set due to its architecture and use of DOCA libraries.
DOCA Telemetry

DOCA Telemetry service (DTS) collects data from built-in providers and from external telemetry applications. Collected data is stored in binary format locally on the DPU and can be propagated onwards using Prometheus endpoint pulling, pushing to Fluent Bit, or using other supported providers. Exporting NetFlow packets collected using the DOCA Telemetry NetFlow API is a great example of DTS usage.

DOCA UROM

DOCA UROM service provides a framework for offloading significant portions of HPC software stack directly from the host and to the BlueField networking platform.

Info

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.
NVIDIA BlueField Container Deployment Guide

This guide provides an overview and deployment configuration of DOCA containers for NVIDIA® BlueField® DPU.

Introduction

DOCA containers allow for easy deployment of ready-made DOCA environments to the DPU, whether it is a DOCA service bundled inside a container and ready to be deployed, or a development environment already containing the desired DOCA version.

Containerized environments enable the users to decouple DOCA programs from the underlying BlueField software. Each container is pre-built with all needed libraries and configurations to match the specific DOCA version of the program at hand. One only needs to pick the desired version of the service and pull the ready-made container of that version from NVIDIA’s container catalog.
The different DOCA containers are listed on NGC, NVIDIA's container catalog, and can be found under both the "DOCA" and "DPU" labels.

**Prerequisites**

- Refer to the NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField related software
- BlueField image version required is 3.9.0 and higher

![Note]

Container deployment based on standalone Kubelet, as presented in this guide, is currently in alpha version and is subject to change in future releases.

**Container Deployment**

Deploying containers on top of the BlueField DPU requires the following setup sequence:

1. Pull the container .yaml configuration files.
2. Modify the container's .yaml configuration file.
3. Deploy the container. The image is automatically pulled from NGC.

Some of the steps only need to be performed once, while others are required before the deployment of each container.

What follows is an example of the overall setup sequence using the DOCA Firefly container as an example.
Pull Container YAML Configurations

Note

This step pulls the .yaml configurations from NGC. If you have already performed this step for other DOCA containers you may skip to the next section.

To pull the latest resource version:

1. Pull the entire resource as a *.zip file:

   wget https://api.ngc.nvidia.com/v2/resources/nvidia/doca/doca_container_configs/versions/2.8.0v1/zip -O doca_container_configs_2.8.0v1.zip

2. Unzip the resource:
More information about additional versions can be found in the NGC resource page.

**Container-specific Instructions**

Some containers require specific configuration steps for the resources used by the application running inside the container and modifications for the `.yaml` configuration file of the container itself.

Refer to the container-specific instructions listed under the container's relevant page on NGC.

**Structure of NGC Resource**

The DOCA NGC resource downloaded in section "Pull Container YAML Configurations" contains a `configs` directory under which a dedicated folder per DOCA version is located. For example, 2.0.2 will include all currently available `.yaml` configuration files for DOCA 2.0.2 containers.

```bash
doca_container_configs_2.0.2v1
    configs
    1.2.0
    ...
    2.0.2
        doca_application_recognition.yaml
        doca_blueman.yaml
        doca_devel.yaml
        doca_devel_cuda.yaml
        doca_firefly.yaml
        doca_flow_inspector.yaml
        doca_hbn.yaml
        doca_ips.yaml
        doca_ips.yaml
```
In addition, the resource also contains a `scripts` directory under which services may choose to provide additional helper-scripts and configuration files to use with their services.

The folder structure of the `scripts` directory is as follows:

```
+ doca_container_configs_2.0.2v1
  + configs
  | +-- ...
  + scripts
    + doca_firefly  <== Name of DOCA Service
    + doca_hbn     <== Name of DOCA Service
      | +-- 1.3.0
      | | +-- ...
      | +-- 1.4.0
      | | +-- ...
```

A user wishing to deploy an older version of the DOCA service would still have access to the suitable YAML file (per DOCA release under `configs`) and scripts (under the service-specific version folder which resides under `scripts`).

**Spawn Container**

Once the desired `.yaml` file is updated, simply copy the configuration file to Kubelet's input folder. Here is an example using the `doca_firefly.yaml`, corresponding to the DOCA Firefly service.

```
cp doca_firefly.yaml /etc/kubelet.d
```

Kubelet automatically pulls the container image from NGC and spawns a pod executing the container. In this example, the DOCA Firefly service starts executing right away and its printouts would be seen via the container's logs.
Review Container Deployment

When deploying a new container, it is recommended to follow this procedure to ensure successful completion of each step in the deployment:

1. View currently active pods and their IDs:

   ```
   sudo crictl pods
   ```

   ![Info]
   
   It may take up to 20 seconds for the pod to start.

When deploying a new container, search for a matching line in the command's output:

<table>
<thead>
<tr>
<th>POD ID</th>
<th>CREATED</th>
<th>STATE</th>
<th>NAME</th>
<th>NAMESPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>06bd84c07537e</td>
<td>4 seconds ago</td>
<td>Ready</td>
<td>doca-firefly-my-dpu</td>
<td>default</td>
</tr>
<tr>
<td>0</td>
<td>(default)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. If a matching line fails to appear, it is recommended to view Kubelet's logs to get more information about the error:

   ```
   sudo journalctl -u kubelet --since -5m
   ```

   Once the issue is resolved, proceed to the next steps.
3. Verify that the container image is successfully downloaded from NGC into the DPU's container registry (download time may vary based on the size of the container image):

```
sudo crictl images
```

Example output:

```
<table>
<thead>
<tr>
<th>IMAGE</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8s.gcr.io/pause</td>
<td>3.9</td>
<td>829e9de338bd5</td>
<td>268kB</td>
</tr>
<tr>
<td>nvcr.io/nvidia/doca/doca_firefly</td>
<td>1.1.0</td>
<td>134cb22f34611</td>
<td>87.4MB</td>
</tr>
</tbody>
</table>
```

4. View currently active containers and their IDs:

```
sudo crictl ps
```

Once again, find a matching line for the deployed container (boot time may vary depending on the container's image size):

```
<table>
<thead>
<tr>
<th>CONTAINER</th>
<th>IMAGE</th>
<th>CREATED</th>
<th>STATE</th>
<th>NAME</th>
<th>ATTEMPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>POD ID</td>
<td>POD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b505a05b7dc23</td>
<td>134cb22f34611</td>
<td>4 minutes ago</td>
<td>Running</td>
<td>doca-firefly</td>
<td>0</td>
</tr>
<tr>
<td>06bd84c07537e</td>
<td>doca-firefly-my-dpu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Info

For more troubleshooting information and tips, refer to the matching section in our **Troubleshooting Guide**.
5. In case of failure, to see a line matching the container, check the list of all recent container deployments:

```
sudo crictl ps -a
```

It is possible that the container encountered an error during boot and exited right away:

```
CONTAINER           IMAGE               CREATED             STATE               NAME                     ATTEMPT
POD ID              POD
de2361ec15b61       134cb22f34611     1 second ago       Exited   doca-firefly   1
4aea5f5ad9c91d       doca-firefly-my-dpu
```

6. During the container's lifetime, and for a short timespan after it exits, once can view the containers logs as were printed to the standard output:

```
sudo crictl logs <container-id>
```

In this case, the user can learn from the log that the wrong configuration was passed to the container:

```
$ sudo crictl logs de2361ec15b61
Starting DOCA Firefly - Version 1.1.0
...
Requested the following PTP interface: p10
Failed to find interface "p10". Aborting
```

⚠️ **Info**
Stop Container

The recommended way to stop a pod and its containers is as follows:

1. Delete the .yaml configuration file for Kubelet to stop the pod:

```
rm /etc/kubelet.d/<file name>.yaml
```

2. Stop the pod directly (only if it still shows "Ready"):

```
sudo crictl stopp <pod-id>
```

3. Once the pod stops, it may also be necessary to stop the container itself:

```
sudo crictl stop <container-id>
```

Troubleshooting Common Errors

This section provides a list of common errors that may be encountered when spawning a container. These account for the vast majority of deployment errors and are easy to verify first before trying to parse the Kubelet journal log.
Yaml Syntax

The syntax of the `.yaml` file is extremely sensitive and minor indentation changes may cause it to stop working. The file uses spaces (') for indentations (two per indent). Using any other number of spaces causes an undefined behavior.

Huge Pages

The container only spawns once all the required system resources are allocated on the DPU and can be reserved for the container. The most notable resource is huge pages.

1. Before deploying the container, make sure that:
   1. Huge pages are allocated as required per container.
   2. Both the amount and size of pages match the requirements precisely.

2. Once huge pages are allocated, it is recommended to restart the container service to apply the change:

   ```
   sudo systemctl restart kubelet.service
   sudo systemctl restart containerd.service
   ```

3. Once the above operations are completed successfully, the container could be deployed (YAML can be copied to `/etc/kubelet.d`).

Advanced Troubleshooting

If more troubleshooting is required, refer to the matching section in the Troubleshooting Guide.
Manual Execution from Within Container - Debugging

Note

The deployment described in this section requires an in-depth knowledge of the container's structure. As this structure might change from version to version, it is only recommended to use this deployment for debugging, and only after other debugging steps have been attempted.

Although most containers define the entrypoint.sh script as the container's ENTRYPOINT, this option is only valid for interaction-less sessions. In some debugging scenarios, it is useful to have better control of the programs executed within the container via an interactive shell session. Hence, the .yaml file supports an additional execution option.

Uncommenting (i.e., removing # from) the following 2 lines in the .yaml file causes the container to boot without spawning the container's entrypoint script.

```
# command: ["sleep"]
# args: ["infinity"]
```

In this execution mode, users can attach a shell to the spawned container:

```
crictl exec -it <container-id> /bin/bash
```

Once attached, users get a full shell session enabling them to execute internal programs directly at the scope of the container.

Air-gapped Container Deployment
Container deployment on the BlueField DPU can be done in air-gapped networks and does not require an Internet connection. As explained previously, per DOCA service container, there are 2 required components for successful deployment:

- Container image – hosted on NVIDIA’s NGC catalog
- YAML file for the container

From an infrastructure perspective, one additional module is required:

- k8s.gcr.io/pause container image

### Pulling Container for Offline Deployment

When preparing an air-gapped environment, users must pull the required container images in advance so they could be imported locally to the target machine:

```
docker pull <container-image:tag>
docker save <container-image:tag> > <name>.tar
```

The following example pulls DOCA Firefly 1.1.0-doca2.0.2:

```
docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2
docker save nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2 > firefly_v1.1.0.tar
```

1. **Note**

   Some of DOCA’s container images support multiple architectures, causing the `docker pull` command to pull the image according to the architecture of the machine on which it is invoked. Users may force the operation to pull an Arm image by passing the `--platform` flag:
Importing Container Image

After exporting the image from the container catalog, users must place the created *.tar files on the target machine on which to deploy them. The import command is as follows:

```
docker pull --platform=linux/arm64 <container-image:tag>
```

```
ctr --namespace k8s.io image import <name>.tar
```

For example, to import the firefly.tar file pulled in the previous section:

```
ctr --namespace k8s.io image import firefly_v1.1.0.tar
```

Examining the status of the operation can be done using the image inspection command:

```
crictl images
```

Built-in Infrastructure Support

The DOCA image comes pre-shipped with the k8s.gcr.io/pause image:

```
/opt/mellanox/doca/services/infrastructure/
docker_pause_3_9.tar
enable_offline_containers.sh
```
This image is imported by default during boot as part of the automatic activation of DOCA Telemetry Service (DTS).

**Note**

Importing the image independently of DTS can be done using the `enable_offline_container.sh` script located under the same directory as the image's *.tar file.

This image can also be pulled and imported manually, using the following instructions:

- **To export the image:**
  
  ```
  docker pull k8s.gcr.io/pause:3.9
  docker save k8s.gcr.io/pause:3.9 > docker_pause_3_9.tar
  ```

- **To import the image:**
  
  ```
  ctr --namespace k8s.io image import docker_pause_3_9.tar
  ```

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8s.gcr.io/pause</td>
<td>3.9</td>
<td>829e9de338bd5</td>
<td>268kB</td>
</tr>
</tbody>
</table>

**DOCA Services for Host**

A subset of the DOCA services is available for host-based deployment as well. This is indicated in those services' deployment and can also be identified by having container tags on NGC with the `-host` suffix.

In contrast to the managed DPU environment, the deployment of DOCA services on the host is based on docker. This deployment can be extended further based on the user's
own container runtime solution.

**Docker Deployment**

DOCA services for the host are deployed directly using Docker.

1. Make sure Docker is installed on your host. Run:

   ```
   docker version
   ```

   If it is not installed, visit the official [Install Docker Engine](#) webpage for installation instructions.

2. Make sure the Docker service is started. Run:

   ```
   sudo systemctl daemon-reload
   sudo systemctl start docker
   ```

3. Pull the container image directly from NGC (can also be done using the `docker run` command):

   1. Visit the NGC page of the desired container.

   2. Under the "Tags" menu, select the desired tag and click the paste icon so it is copied to the clipboard.

   3. The `docker pull` command will be as follows:

   ```
   sudo docker pull <NGC container tag here>
   ```

   For example:

   ```
   sudo docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2-host
   ```
4. Deploy the DOCA service using Docker:

1. The deployment is performed using the following command:

   ```bash
   sudo docker run --privileged --net=host -v <host directory>:<container directory> -e <env variables> -it <container tag> /entrypoint.sh
   ```

   **Info**

   For more information, refer to [Docker's official documentation](https://docs.docker.com).

2. The specific deployment command for each DOCA service is listed in their respective deployment guide.
NVIDIA DOCA BlueMan Service Guide

This guide provides instructions on how to use the DOCA BlueMan service on top of NVIDIA® BlueField® DPU.

Introduction

DOCA BlueMan runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface.

All the information that BlueMan provides is gathered from the DOCA Telemetry Service (DTS), starting from DTS version 1.11.1-doca1.5.1.

Requirements

- BlueField image version 3.9.3.1 or higher

- DTS and the DOCA Privileged Executer (DPE) daemon must be up and running
Verifying DTS Status

All the information that BlueMan provides is gathered from DTS.

Verify that the state of the DTS pod is ready:

```
$ crictl pods --name doca-telemetry-service
```

Verify that the state of the DTS container is running:

```
$ crictl ps --name doca-telemetry-service
```

Verifying DPE Status

All the information that DTS gathers for BlueMan is from the DPE daemon.

Verify that the DPE daemon is active:

```
$ systemctl is-active dpe.service
active
```

If the daemon is inactive, activate it by starting the dpe.service:

```
$ systemctl start dpe.service
```

Service Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to the NVIDIA DOCA Container Deployment Guide.
## DOCA Service on NGC

BlueMan is available on NGC, NVIDIA’s container catalog. Service-specific configuration steps and deployment instructions can be found under the service’s container page.

### Default Deployment – BlueField BSP

BlueMan service is located under `/opt/mellanox/doca/services/blueman/`.

The following is a list of the files under the BlueMan directory:

- `doca_blueman_fe_service_<version>-doca<version>_arm64.tar`
- `doca_blueman_conv_service_<version>-doca<version>_arm64.tar`
- `doca_blueman_standalone.yaml`
- `bring_up_doca_blueman_service.sh`

### Enabling BlueMan Service

#### Using Script

Run `bring_up_doca_blueman_service.sh`:

```bash
$ chmod +x /opt/mellanox/doca/services/blueman/bring_up_doca_blueman_service.sh
$ /opt/mellanox/doca/services/blueman/bring_up_doca_blueman_service.sh
```

#### Manual Procedure

1. Import images to crictl images:

```bash
$ cd /opt/mellanox/doca/services/blueman/
$ ctr --namespace k8s.io image import doca_blueman_fe_service_<version>-doca<version>_arm64.tar
```
2. Verify that the DPE daemon is active:

$ systemctl is-active dpe.service
active

If the daemon is inactive, activate it by starting the dpe.service:

$ systemctl start dpe.service

3. Copy blueman_standalone.yaml to /etc/kubelet.d/:

$ cp doca_blueman_standalone.yaml /etc/kubelet.d/

Verifying Deployment Success

1. Verify that the DPE daemon is active:

$ systemctl is-active dpe.service

2. Verify that the state of the DTS container is running:

$ crictl ps --name doca-telemetry-service
3. Verify that the state of the BlueMan service container is running:

```
$ crictl ps --name doca-blueman-fe
$ crictl ps --name doca-blueman-conv
```

Configuration

The configuration of the BlueMan back end is located under `/opt/mellanox/doca/services/telemetry/config/blueman_config.ini`. Users can interact with the `blueman_config.ini` file which contains the default range values of the Pass, Warning, and Failed categories which are used in the health page. Changing these values gets reflected in the BlueMan webpage within 60 seconds.

Example of `blueman_config.ini`:

```
;Health Cpu usages Pass, warning, Failed
[Health:CPU_Usages:Pass]
range = 0,80
[Health:CPU_Usages:Warning]
range = 80,90
[Health:CPU_Usages:Failed]
range = 90,100
```

Collected Data

- **Info**
  - General info – OS name, kernel, part number, serial number, DOCA version, driver, board ID, etc.
  - Installed packages – list of all installed packages on the DPU including their version
  - CPU info – vendor, cores, model, etc.
  - FW info – all the mlxconfig parameters with default/current/next boot data
- DPU operation mode

- Health
  - System service
  - Kernel modules
  - Dmesg
  - DOCA services
  - Port status of the PF and OOB
  - Core usage and processes running on each core
  - Memory usage
  - Disk usage
  - Temperature

- Telemetry – all telemetry counters that come from DTS according to the enabled providers displayed on tables
  - Users have the ability to build graphs of specific counters

**Connecting to BlueMan Web Interface**

To log into BlueMan, enter the IP address of the DPU's OOB interface (`http://<DPU_OOB_IP>`) to a web browser located in the same network as the DPU.

The login credentials to use are the same pair used for the SSH connection to the DPU.
Troubleshooting

For general troubleshooting, refer to the NVIDIA DOCA Troubleshooting Guide.

For container-related troubleshooting, refer to the "Troubleshooting" section in the NVIDIA DOCA Container Deployment Guide.

The following are additional troubleshooting tips for DOCA BlueMan:

- The following error message in the login page signifies a failure to connect to the DPE daemon: "The service is currently unavailable. Please check server up and running."

  1. Restart the DPE daemon:

     $ systemctl restart dpe.service

  2. Verify that DTS is up and running by following the instructions in section "Verifying DTS Status".
• If the message "Invalid Credentials" appears in the login page, verify that the username and password are the same ones used to SSH to the DPU.

• If all of the above is configured as expected and there is still some failure to log in, it is recommended to check if there are any firewall rules that block the connection.

• For other issues, check the `/var/log/syslog` and `/var/log/doCA/telemetry/blueman_service.log` log file.
NVIDIA DOCA Firefly Service Guide

This guide provides instructions on how to use the DOCA Firefly service container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Firefly Service provides precision time protocol (PTP) based time syncing services to the BlueField DPU.

PTP is a protocol used to synchronize clocks in a network. When used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, which is far better than what is normally obtainable with network time protocol (NTP). PTP support is divided between the kernel and user space. The ptp4l program implements the PTP boundary clock and ordinary clock. With hardware time stamping, it is used to synchronize the PTP hardware clock to the master clock.
Requirements

Some of the features provided by Firefly require specific BlueField DPU hardware capabilities:

- PTP – Supported by all BlueField DPUs
- PPS – Requires BlueField DPU with PPS capabilities
- SyncE - Requires converged card BlueField DPUs

Failure to run PPS due to missing hardware support will be noted in the service's output. However, the service will continue to run the timing services it can provide on the provided hardware.

Firmware Version
Firmware version must be 24.34.1002 or higher.

**BlueField BSP Version**

Supported BlueField image versions are 3.9.0 and higher.

**Embedded Mode**

**Configuring Firmware Settings on DPU for Embedded Mode**

1. Set the DPU to embedded mode (default mode):

   ```
   sudo mlxconfig -y -d 03:00.0 s INTERNAL_CPU_MODEL=1
   ```

2. Enable the real time clock (RTC):

   ```
   sudo mlxconfig -d 03:00.0 set REAL_TIME_CLOCK_ENABLE=1
   ```

3. Graceful shutdown and power cycle the DPU to apply the configuration.

4. You may check the DPU mode using the following command:

   ```
   sudo mlxconfig -d 03:00.0 q | grep INTERNAL_CPU_MODEL
   # Example output
   INTERNAL_CPU_MODEL EMBEDDED_CPU(1)
   ```

**Ensuring OVS Hardware Offload**
DOCA Firefly requires that hardware offload is activated in Open vSwitch (OVS). This is enabled by default as part of the BFB image installed on the DPU.

To verify the hardware offload configuration in OVS:

```
sudo ovs-vsctl get Open_vSwitch . other_config | grep hw-offload
# Example output
{hw-offload="true"}
```

If inactive:

1. Activate hardware offloading by running:

```
sudo ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
```

2. Restart the OVS service:

```
sudo /etc/init.d/openvswitch-switch restart
```

3. Graceful shutdown and power cycle the DPU to apply the configuration.

**Helper Scripts**

Firefly's deployment contains a script to help with the configuration steps required for the network interface in embedded mode:

- `scripts/doca_firefly/<firefly-version>/prepare_for_embedded_mode.sh`
- `scripts/doca_firefly/<firefly-version>/set_new_sf.sh`

The latest DOCA Firefly version is 1.4.0.

Both scripts are included as part of DOCA's container resource which can be downloaded according to the instructions in the NVIDIA DOCA Container Deployment Guide. For more
information about the structure of the DOCA container resource, refer to section "Structure of NGC Resource" in the deployment guide.

① Note

Due to technical limitations of the NGC resource, both scripts are provided without execute (+x) permissions. This could be resolved by running the following command:

```
chmod +x scripts/doca_firefly/<firefly-version>/*.sh
```

**prepare_for_embedded_mode.sh**

This script automates all the steps mentioned in section "Setting Up Network Interfaces for Embedded Mode" and configures a freshly installed BFB image to the settings required by DOCA Firefly.

Notes:

- The script deletes all previous OVS settings and creates a single OVS bridge that matches the definitions in section "Setting Up Network Interfaces for Embedded Mode"

- The script should only be run once when connecting to the DPU for the first time or after a power cycle

- The only manual step required after using this script is configuring the IP address for the created network interface (step 5 in section "Setting Up Network Interfaces for Embedded Mode")

- The script automatically uses port 0 (p0). Configurations for port 1 should be done manually based on the commands listed in sections "set_new_sf.sh" and "Setting Up Network Interfaces for DPU Mode".

Script arguments:

- SF number (checks if already exists)
Examples:

- Prepare OVS settings using an SF indexed 4:

  ```
  chmod +x ./*.sh
  ./prepare_for_embedded_mode.sh 4
  ```

  The script makes use of `set_new_sf.sh` as a helper script.

**set_new_sf.sh**

Creates a new trusted SF and marks it as "trusted".

Script arguments:

- PCIe address
- SF number (checks if already exists)
- MAC address (if absent, a random address is generated)

Examples:

- Create SF with number "4" over port 0 of the DPU:

  ```
  ./set_new_sf.sh 0000:03:00.0 4
  ```

- Create SF with number "5" over port 0 of the DPU and a specific MAC address:

  ```
  ./set_new_sf.sh 0000:03:00.0 5 aa:bb:cc:dd:ee:ff
  ```

- Create SF with number "4" over port 1 of the DPU:
The first two examples should work out of the box for a BlueField-2 device and create SF4 and SF5 respectively.

**Setting Up Network Interfaces for DPU Mode**

1. Create a trusted SF to be used by the service according to the Scalable Function Setup Guide.

   ![Note]
   The following instructions assume that the SF has been created using index 4.

2. Create the required OVS setting as is shown in the architecture diagram:

   ```
   $ sudo ovs-vsctl add-br uplink
   $ sudo ovs-vsctl add-port uplink p0
   $ sudo ovs-vsctl add-port uplink en3f0pf0sf4
   # This port is needed to ensure we have traffic host<->network as well
   $ sudo ovs-vsctl add-port uplink pf0hpf
   ```

3. Verify the OVS settings:

   ```
   sudo ovs-vsctl show
   Bridge uplink
   Port pf0hpf
   Interface pf0hpf
   Port en3f0pf0sf4
   ```
4. Enable TX timestamping on the SF interface (not the representor):

```bash
# tx port timestamp offloading
sudo ethtool --set-priv-flags enp3s0f0s4 tx_port_ts on
```

5. Enable the interface and set an IP address for it:

```bash
# configure ip for the interface:
sudo ifconfig enp3s0f0s4 <ip-addr> up
```

6. Configure OVS to support TX timestamping over this SF and multicast traffic in general:

```bash
# Multicast-related definitions
$ sudo ovs-vsctl set Bridge uplink mcast_snooping_enable=true
$ sudo ovs-vsctl set Bridge uplink other_config:mcast-snooping-disable-flood-unregistered=true
$ sudo ovs-vsctl set Port p0 other_config:mcast-snooping-flood=true
$ sudo ovs-vsctl set Port p0 other_config:mcast-snooping-flood-reports=true
# PTP-related definitions
$ sudo ovs-ofctl add-flow uplink in_port=en3f0pf0sf4,udp,tp_src=319,actions=output:p0
$ sudo ovs-ofctl add-flow uplink in_port=p0,udp,tp_src=319,actions=output:en3f0pf0sf4
$ sudo ovs-ofctl add-flow uplink in_port=en3f0pf0sf4,udp,tp_src=320,actions=output:p0
$ sudo ovs-ofctl add-flow uplink in_port=p0,udp,tp_src=320,actions=output:en3f0pf0sf4
```

Note
Separated Mode

Configuring Firmware Settings on DPU for Separated Mode

1. Set the BlueField mode of operation to "Separated":

   ```
sudo mlxconfig -y -d 03:00.0 s INTERNAL_CPU_MODEL=0
   ```

2. Enable RTC:

   ```
sudo mlxconfig -d 03:00.0 set REAL_TIME_CLOCK_ENABLE=1
   ```

3. Graceful shutdown and power cycle the DPU to apply the configuration.

4. You may check the BlueField's operation mode using the following command:

   ```
sudo mlxconfig -d 03:00.0 q | grep INTERNAL_CPU_MODEL
# Example output
   INTERNAL_CPU_MODEL          SEPARATED_HOST(0)
   ```

Setting Up Network Interfaces for Separated Mode
1. Make sure that that `p0` is not connected to an OVS bridge:

```
sudo ovs-vsctl show
```

2. Enable TX timestamping on the `p0` interface:

```
# TX port timestamp offloading (assuming PTP interface is p0)
sudo ethtool --set-priv-flags p0 tx_port_ts on
```

3. Enable the interface and set an IP address for it:

```
# Configure IP for the interface
sudo ifconfig p0 <ip-addr> up
```

**Host-based Deployment**

Host-based deployment requires the same configuration described under section "Separated Mode".

**Service Deployment**

**DPU Deployment**

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to NVIDIA DOCA Container Deployment Guide.

Service-specific configuration steps and deployment instructions can be found under the service's container page.
**Note**

DOCA Firefly can also be deployed on DPUs not connected to the Internet. For instructions, refer to the relevant section in the NVIDIA DOCA Container Deployment Guide.

---

**Host Deployment**

DOCA Firefly has a version adapted for host-based deployments. For more information about the deployment of DOCA containers on top of a host, refer to the NVIDIA BlueField DPU Container Deployment Guide.

The following is the docker command for deploying DOCA Firefly on the host:

```
sudo docker run --privileged --net=host -v /var/log/doca/firefly:/var/log/firefly -v /etc/firefly:/etc/firefly -e PTP_INTERFACE='eth2' -it nvcr.io/nvidia/doca/doca_firefly:1.4.0-doca2.7.0-host /entrypoint.sh
```

Where:

- Additional YAML configs may be passed as environment variables as additional `-e` key-value pairs as done with `PTP_INTERFACE` above

- The exact container tag should be the desired tag as chosen on DOCA Firefly's NGC page

---

**Configuration**

All modules within the service have configuration files that allow customizing various settings, both general and PTP-related.

**Built-In Config File**
Each profile has its own base PTP configuration file for `ptp4l`. For example, the Media profile PTP configuration file is `ptp4l-media.conf`.

The built-in PTP configuration files can be found in section "PTP Profile Default Config Files". For ease-of-use, those files are provided as part of DOCA's container resource as downloaded from NGC and are placed under Firefly's `configs` directory (`scripts/doca_firefly/<firefly version>/configs`).

**Note**

When using a built-in configuration file, Firefly uses the files as stored within the container itself in the `/etc/linuxptp` directory. The configuration files included in the NGC resource are only provided for ease of access. Modifying them does **not** impact the configuration used in practice by the container. Instead, updates to the configuration should be done as described in the following sections.

---

**Custom Config File**

Instead of using a profile's base config file, users can create a file of their own, for each of the modules.

To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

For example, to set a custom `linuxptp` config file, the user can set the parameter `PTP_CONFIG_FILE` in the YAML file:

```
- name: PTP_CONFIG_FILE
  value: my_custom_ptp.conf
```

In this example, `my_custom_ptp.conf` should be placed at `/etc/firefly/my_custom_ptp.conf.`
Overriding Specific Config File Parameters

Instead of replacing the entire config file, users may opt to override specific parameters. This can be done using the following variable syntax in the YAML file: `CONF_<TYPE>_<SECTION>_<PARAMETER_NAME>`.

- **TYPE** – either PTP, MONITOR, PHC2SYS, SYNCE, or SERVO
- **SECTION** – the section in the config file that the parameter should be placed in

If the specified section does not already exist in the config file, a new section is created unless it refers to a PTP network interface that has not been included in the `PTP_INTERFACE` YAML field.

- **PARAMETER_NAME** – the config parameter name as should be placed in the config file
For example, the following variable in the YAML file definition changes the value of the parameter `priority1` under section `global` in the PTP config file to 64.

```yaml
- name: CONF_PTP_global_priority1
  value: "64"
```

**Note**

Configuring `unicast_master_table` through the YAML file is not supported due to the structure of the table (i.e., multiple entries sharing the same key).

**Ensuring and Debugging Correctness of Config Files**

The previous sections describe 2 layers for the configuration file definitions:

- Basic configuration file – either a built-in config file or a custom config file
- Adding/overriding values to/from the YAML file

In practice, there are slightly more layers in place, and the precedence is as follows (presented in increasing order):

- Default configuration values of the PTP program (ptp4l for instance) – holds values of all available configuration options
• Your chosen configuration file – contains a subset of options
• Definitions from the YAML file – narrower subset
• Firefly mandatory values

When combining the supplied configuration file with the definitions from the YAML file, Firefly goes over those definitions and checks them against a predefined set of configuration options:

• Warning only – warns if a certain value leads to known issues in a supported deployment scenario
• Override – container-internal definitions that should not be set by the user and will be overridden by Firefly

Suitable log messages are provided in either case:

```
# Example for a warning
2023-01-31 11:55:13 - Firefly - Config - WARNING - Value "4" for definition "fault_reset_interval" will be invalid in Embedded Mode, expected a value lesser or equal to "1"
2023-01-31 11:55:13 - Firefly - Config - WARNING - Continuing with invalid value
# Example for an override
2023-01-31 11:21:00 - Firefly - Config - WARNING - Invalid value "/var/run/ptp4l2" for definition "uds_address", expected "/var/run/ptp4l"
2023-01-31 11:21:00 - Firefly - Config - INFO    - Setting definition "uds_address" value to the following: "/var/run/ptp4l"
```

At the end of this process, an updated configuration file is generated by Firefly to be used later by the various time providers. To avoid accidental modification of a user-supplied configuration file or permission issues, the finalized file is generated within the container under the /tmp directory.

For instance, if using a custom configuration file named my_custom_ptp.conf under the /etc/firefly directory on the DPU, the updated file will reside within the container at the following path: /tmp/my_custom_ptp.conf.
For troubleshooting possible issues with the configuration file, one can do one of the following:

- Connect to the container directly as is explained in the debugging finalized configuration file bullet under "Troubleshooting".

- Map the container's /tmp directory to the DPU using the built-in support in the YAML file:
  
  Before the change:

  ```yaml
  # Uncomment when debugging the finalized configuration files used - Part #1
  ##- name: debug-firefly-volume
  #  hostPath:
  #    path: /tmp/firefly
  #    type: DirectoryOrCreate
  containers:
    ...
    volumeMounts:
      - name: logs-firefly-volume
        mountPath: /var/log/firefly
      - name: conf-firefly-volume
        mountPath: /etc/firefly
  # Uncomment when debugging the finalized configuration files used - Part #2
  #- name: debug-firefly-volume
  #  mountPath: /tmp
  
  # Uncomment when debugging the finalized configuration files used - Part #1
  - name: debug-firefly-volume
    hostPath:
      path: /tmp/firefly
      type: DirectoryOrCreate
  containers:
    ...
    volumeMounts:
      - name: logs-firefly-volume
        mountPath: /var/log/firefly
      - name: conf-firefly-volume
        mountPath: /etc/firefly
  
  After the change:
DOCA Firefly Service uses the following third-party providers to provide time syncing services:

- **Linuxptp** - Version v4.2
  - PTP – PTP service, provided by the PTP4L program
  - **PHC2SYS** – OS time calibration, provided by the PHC2SYS program
- **Testptp**
  - **PPS** - PPS settings service

In addition, DOCA Firefly Service also makes use of the following NVIDIA modules:

- **SyncE**

---

**Note**

The finalized configuration file keeps the sections and config options in the same order as they appear in the original file, yet the file is stripped from spare new lines or comment lines. This should be taken into considerations when directly accessing it during a debugging session.

```yaml
mountPath: /etc/firefly
# Uncomment when debugging the finalized configuration files used - Part #2
- name: debug-firefly-volume
  mountPath: /tmp
```

---

**Description**

**Providers**

DOCA Firefly Service uses the following third-party providers to provide time syncing services:
- SYNCE – Synchronous Ethernet Deamon (synced)

- Firefly
  - MONITOR - Firefly PTP Monitor

- Firefly
  - SERVO - Firefly PTP Servo

Each of the providers can be enabled, disabled, or set to use the setting defined by the configuration profile:

- YAML setting – `<provider name>_<STATE`

- Supported values – enable, disable, defined_by_profile

**Note**

For the default profile settings per provider, refer to the table under section "Profiles".

An example YAML setting for specifically disabling the `phc2sys` provider is the following:

```
- name: PHC2SYS_STATE
  value: "disable"
```

**Note**

The `defined_by_profile` setting is only available for well-defined profiles. As such, it cannot be used when the custom profile is selected. For
Profiles

DOCA Firefly Service includes profiles which represent common use cases for the Firefly service that provide a different default configuration per profile:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Default</th>
<th>Media</th>
<th>Telco (L2)</th>
<th>Custom</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Any user that requires PTP</td>
<td>Media production s</td>
<td>Telco networks</td>
<td>Custom configuration for a dedicated user scenario</td>
</tr>
<tr>
<td>PTP</td>
<td>Enabled</td>
<td>Enabled</td>
<td>Enabled</td>
<td>No default. Enable/disable should be set by the user.</td>
</tr>
<tr>
<td>PTP profile</td>
<td>PTP default profile</td>
<td>SMPTE 2059-2</td>
<td>G.8275.1</td>
<td>Set by the user</td>
</tr>
<tr>
<td>PTP Client/Serv er 1</td>
<td>Both</td>
<td>Client-only</td>
<td>Both</td>
<td>Set by the user</td>
</tr>
<tr>
<td>PHC2SYS</td>
<td>Enabled</td>
<td>Enabled</td>
<td>Enabled</td>
<td>No default. Enable/disable should be set by the user.</td>
</tr>
<tr>
<td>PPS (in/out)</td>
<td>Enabled</td>
<td>Enabled</td>
<td>Enabled</td>
<td>No default. Enable/disable should be set by the user.</td>
</tr>
<tr>
<td>PTP Monitor</td>
<td>Disabled</td>
<td>Disabled</td>
<td>Disabled</td>
<td>No default. Enable/disable should be set by the user.</td>
</tr>
<tr>
<td>SyncE</td>
<td>Disabled</td>
<td>Disabled</td>
<td>Enabled</td>
<td>No default. Enable/disable should be set by the user.</td>
</tr>
<tr>
<td>Servo</td>
<td>Disabled</td>
<td>Disabled</td>
<td>Disabled</td>
<td>No default. Enable/disable should be set by the user.</td>
</tr>
</tbody>
</table>
1. Client-only is only relevant to a single PTP interface. If more than one PTP interface is provided in the YAML file, both modes are enabled.

**Outputs**

**Container Output**

While running, the full output of the DOCA Firefly Service container can be viewed using the following command:

```
sudo crictl logs <CONTAINER-ID>
```

Where `CONTAINER-ID` can be retrieved using the following command:

```
sudo crictl ps
```

For example, in the following output, the container ID is `8f368b98d025b`.

```
CONTAINER IMAGE CREATED STATE NAME ATTEMPT POD ID
8f368b98d025b 289809f312b4c 2 seconds ago Running doca-firefly 0
5af59511b4be4  doca-firefly-some-computer-name
```

The output of the container depends on the services supported by the hardware and enabled by configuration and the selected profile. However, note that any of the configurations runs PTP, so when DOCA FireFly is running successfully expect to see the line "Running ptp4l".

The following is an example of the expected container output when running the default profile on a DPU that supports PPS:
The following is an example of the expected container output when running the default profile on a DPU that does not support PPS:

```
2023-09-07 14:04:23 - Firefly - Init  INFO  - Starting DOCA Firefly - Version 1.4.0
2023-09-07 14:04:23 - Firefly - Init  INFO  - Selected features:
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+] PTP    - Enabled - ptp4l will be used
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+] MONITOR - Enabled - PTP Monitor will be used
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+] PHC2SYS - Enabled - phc2sys will be used
2023-09-07 14:04:23 - Firefly - Init  INFO  - [-] SyncE  - Disabled
2023-09-07 14:04:23 - Firefly - Init  INFO  - [-] SERVO - Disabled
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+ ] PPS    - Enabled - testptp will be used (if supported by hardware)
2023-09-07 14:04:23 - Firefly - Init  INFO  - Going to analyze the configuration files
2023-09-07 14:04:23 - Firefly - Init  INFO  - Requested the following PTP interface: p0
2023-09-07 14:04:23 - Firefly - Init  INFO  - Starting PPS configuration
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+ ] PPS is supported by hardware
2023-09-07 14:04:23 - Firefly - Init  INFO  - set pin function okay
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+ ] PPS in - Activated
2023-09-07 14:04:23 - Firefly - Init  INFO  - set pin function okay
2023-09-07 14:04:23 - Firefly - Init  INFO  - [+ ] PPS out - Activated
2023-09-07 14:04:23 - Firefly - Init  INFO  - name mlx5_pps0 index 0 func 1 chan 0
2023-09-07 14:04:23 - Firefly - Init  INFO  - name mlx5_pps1 index 1 func 2 chan 0
2023-09-07 14:04:23 - Firefly - Init  INFO  - periodic output request okay
2023-09-07 14:04:23 - Firefly  INFO  - Running ptp4l
2023-09-07 14:04:23 - Firefly - Init  INFO  - Running Firefly PTP Monitor
2023-09-07 14:04:23 - Firefly - Init  INFO  - Running phc2sys
```
Firefly Output

On top of the container's log, Firefly defines an additional, non-volatile log that can be found in `/var/log/doca/firefly/firefly.log`.

This file contains the same output described in section "Container Output" and is useful for debugging deployment errors should the container stop its execution.

Note

To avoid disk space issues, the `/var/log/doca/firefly/firefly.log` file only contains the log from Firefly's initialization, and not the logs of the rest of the modules (ptp4l, phc2sys, etc.) or that of the PTP monitor. The latter is still included in the container log and can be inspected using the command `sudo crictl logs <CONTAINER-ID>`.

ptp4l Output
The ptp4l output can be found in the file `/var/log/doca/firefly/ptp4l.log.

Example output:

```
ptp4l[192710.691]: rms 1 max 1 freq -114506 +/- 0 delay -15 +/- 0
ptp4l[192712.692]: rms 6 max 9 freq -114501 +/- 3 delay -15 +/- 0
ptp4l[192714.692]: rms 7 max 9 freq -114511 +/- 3 delay -13 +/- 0
ptp4l[192716.692]: rms 5 max 7 freq -114502 +/- 1 delay -13 +/- 0
ptp4l[192718.693]: rms 4 max 6 freq -114509 +/- 2 delay -13 +/- 0
ptp4l[192720.693]: rms 3 max 3 freq -114506 +/- 2 delay -13 +/- 0
ptp4l[192722.694]: rms 4 max 6 freq -114509 +/- 3 delay -12 +/- 0
ptp4l[192724.694]: rms 5 max 7 freq -114510 +/- 5 delay -12 +/- 1
ptp4l[192726.695]: rms 4 max 5 freq -114508 +/- 3 delay -11 +/- 0
ptp4l[192728.695]: rms 6 max 9 freq -114504 +/- 4 delay -11 +/- 0
```

**phc2sys Output**

The phc2sys output can be found in the file `/var/log/doca/firefly/phc2sys.log.

Example output:

```
phc2sys[1873325.928]: reconfiguring after port state change
phc2sys[1873325.928]: selecting CLOCK_REALTIME for synchronization
phc2sys[1873325.928]: selecting enp3s0f0s4 as the master clock
phc2sys[1873325.928]: CLOCK_REALTIME phc offset 1378 s2 freq -165051 delay 255
phc2sys[1873326.928]: CLOCK_REALTIME phc offset 1378 s2 freq -163673 delay 240
phc2sys[1873327.928]: port 62b785.ffe.0c9369-1 changed state
phc2sys[1873327.929]: CLOCK_REALTIME phc offset 14 s2 freq -164624 delay 255
phc2sys[1873328.936]: CLOCK_REALTIME phc offset 89 s2 freq -164545 delay 240
```

**SyncE Output**

The SyncE output can be found in the file `/var/log/doca/firefly/synced.log.

Example output:
INFO  [05/09/2023 05:11:01.493414]: SyncE Group #0: is in TRACKING holdover acquired mode on p0, frequency_diff: 0 (ppb)
INFO  [05/09/2023 05:11:02.502963]: SyncE Group #0: is in TRACKING holdover acquired mode on p0, frequency_diff: -113 (ppb)
INFO  [05/09/2023 05:11:03.512491]: SyncE Group #0: is in TRACKING holdover acquired mode on p0, frequency_diff: 37 (ppb)

### Note

The verbosity of the output from the SYNCE module is limited by default. To set the output to be more verbose, set the `verbose` option to 1 (True).

**Before:**

```
# Example #4 - Overwrite the value of verbose in the [global] section of the SyncE configuration file.
#- name: CONF_SYNCE_global_verbose
  value: "1"
```

**After:**

```
# Example #4 - Overwrite the value of verbose in the [global] section of the SyncE configuration file.
- name: CONF_SYNCE_global_verbose
  value: "1"
```

### Firefly Servo Output

The Firefly servo output can be found in the file `/var/log/doca/firefly/servo.log`. 
**Example output:**

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Service</th>
<th>Module</th>
<th>Offset</th>
<th>Frequency</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>2024-03-18 09:04:22</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+8 +/- 2</td>
<td>5.66 +/- 0.41</td>
<td>-48 +/- 2</td>
</tr>
<tr>
<td>2024-03-18 09:04:24</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+4 +/- 2</td>
<td>-6.35 +/- 0.36</td>
<td>-47 +/- 2</td>
</tr>
<tr>
<td>2024-03-18 09:04:26</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+2 +/- 2</td>
<td>-6.75 +/- 0.41</td>
<td>-47 +/- 1</td>
</tr>
<tr>
<td>2024-03-18 09:04:28</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+0 +/- 2</td>
<td>-6.97 +/- 0.35</td>
<td>-47 +/- 1</td>
</tr>
<tr>
<td>2024-03-18 09:04:30</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+0 +/- 3</td>
<td>-7.30 +/- 0.60</td>
<td>-47 +/- 1</td>
</tr>
<tr>
<td>2024-03-18 09:04:33</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+1 +/- 2</td>
<td>-6.93 +/- 0.41</td>
<td>-47 +/- 1</td>
</tr>
<tr>
<td>2024-03-18 09:04:35</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+1 +/- 2</td>
<td>-6.81 +/- 0.48</td>
<td>-47 +/- 1</td>
</tr>
<tr>
<td>2024-03-18 09:04:37</td>
<td>Firefly</td>
<td>SERVO</td>
<td>+2 +/- 2</td>
<td>-6.76 +/- 0.52</td>
<td>-48 +/- 2</td>
</tr>
</tbody>
</table>

**Tx Timestamping Support on DPU Mode**

When the BlueField is operating in DPU mode, additional OVS configuration is required as mentioned in step 6 of section "Setting Up Network Interfaces for DPU Mode". This configuration achieves the following:

- Proper support for incoming/outgoing multicast traffic
- Enabling Tx timestamping

Firefly only gets the packet timestamping for outgoing PTP messages (Tx timestamping) when they are offloaded to the hardware. As such, when working with OVS, users must ensure this traffic flow is properly recognized and offloaded. If offloading does not take place, Firefly gets stuck in a fault loop while waiting to receive the Tx timestamp events:

```
ptp4l[2912.797]: timed out while polling for tx timestamp
ptp4l[2912.797]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
ptp4l[2912.797]: port 1 (enp3s0f0s4): send sync failed
ptp4l[2923.528]: timed out while polling for tx timestamp
ptp4l[2923.528]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
ptp4l[2923.528]: port 1 (enp3s0f0s4): send sync failed
```

The solution to this issue:
• Activation of hardware offloading in OVS

• OpenFlow rules that ensure OVS properly recognizes the traffic and offloads it to the hardware

• Modification to the fault_reset_interval configuration value to ensure timely recovery from the fault induced by the first packet being always treated by software (until the rule is offloaded to hardware). As such, Firefly requires that the fault_reset_interval value is 1 or less. Proper warnings are raised if an improper value is detected. The value is updated accordingly in the built-in profiles.

When these configurations are in order, Firefly includes a report for a single fault during boot, but recovers from it and continues as usual:

```
ptp4l[3715.687]: timed out while polling for tx timestamp
ptp4l[3715.687]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
ptp4l[3715.687]: port 1 (enp3s0f0sf4): send delay request failed
```

**Troubleshooting Tx Timestamp Issues**

As explained earlier, there are several layers required to ensure Tx timestamping works as necessary by Firefly. The following is a list of commands to debug the state of each layer:

1. Inspect the OpenFlow rules:

```
$ sudo ovs-ofctl dump-flows uplink
cookie=0x0, duration=4075.576s, table=0, n_packets=2437, n_bytes=209582,
udp,in_port=en3f0pf0sf4,tp_src=319 actions=output:p0
cookie=0x0, duration=4075.549s, table=0, n_packets=1216, n_bytes=109420,
udp,in_port=p0,tp_src=319 actions=output:en3f0pf0sf4
cookie=0x0, duration=4075.521s, table=0, n_packets=13, n_bytes=1242,
udp,in_port=en3f0pf0sf4,tp_src=320 actions=output:p0
cookie=0x0, duration=4074.604s, table=0, n_packets=3034, n_bytes=297376,
udp,in_port=p0,tp_src=320 actions=output:en3f0pf0sf4
cookie=0x0, duration=4075.856s, table=0, n_packets=184, n_bytes=12901, priority=0
actions=NORMAL
```
2. Inspect hardware TC rules while DOCA Firefly is deployed (the rules age out after 10 seconds without traffic):

$ sudo tc -s -d filter show dev en3f0pf0sf4 egress
filter ingress protocol ip pref 4 flower chain 0
filter ingress protocol ip pref 4 flower chain 0 handle 0x1
  eth_type ipv4
  ip_proto udp
  src_port 320
  ip_flags nofrag
  in_hw in_hw_count 1
    action order 1: mirred (Egress Redirect to device p0) stolen
    index 3 ref 1 bind 1 installed 7 sec used 7 sec
    Action statistics:
    Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
    backlog 0b 0p requeues 0
    cookie bec8bd6ede4e86341e9045a6ed58ca2
    no_percpu

filter ingress protocol ip pref 4 flower chain 0 handle 0x2
  eth_type ipv4
  ip_proto udp
  src_port 319
  ip_flags nofrag
  in_hw in_hw_count 1
    action order 1: mirred (Egress Redirect to device p0) stolen
    index 4 ref 1 bind 1 installed 6 sec used 6 sec
    Action statistics:
    Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
    backlog 0b 0p requeues 0
    cookie c568d97efd400de98608fbbf86ccdf3c
    no_percpu

**Note**

If no TC rules are present when Firefly is running, this usually indicates that hardware offloading is disabled at the OVS level,
PTP

Firefly uses the `ptp4l` utility to handle the Precision Time Protocol (IEEE 1588).

Through the YAML file, users can configure the network interfaces used for the protocol:

```yaml
# Network interfaces to be used (For multiple interfaces use a space (" ") separated list)
- name: PTP_INTERFACE
  # Set according to used interfaces on the local setup
  value: "p0"
```

Before the deployment of the container, users should configure this field to point at the desired network interface(s) configured in the previous steps.

PHC2SYS

Firefly uses the `phc2sys` utility to synchronize the OS's clock to the accurate time stamps received by `ptp4l`.

Through the YAML file, users can configure the command-line arguments used by the `phc2sys` program:

```yaml
- name: PHC2SYS_ARGS
  value: "-a -r"
```

Firefly adds the following command-line arguments on top of the user-selected flags:
• Use of chosen configuration file (empty configuration file by default, or user-supplied file if specified in the YAML file)

• Redirection of output to a log file using the `-m` command line option

**Note**

`phc2sys` must use the same `domainNumber` setting used by `ptp4l`. If the same `domainNumber` is not set by the user, Firefly does that automatically.

**Note**

`phc2sys` is only able to accurately sync the clock of the hosting environment (usually the DPU, but may also be the host if deployed there) if other timing services, such as NTP, are disabled.

So, for instance, on Ubuntu 22.04, users must ensure that the NTP timing service is disabled by running:

```
systemctl stop systemd-timesyncd
```

**SYNCE**

Firefly uses the proprietary `synced` utility to implement the Synchronous Ethernet protocol, aimed at ensuring synchronization of the clock's frequency with the reference clock. Once achieved, both clocks are declared as "syntonized".

Through the YAML file, users can configure the network interfaces used for the protocol:
Before the deployment of the container, one should configure this field to point at the desired network interface(s) configured in the previous steps.

DOCA includes synced support for the "dpll" backend (default) which adds support for SFs and VFs. The "dpll" backend is the default backend used. If DOCA detects the system does not support it, it will automatically falls back to the "mft" backend.

**Note**

In versions older than kernel 6.8 or BlueField Platform Software 2.8.0, only PFs are supported and only using the "mft" backend.

The backend option can be explicitly set using the YAML file by uncommenting the following lines:

```yaml
# Network interfaces to be used (For multiple interfaces use a space (" ") separated list)
- name: SYNCE_INTERFACE
  # Set according to used interfaces on the local setup
  value: "p0"
```

```yaml
# Example #5 - Explicitly specify the used backend in the [global] section of the SyncE configuration file.
#- name: CONF_SYNCE_global_backend
  # Options are "mft"/"dpll". If nothing is specified in YAML, "dpll" is taken as the default value: "mft"
```
The following is an example for the OVS commands required to route the SyncE-related traffic when using a SF on top of the "dplI" backend:

```
$ sudo ovs-ofctl add-flow uplink dl_dst=01:80:c2:00:00:02,in_port=en3f0pf0sf4,actions=p0
$ sudo ovs-ofctl add-flow uplink dl_dst=01:80:c2:00:00:02,in_port=p0,actions=en3f0pf0sf4
$ sudo ovs-ofctl add-flow uplink dl_dst=01:80:c2:00:00:02,actions=controller
```

### Info

This example uses the same OVS settings used earlier in the guide:

- **uplink** – bridge name
- **en3f0pf0sf4** – SF representor
- **p0** – PF interface we are working (port 0)

If your deployment uses different values make sure to adjust the above commands accordingly.

If the kernel version does not yet support this feature, and SF/VF are used, the following error is printed:

```
... mlx5 DPLL kernel support appears to be missing
Falling back to MFT tools backend
... ```

If this error is shown, only PFs can be used, and synced falls back to using the "mft" backend.
PTP Monitor

PTP monitor periodically queries for various PTP-related information and prints it to the container's log.

The following is a sample output of this tool:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmIdentity:</td>
<td>48:B0:2D:FF:FE:5C:4D:24 (48b02d.ffe.5c4d24)</td>
</tr>
<tr>
<td>portIdentity:</td>
<td>48:B0:2D:FF:FE:5C:53:44 (48b02d.ffe.5c5344-1)</td>
</tr>
<tr>
<td>port_state:</td>
<td>Active</td>
</tr>
<tr>
<td>domainNumber:</td>
<td>2</td>
</tr>
<tr>
<td>master_offset:</td>
<td>avg: 1      max: -8  rms: 3</td>
</tr>
<tr>
<td>gmPresent:</td>
<td>true</td>
</tr>
<tr>
<td>ptp_stable:</td>
<td>Recovered</td>
</tr>
<tr>
<td>UtcOffset:</td>
<td>37</td>
</tr>
<tr>
<td>timeTraceable:</td>
<td>0</td>
</tr>
<tr>
<td>frequencyTraceable:</td>
<td>0</td>
</tr>
<tr>
<td>grandmasterPriority1:</td>
<td>128</td>
</tr>
<tr>
<td>gmClockClass:</td>
<td>248</td>
</tr>
<tr>
<td>gmClockAccuracy:</td>
<td>0x6</td>
</tr>
<tr>
<td>grandmasterPriority2:</td>
<td>128</td>
</tr>
<tr>
<td>gmOffsetScaledLogVariance:</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>ptp_time (TAI):</td>
<td>Thu Sep 7 11:22:50 2023</td>
</tr>
<tr>
<td>ptp_time (UTC adjusted):</td>
<td>Thu Sep 7 11:22:13 2023</td>
</tr>
<tr>
<td>system_time (UTC):</td>
<td>Thu Sep 7 11:22:13 2023</td>
</tr>
<tr>
<td>error_count:</td>
<td>1</td>
</tr>
<tr>
<td>last_err_time (UTC):</td>
<td>Thu Sep 7 09:55:48 2023</td>
</tr>
</tbody>
</table>

Among others, this monitoring provides the following information:

- Details about the Grandmaster the DPU is syncing with
- Current PTP timestamp
- Health information such as connection errors during execution and whether they have been recovered from

PTP monitoring is disabled by default and can be activated by replacing the disable value with the IP address for the monitor server to use:
Once activated, the information can viewed from the container using the following command:

```
sudo crictl logs --tail=20 <CONTAINER-ID>
```

It is recommended to use the following `watch` command to actively monitor the PTP state:

```
sudo watch -n 1 crictl logs --tail=20 <CONTAINER-ID>
```

When triaging deployment issues, additional logging information can be found in the monitor's developer logs: `/var/log/doca/firefly/firefly_monitor_dev.log`.

![Note]

The monitoring feature connects to ptp4l's local UDS server to query the necessary information. This is why the configuration manager prevents users from modifying the `uds_address` and `uds_ro_address` fields used by ptp4l within the container.

**Configuration**

The PTP monitor supports configuration options which are passed through a dedicated configuration file like the rest of DOCA Firefly's modules. The built-in monitor configuration file can be found in the section "PTP Monitor". For ease of use, the file is also provided as part of DOCA's container resource as downloaded from NGC.

"Firefly Modules Configuration Options" contains a complete explanation of each of the configuration options alongside their default values.
To set a custom config file, users should locate their config file in the directory `/etc/firefly` and set the config file name in DOCA Firefly's YAML file.

```
- name: MONITOR_CONFIG_FILE
  value: my_custom_monitor.conf
```

In this example, `my_custom_monitor.conf` should be placed at `/etc/firefly/my_custom_monitor.conf`.

**Time Representations (PTP Time vs System Time)**

Under most deployment scenarios, the PTP time shown by the monitor is presented according to the International Atomic Time (TAI) standard, while the system time would most commonly use the Coordinated Universal Time (UTC). Due to the differences between these time representation models, the monitor provides 2 different time readings (each marked accordingly):

```
... UtcOffset: 37 ...

ptp_time (TAI): Thu Sep 7 11:22:50 2023
ptp_time (UTC adjusted): Thu Sep 7 11:22:13 2023
system_time (UTC): Thu Sep 7 11:22:13 2023
```

This difference (37 seconds in the above example) is intentional and stems from the amount of leap seconds since epoch. This is indicated by the `UtcOffset` field that is also included in the monitor's report.

**Monitor Server**

In addition to printing the monitoring data to the container's standard output available through the container logs, the monitoring data is also exposed through a gRPC server that clients can subscribe to. This allows a monitoring client on the host to subscribe to monitor events from the service running on top of the DPU, thus providing better visibility.
The following diagram presents the recommended deployment architecture for connecting the monitoring client (on the host) to the monitor server (on the DPU).

Based on the above, when activating the monitor feature, the user must provide the IP address to be used by the monitor server:

- name: MONITOR_STATE
Users can choose to only view the monitoring events through the container logs without connecting to the monitoring server. In this case, it is recommended to configure the local host IP address (127.0.0.1) in the YAML file to avoid exposing it to an unwanted network.

**Monitor Client**

The required files for the monitor client are available under the service's dedicated NGC resource "scripts" directory.

Example command line for executing the python-based monitor client from a Linux host:

```
$ sudo pip3 install click protobuf grpcio
$ ./doca_firefly_monitor_client.py <ip-address-for-the-monitoring-server>
```

**Note**

Reference source files and the .proto file used for Firefly's monitor are placed under the `src/` within the NGC resource.

**Firefly Servo**

Firefly's Servo module can be seen as an extension to the built-in set of servos offered by linuxptp. When active, linuxptp is automatically set to “free running” and the control over the physical hardware clock (PHC) is handed over to Firefly's own servo.
The following is a sample output of this tool when using the l2-telco profile (16 messages per seconds):

```
2024-03-18 07:46:45 - Firefly - SERVO - INFO   - Detected new master clock: 48b02d.fffe.5c4d24-1
2024-03-18 07:46:45 - Firefly - SERVO - INFO   - Transition from servo state IDLE to FREE_RUNNING
2024-03-18 07:46:47 - Firefly - SERVO - INFO   - Estimated a logSyncInterval of: -4
2024-03-18 07:46:47 - Firefly - SERVO - INFO   - Measured offset 18691 delay -47
2024-03-18 07:46:48 - Firefly - SERVO - INFO   - Transition from servo state FREE_RUNNING to LOCKED
2024-03-18 07:46:50 - Firefly - SERVO - INFO   - offset +164 +/- 164 freq -1.50 +/- 0.00 delay -48 +/- 1
2024-03-18 07:46:52 - Firefly - SERVO - INFO   - Transition from servo state LOCKED to LOCKED_STABLE
2024-03-18 07:46:52 - Firefly - SERVO - INFO   - offset +0 +/- 1 freq -1.41 +/- 0.47 delay -48 +/- 1
2024-03-18 07:46:54 - Firefly - SERVO - INFO   - offset -8 +/- 4 freq -4.21 +/- 1.40 delay -47 +/- 1
2024-03-18 07:46:57 - Firefly - SERVO - INFO   - offset -12 +/- 2 freq -5.46 +/- 0.73 delay -47 +/- 1
2024-03-18 07:46:59 - Firefly - SERVO - INFO   - offset -13 +/- 2 freq -6.13 +/- 0.65 delay -47 +/- 1
2024-03-18 07:47:01 - Firefly - SERVO - INFO   - offset -13 +/- 3 freq -6.19 +/- 1.23 delay -47 +/- 2
2024-03-18 07:47:03 - Firefly - SERVO - INFO   - offset -19 +/- 2 freq -8.04 +/- 0.96 delay -47 +/- 1
2024-03-18 07:47:06 - Firefly - SERVO - INFO   - offset -14 +/- 3 freq -6.46 +/- 1.11 delay -47 +/- 1
2024-03-18 07:47:08 - Firefly - SERVO - INFO   - offset -16 +/- 2 freq -7.32 +/- 0.78 delay -48 +/- 2
2024-03-18 07:47:10 - Firefly - SERVO - INFO   - offset -15 +/- 2 freq -7.11 +/- 0.87 delay -47 +/- 2
2024-03-18 07:47:12 - Firefly - SERVO - INFO   - offset -14 +/- 1 freq -6.74 +/- 0.57 delay -47 +/- 2
2024-03-18 07:47:15 - Firefly - SERVO - INFO   - offset -12 +/- 3 freq -6.20 +/- 1.01 delay -48 +/- 1
2024-03-18 07:47:17 - Firefly - SERVO - INFO   - offset -13 +/- 2 freq -6.40 +/- 0.89 delay -47 +/- 1
2024-03-18 07:47:19 - Firefly - SERVO - INFO   - offset -11 +/- 2 freq -5.98 +/- 0.86 delay -48 +/- 1
2024-03-18 07:47:21 - Firefly - SERVO - INFO   - offset -10 +/- 2 freq -5.75 +/- 0.87 delay -46 +/- 1
2024-03-18 07:47:24 - Firefly - SERVO - INFO   - offset -8 +/- 1 freq -5.15 +/- 0.42 delay -47 +/- 1
```

As can be seen, the servo's behavior is similar to that of linuxptp's pt4l and consists of a state machine that tracks the state of the active PTP port (FREE_RUNNING, LOCKED, LOCKED_STABLE, etc).

Firefly's Servo is disabled by default (in all profiles) and can be activated by replacing the define_by_profile value with enable:

```
# Activation status
- name: SERVO_STATE
  # Options are "enable"/"disable"/"defined_by_profile"
  value: "enable"
```
Once activated, the information can be viewed from the module's log file
/var/log/doca/firefly/servo.log.

**Firefly Servo Configuration**

Firefly's Servo is currently aimed for telco-related deployments, using the l2-telco profile including the use of SyncE. As such, the default values in the built-in configuration file are optimized for those scenarios.

The servo supports configuration options which are passed through a dedicated configuration file like the rest of DOCA Firefly's modules. The built-in servo configuration file can be found in the section "Firefly Servo". For ease of use, the file is also provided as part of DOCA's container resource as downloaded from NGC.

"Firefly Modules Configuration Options" contains a complete explanation of each of the configuration options alongside their default values.

To set a custom config file, users should locate their config file in the directory /etc/firefly and set the config file name in DOCA Firefly's YAML file.

```
- name: SERVO_CONFIG_FILE
  value: my_custom_servo.conf
```

In this example, my_custom_servo.conf should be placed at /etc/firefly/my_custom_servo.conf.

**Dynamic Packet Rate Support**

The servo has the ability to dynamically detect the packet rate used by the PTP grandmaster clock, so to calibrate itself accordingly incase it differs from the recommended 16 packets per seconds.

```
2024-03-18 07:46:45 - Firefly - SERVO - INFO - Transition from servo state IDLE to FREE_RUNNING
2024-03-18 07:46:47 - Firefly - SERVO - INFO - Estimated a logSyncInterval of: -4
2024-03-18 07:46:47 - Firefly - SERVO - INFO - Measured offset 18691 delay -47
```
In a case the message rate is constant and known in advance, the dynamic estimation can be disabled, in favour of a provided message rate:

```yaml
- name: CONF_SERVO_global_servo_const_log_sync_interval
  value: "-2"
```

In the above example, a fixed message rate of 4 packets per seconds will be used (logSyncInterval of "-2").

**Note**

While the servo was tested to produce stable results with various packets rates (2, 4, 8, 16, 32, 64, 128), it is only officially recommended for use in deployments using a packet rate of 16 packets per second.

---

**VLAN Tagging**

DOCA Firefly natively supports VLAN-tagging-enabled network interfaces.

**Separated Mode**

The name of the VLAN-enabled network interface should be the one passed through the YAML file in the `PTP_INTERFACE` field.

**Embedded Mode**

In addition to passing on the VLAN-enabled interface through the YAML as listed in the previous section, the user is also required to configure the network routing within the DPU to support the VLAN tagging:
1. The following example configures a VLAN tag of 10 to the `enp3s0f0s4` interface:

```bash
$ sudo ip link add link enp3s0f0s4 name enp3s0f0s4.10 type vlan id 10
$ sudo ip link set up enp3s0f0s4.10
$ sudo ifconfig enp3s0f0s4.10 192.168.104.1 up
```

In this example, `enp3s0f0s4.10` is the interface to be passed to DOCA Firefly.

2. Additional commands to route the traffic within the DPU:

```bash
$ sudo ovs-ofctl add-flow uplink in_port=en3f0pf0sf4,dl_vlan=10,actions=output:p0
$ sudo ovs-ofctl add-flow uplink in_port=p0,dl_vlan=10,actions=output:en3f0pf0sf4
```

**Multiple Interfaces**

DOCA Firefly can support multiple network interfaces through the following YAML file syntax:

```yaml
- name: PTP_INTERFACE
  value: "<space (' ') separated list of interface names>"
```

For example:

```yaml
- name: PTP_INTERFACE
  value: "p0 p1"
```

⚠️ **Note**
The monitoring feature is supported for multiple interfaces only when the `clientOnly` configuration is enabled.

### Note

Automatic mode (`-a`) for `phc2sys` is not supported when working with multiple interfaces. It is recommended to disable `phc2sys` in this mode.

### Troubleshooting

When troubleshooting container deployment issues, it is highly recommended to follow the deployment steps and tips in the "Review Container Deployment" section of the NVIDIA DOCA Container Deployment Guide.

To debug the finalized configuration file used by Firefly, users can connect to the container as follows:

1. Open a shell session on the running container using the container ID:

   ```bash
   sudo crictl exec -it <container-id> /bin/bash
   ```

2. Once connected to the container, the finalized configuration file can be found under the `/tmp` directory using the same filename as the original configuration file.

### Info
More information regarding the configuration files can be found under section "Ensuring and Debugging Correctness of Config File".

Pod is Marked as "Ready" and No Container is Listed

Error

When deploying the container, the pod's STATE is marked as Ready, an image is listed, however no container can be seen running:

```
$ sudo crictl pods
<table>
<thead>
<tr>
<th>POD ID</th>
<th>CREATED</th>
<th>STATE</th>
<th>NAME</th>
<th>NAMESPACE</th>
<th>ATTEMPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>06bd84c07537e</td>
<td>4 seconds ago</td>
<td>Ready</td>
<td>doca-firefly-my-dpu</td>
<td>default</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
$ sudo crictl images
<table>
<thead>
<tr>
<th>IMAGE</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8s.gcr.io/pause</td>
<td>3.2</td>
<td>2a060e2e7101d</td>
<td>251kB</td>
</tr>
<tr>
<td>nvcr.io/nvidia/doca/doca_firefly</td>
<td>1.1.0-doca2.0.2</td>
<td>134cb22f34611</td>
<td>87.4MB</td>
</tr>
</tbody>
</table>
```

```
$ sudo crictl ps
<table>
<thead>
<tr>
<th>CONTAINER</th>
<th>IMAGE</th>
<th>CREATED</th>
<th>STATE</th>
<th>NAME</th>
<th>ATTEMPT</th>
<th>POD</th>
</tr>
</thead>
</table>
```

Solution

In most cases, the container did start, but immediately exited. This could be checked using the following command:

```
$ sudo crictl ps -a
<table>
<thead>
<tr>
<th>CONTAINER</th>
<th>IMAGE</th>
<th>CREATED</th>
<th>STATE</th>
<th>NAME</th>
<th>ATTEMPT</th>
<th>POD</th>
</tr>
</thead>
</table>
```
Should the container fail (i.e., state of `Exited`) it is recommended to examine Firefly's main log at `/var/log/doca/firefly/firefly.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the the container's ID:

```
$ sudo crictl logs 556bb78281e1d
Starting DOCA Firefly - Version 1.1.0
...
Requested the following PTP interface: p10
Failed to find interface "p10". Aborting
```

**Custom Config File is Not Found**

**Error**

When DOCA Firefly is deployed using a custom configuration file, a deployment error occurs and the following log message appears:

```
... 2023-09-07 14:04:23 - Firefly - Init - ERROR - Custom config file not found: my_file.conf. Aborting
...
```

**Solution**

Check the custom file name written in the YAML file and make sure that you properly placed the file with that name under the `/etc/firefly/` directory of the DPU.
Profile is Not Supported

Error

When DOCA Firefly is deployed, a deployment error occurs and the following log message appears:

... 
2023-09-07 14:04:23 - Firefly - Init  - ERROR  - profile <name> is not supported. Aborting 
... 

Solution

Verify that the profile selected in the YAML file matches one of the supported profiles as listed in the profiles table.

Note

The profile name is case sensitive. The name must be specified in lower-case letters.

PPS Capability is Missing

Error

When DOCA Firefly is deployed and configured to use the PPS module, a deployment error occurs and the following log message appears:

... 
2023-09-07 14:04:23 - Firefly - Init  - INFO  - Starting PPS configuration
...
This log indicates that the DPU hardware does not support PPS. However, PTP can still run on this hardware and you should see the line `Running ptp4l` in the container log, indicating that PTP is running successfully.

**Timed Out While Polling for Tx Timestamp**

**Error**

When the BlueField is operating in DPU mode, DOCA Firefly gets stuck in a fault loop while waiting to receive the Tx timestamp events:

```
ptp4l[2912.797]: timed out while polling for tx timestamp
ptp4l[2912.797]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
ptp4l[2912.797]: port 1 (enp3s0f0s4): send sync failed
ptp4l[2923.528]: timed out while polling for tx timestamp
ptp4l[2923.528]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
ptp4l[2923.528]: port 1 (enp3s0f0s4): send sync failed
```
Solution

DOCA Firefly's configurations were already adjusted to accommodate for Tx port timestamping. For more information about the reason for this error and for the designed recovery mechanism from it, refer to section "Tx Timestamping Support on DPU Mode".

Warning – Time Jumped Backwards

Error

When using Firefly's Servo module, the following warning log message is encountered on start:

2024-01-01 14:04:23 - Firefly - SERVO - WARNING - Clock is going to jump backwards in time - this might have a system-wide impact

Solution

This warning message indicates that the system's time jumped backwards with a value of at least one minute. This event is logged by Firefly given that such jumps might have system-wide implications. For more information, refer to section "Failed to Reserve Sandbox Name" in the NVIDIA DOCA Troubleshooting Guide.
Such jumps can only happen during Firefly's boot, before the Servo achieves initial time synchronization with the reference clock.

**PTP Profile Default Config Files**

**Media Profile**

```plaintext
# This config file contains configurations for media & entertainment alongside
# DOCA Firefly specific adjustments.
#

[global]
domainNumber 127
priority1 128
priority2 127
use_syslog 1
logging_level 6
tx_timestamp_timeout 30
hybrid_e2e 1
dscp_event 46
dscp_general 46
logAnnounceInterval -2
announceReceiptTimeout 3
logSyncInterval -3
logMinDelayReqInterval -3
delay_mechanism E2E
network_transport UDIPv4
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval 1
# Required for multiple interfaces support
boundary_clock_jbod 1
```

**Default Profile**
# This config file extends linuxptp default.cfg config file with DOCA Firefly
# specific adjustments.
#
[global]
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval 1
# Required for multiple interfaces support
boundary_clock_jbod 1

Telco (L2) Profile

# This config file extends linuxptp G.8275.1.cfg config file with DOCA Firefly
# specific adjustments.
#
[global]
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
maxStepsRemoved 255
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
G.8275.portDS.localPriority 128
ptp_dst_mac 01:80:C2:00:00:0E
network_transport L2
domainNumber 24
# Value lesser or equal to 1 is required for Embedded Mode
fault_reset_interval 1
# Required for multiple interfaces support
boundary_clock_jbod 1

Firefly Modules Configuration Options
PTP Monitor

monitor-default.conf

```
# Default values for all of Firefly's PTP monitor configuration values.
#
[global]
# General
report_interval 1000
# Debugging & Logging
doca_logging_level 50
```

Configuration Options

- `report_interval` – the time interval (in milliseconds) for when the monitor should publish a report to all defined output providers (standard output, gRPC clients, etc). Default: 1000 (1 second).

- `doca_logging_level` – Logging level for the module, based on DOCA’s logging levels. Default is 50 (INFO). Valid options:
  - 10=DISABLE
  - 20=CRITICAL
  - 30=ERROR
  - 40=WARNING
  - 50=INFO
  - 60=DEBUG
Firefly Servo

servo-default.conf

#
# Default values for all of Firefly's servo configuration values
#

[global]
# Time thresholds
offset_from_master_min_threshold  -1500
offset_from_master_max_threshold  1500
init_max_time_adjustment  0
max_time_adjustment  1500
step_adjustment_threshold  0
hold_over_timer  0
# Sampling Window & servo logic
warmup_period  1500
sync_filter_length  6
delay_request_filter_length  6
servo_adjustment_interval  4
servo_init_adjustment_interval  24
servo_const_log_sync_interval  0xFF
servo_window_min_samples  2
servo_num_offset_values  5
servo_pi_cutoff_frequency  0.0159
servo_pi_dumping_factor  7.85

# Debugging & Logging
summary_interval  2000
doca_logging_level  50
free_running  0

Configuration Options

- offset_from_master_min_threshold – Minimal threshold (in nanoseconds) for declaring time offset from the master clock as "stable". Default is -1500 (-1.5 microseconds).
- offset_from_master_max_threshold – Maximal threshold (in nanoseconds) for declaring time offset from the master clock as "stable". Default is +1500 (+1.5 microseconds).

- init_max_time_adjustment – When active, defines the maximal allowed time (step) adjustment (in nanoseconds) before the servo reaches the "locked" state. Default is 0 (disabled).

- max_time_adjustment – When active, defines the maximal allowed reference time adjustment (in nanoseconds) after the servo has reached the "locked" state. Default is 1500 (1.5 microseconds).

- step_adjustment_threshold – When active, defines the thresholds above which a time (step) adjustment (in nanoseconds) would be allowed, even after the servo has reached the "locked" state. Default is 0 (disabled).

- hold_over_timer – When active, defines the time duration (in seconds) in which the servo stays in "hold over" mode, until reverting back to "free running". Default is 0 ("hold over" state is disabled).

- warmup_period – Time span (in milliseconds) during which samples are collected to estimate the logSyncInterval value (packet rate). Default is 1500 (1.5 seconds).

- sync_filter_length – Number of SYNC messages in the servo’s history buffer. Default is 6.

- delay_request_filter_length – Number of DELAY_REQUEST messages in the servo’s history buffer. Default is 6 messages.

- servo_adjustment_interval – Number of SYNC messages after which the PHC is updated once the servo has reached the "locked" state at least once. Default is 4 messages.

- servo_init_adjustment_interval – Number of SYNC messages after which the PHC is updated before the servo has ever reached the "locked" state. Default is 24 messages.

- servo_const_log_sync_interval – Known fixed value to be used as the logSyncInterval instead of trying to estimate it at runtime. Default is 0xFF (disabled).

- servo_window_min_samples – Minimal number of samples needed for a servo calculation. Default is 2 messages.

- servo_num_offset_values – Number of consecutive timestamps within the "offset from master" threshold that are required so to transition from the "locked" state and to
the "locked stable" state. Default is 5 offset values.

- `servo_pi_cutoff_frequency` – The PI servo's cutoff frequency value. Default is 0.0159.
- `servo_pi_dumping_factor` – The PI servo's dumping factor value. Default is 7.85.
- `summary_interval` – The time interval (in milliseconds) for when the servo should publish a report log event. Default is 2000 (2 seconds).
- `doca_logging_level` – Logging level for the module, based on DOCA's logging levels. Default is 50 (INFO). Valid options:
  - 10=DISABLE
  - 20=Critical
  - 30=ERROR
  - 40=WARNING
  - 50=INFO
  - 60=DEBUG
- `free_running` – Tell the servo to only log the operations, without actually adjusting the PHC. Default is 0 (disabled).
This guide provides instructions on how to use the DOCA Flow Inspector service container on top of NVIDIA® BlueField® DPU.

**Introduction**

DOCA Flow Inspector service enables real-time data monitoring and extraction of telemetry components. These components can be leveraged by various services, including those focused on security, big data, and other purposes.

DOCA Flow Inspector service is linked to DOCA Telemetry Service (DTS). It receives mirrored packets from the user parses the data, and forwards it to the DTS, which aggregates predefined statistics from various providers and sources. The service utilizes the DOCA Telemetry Exporter API to communicate with the DTS, while the DPDK infrastructure facilitates packet acquisition at a user-space layer.

DOCA Flow Inspector operates within its dedicated Kubernetes pod on BlueField, aimed at receiving mirrored packets for analysis. The received packets are parsed and transmitted, in a predefined structure, to a telemetry collector that manages the remaining telemetry aspects.
Service Flow

The DOCA Flow Inspector receives a configuration file in a JSON format which includes which of the mirrored packets should be filtered and which information should be sent to DTS for inspection.

The configuration file can include several export units under the "export-units" attribute. Each one is comprised of a "filter" and an "export". Each packet that matches one filter (based on the protocol and ports in the L4 header) is then parsed to the corresponding requested struct defined in the export. That information only is sent for inspection. A packet that does not match any filter is dropped.

In addition, the configuration file could contain FI optional configuration flags, see JSON format and example in the Configuration section.

The service watches for changes in the JSON configuration file in runtime and for any change that reconfigures the service.

The DOCA Flow Inspector runs on top of DPDK to acquire L4. The packets are then filtered and HW-marked with their export unit index. The packets are then parsed according to their export unit and export struct, and then forwarded to the telemetry collector using IPC.
Configuration phase:

1. A JSON file is used as input to configure the export units (i.e., filters and corresponding export structs).

2. The filters are translated to HW rules on the SF (scalable function port) using the DOCA Flow library.

3. The connection to the telemetry collector is initialized and all export structures are registered to DTS.

Inspection phase:

1. Traffic is mirrored to the relevant SF.

2. Ingress traffic is received through the configured SF.
3. Non-L4 traffic and packets that do not match any filter are dropped using hardware rules.

4. Packets matching a filter are marked with the export unit index they match and are passed to the software layer in the Arm cores.

5. Packets are parsed to the desired struct by the index of export unit.

6. The telemetry information is forwarded to the telemetry agent using IPC.

7. Mirrored packets are freed.

8. If the JSON file is changed, run the configuration phase with the updated file.

**Requirements**

Before deploying the flow inspector container, ensure that the following prerequisites are satisfied:

1. Create the needed files and directories. Folders should be created automatically. Make sure the .json file resides inside the folder:

   ```
   $ touch /opt/mellanox/doca/services/flow_inspector/bin/flow_inspector_cfg.json
   $ sudo mkdir -p /opt/mellanox/doca/services/telemetry/config
   $ sudo mkdir -p /opt/mellanox/doca/services/telemetry/ipc_sockets
   $ sudo mkdir -p /opt/mellanox/doca/services/telemetry/data
   $ sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

   Validate that DTS's configuration folders exist. They should be created automatically when DTS is deployed.

2. Allocate huge pages as needed by DPDK. This requires root privileges.

   ```
   $ sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```
Or alternatively:

```
$ sudo echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
$ sudo mkdir /mnt/huge
$ sudo mount -t hugetlbfs nodev /mnt/huge
```

Deploy a scalable function according to NVIDIA BlueField DPU Scalable Function User Guide and mirror packets accordingly using the Open vSwitch command.

For example:

1. Mirror packets from p0 to sf4:

```
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl add-port ovsbr1 p0
$ ovs-vsctl add-port ovsbr1 en3f0pf0sf4
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
    -- --id=@p2 get port p0 \
    -- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-port=@p2 \
    output-port=@p1 \
    -- set bridge ovsbr1 mirrors=@m
```

2. Mirror packets from pf0hpf or p0 that pass through sf4:

```
$ ovs-vsctl add-br ovsbr1
$ ovs-vsctl add-port ovsbr1 pf0hpf
$ ovs-vsctl add-port ovsbr1 p0
$ ovs-vsctl add-port ovsbr1 en3f0pf0sf4
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
    -- --id=@p2 get port pf0hpf \
    -- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-port=@p2 \
    output-port=@p1 \
    -- set bridge ovsbr1 mirrors=@m
$ ovs-vsctl -- --id=@p1 get port en3f0pf0sf4 \
    -- --id=@p2 get port p0 \
    -- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-port=@p2 \
    output-port=@p1 \
```
The output of last command (creating the mirror) should output a sequence of letters and numbers similar to the following:

```
0d248ca8-66af-427c-b600-af1e286056e1
```

**Note**

The designated SF must be created as a trusted function. Additional details can be found in the NVIDIA BlueField DPU Scalable Function User Guide.

---

### Service Deployment

For information about the deployment of DOCA containers on top of the BlueField DPU, refer to the NVIDIA DOCA Container Deployment Guide.

DTS is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's container page.

**Note**

The order of running DTS and DOCA Flow Inspector is important. You must launch DTS, wait a few seconds, and then launch DOCA Flow Inspector.

---

### Configuration
The DOCA Flow Inspector configuration file should be placed under 
/opt/mellanox/doca/services/flow_inspector/bin/<json_file_name>.json and be built in the following format:

```json
{
   /* Optional param, time period to check for changes in JSON config file (in seconds) and flush telemetry buffer if enabled (default is 60 seconds) */
   "config-sample-rate": <time>,

   /* Optional param, telemetry buffer size in bytes (default is 60KB) */
   "telemetry-buffer-size": <size>,

   /* Optional param, enable periodic telemetry buffer flush and defining the period time (in seconds) */
   "telemetry-flush-rate": <numeric value in seconds>,

   /* Mandatory param, Flow Inspector export units */
   "export-units":
   [
      /* Export Unit 0 */
      {
         "filter":
         {
            "protocols": [<L4 protocols separated by comma>], # What L4 protocols are allowed
            "ports":
            [
               [source port>, <destination port>],
               [source ports range>, <destination ports range>],
               [... more pairs of source, dest ports]
            ]
         },
         "export":
         {
            "fields": [<fields to be part of export struct, separated by comma>] # the Telemetry event will contain these fields.
         }
      },
      [... More Export Units>
   ]
}
```
Export Unit Attributes

Allowed protocols:

- "TCP"
- "UDP"

Port range:

- It is possible to insert a range of ports for both source and destination
- Range should include borders [start_port-end_port]

Allowed ports:

- All ports in range 0-65535 as a string
- Or * to indicate any ports

Allowed fields in export struct:

- timestamp – timestamp indicating when it was received by the service
- host_ip – the IP of the host running the service
- src_mac – source MAC address
- dst_mac – destination MAC address
- src_ip – source IP
- dst_ip – destination IP
- protocol – L4 protocol
- src_port – source port
- dst_port – destination port
- **flags** – additional flags (relevant to TCP only)
- **data_len** – data payload length
- **data_short** – short version of data (payload sliced to first 64 bytes)
- **data_medium** – medium version of data (payload sliced to first 1500 bytes)
- **data_long** – long version of data (payload sliced to first 9*1024 bytes)

**JSON example:**

```json
{
   /* Optional param, time period to check for changes in JSON config file (in seconds) and flush telemetry buffer if enabled (default is 60 seconds) */
   "config-sample-rate": 30,

   /* Optional param, telemetry maximum buffer size in bytes */
   "telemetry-buffer-size": 70000,

   /* Optional param, enable periodic telemetry buffer flush and defining the period time (in seconds) */
   "telemetry-flush-rate": 1.5,

   /* Mandatory param, Flow Inspector export units */
   "export-units": [

      /* Export Unit 0 */
      {
         "filter":
         {
            "protocols": ["tcp", "udp"],
            "ports":
            [
               ["*", "433-460"],
               ["20480", "28341"],
               ["28341", "20480"],
               ["68", "67"],
               ["67", "68"]
            ]
         },
      }
   ]
}
```
"export":
{
    "fields": ["timestamp", "host_ip", "src_mac", "dst_mac", "src_ip", "dst_ip", "protocol", "src_port", "dst_port", "flags", "data_len", "data_long"]
},
/* Export Unit 1 */
{
    "filter":
    {
        "protocols": ["tcp"],
        "ports":
        [ ["5-10","422"], ["80","80"]
        ],
        "export":
        {
            "fields": ["timestamp","dst_ip", "host_ip", "data_len", "flags", "data_medium"]
        }
    }
}

**Note**

If a packet header contains L4 ports or L4 protocol which are not specified in any filter, they are filtered out.
The .yaml file downloaded from NGC can be easily edited according to your needs.

```yaml
env:
  # Set according to the local setup
  - name: SF_NUM_1
    value: "2"  # Additional EAL flags, if needed
  - name: EAL_FLAGS
    value: ""  # Service-Specific command line arguments
  - name: SERVICE_ARGS
    value: "--policy /flow_inspector/flow_inspector_cfg.json -l 60"
```

- The SF_NUM_1 value can be changed according to the SF used in the OVS configuration and can be found using the command in NVIDIA BlueField DPU Scalable Function User Guide.

- The EAL_FLAGS value must be changed according to the DPDK flags required when running the container.

- The SERVICE_ARGS are the runtime arguments received by the service:
  - `-l, --log-level <value>` – sets the (numeric) log level for the program `<10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>`
  - `-p, --policy <json_path>` – sets the JSON path inside the container

## Verifying Output

Enabling write to data in the DTS allows debugging the validity of the DOCA Flow Inspector.

To allow DTS to write locally, uncomment the following line in `/opt/mellanox/doca/services/telemetry/config/dts_config.ini`:

```ini
#output=/data
```
The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example:

```
$ tree /opt/mellanox/doca/services/telemetry/data/
    /opt/mellanox/doca/services/telemetry/data/
    {year}
    {mmd}
    {hash}
    {source_id}
    {source_tag}{timestamp}.bin
    {another_source_id}
    {another_source_tag}{timestamp}.bin
.schema
    schema_{MD5_digest}.json
```

New binary files appear when:

- The service starts
- When the binary file's max age/size restriction is reached
- When JSON file is changed and new schemas of telemetry are created
An hour passes

If no schema or no data folders are present, refer to the Troubleshooting section in NVIDIA DOCA Telemetry Service Guide.

**Note**

source_id is usually set to the machine hostname. source_tag is a line describing the collected counters, and it is often set as the provider's name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```bash
crictl exec -it <Container-ID> /opt/mellanox/collectx/bin/clx_read -s /data/schema /data/path/to/datafile.bin
```

The data written locally should be shown in the following format assuming a packet matching Export Unit 1 from the example has arrived:

```
{
  "timestamp": 1656427771076130,
  "host_ip": "10.237.69.238",
  "src_ip": "11.7.62.4",
  "dst_ip": "11.7.62.5",
  "data_len": 1152,
  "data_short": "Hello World"
}
```

Troubleshooting
When troubleshooting container deployment issues, it is highly recommended to follow the deployment steps and tips in the "Review Container Deployment" section of the NVIDIA DOCA Container Deployment Guide.

**Pod is Marked as "Ready" and No Container is Listed**

**Error**

When deploying the container, the pod's STATE is marked as Ready, an image is listed, however no container can be seen running:

```bash
$ sudo crictl pods
POD ID        CREATED             STATE       NAME                                    NAMESPACE
ATTEMPT  RUNTIME
3162b71e67677 4 seconds ago       Ready       doca-flow-inspector-my-dpu                      default
0          (default)

$ sudo crictl images
IMAGE                     TAG     IMAGE ID          SIZE
k8s.gcr.io/pause          3.2     2a060e2e7101d      487kB
nvcr.io/nvidia/doca/doca_flow_inspector 1.1.0-doca2.0.2 2af1e539eb7ab  86.8MB

$ sudo crictl ps
CONTAINER       IMAGE               CREATED             STATE       NAME                                    ATTEMPT  POD
ID    POD
556bb78281e1d  2af1e539eb7ab  6 seconds ago       Exited       doca-flow-inspector      1
3162b71e67677  doca-flow-inspector-my-dpu
```

**Solution**

In most cases, the container did start, but immediately exited. This could be checked using the following command:

```bash
$ sudo crictl ps -a
CONTAINER       IMAGE               CREATED             STATE       NAME                                    ATTEMPT  POD
ID    POD
556bb78281e1d  2af1e539eb7ab  6 seconds ago       Exited       doca-flow-inspector      1
3162b71e67677  doca-flow-inspector-my-dpu
```
Should the container fail (i.e., state of *Exited*), it is recommended to examine the Flow Inspector's main log at `/var/log/doca/flow_inspector/flow_inspector_fi_dev.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the container's ID:

```
$ sudo crictl logs 556bb78281e1d
...
2023-10-04 11:42:55 - flow_inspector - FI - ERROR - JSON file was not found <config-file-path>.
```

## Pod is Not Listed

### Error

When placing the container's YAML file in the Kubelet's input folder, the service pod is not listed in the list of pods:

```
$ sudo crictl pods

<table>
<thead>
<tr>
<th>POD ID</th>
<th>CREATED</th>
<th>STATE</th>
<th>NAME</th>
<th>NAMESPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTEMPT</td>
<td>RUNTIME</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### Solution

In most cases, the pod does not start due to the absence of the requested hugepages. This can be verified using the following command:

```
$ sudo journalctl -u kubelet -e . . .
   pod="default/doca-flow-inspector-<my-dpu>" err="preemption: error finding a set of pods to preempt: no set of running pods found to reclaim resources: [(res: hugepages-2Mi, q: 104563999874), ]"
```
NVIDIA DOCA HBN Service Guide

This guide provides instructions on how to use the DOCA HBN Service container on top of NVIDIA® BlueField® networking platform.

Introduction

Info

Beyond this page, the content of the HBN Service Guide is distributed across the following subpages:

- HBN Service Release Notes
- HBN Service Deployment
- HBN Service Configuration
- HBN Service Troubleshooting

Host-based Networking (HBN) is a DOCA service that enables the network architect to design a network purely on L3 protocols, enabling routing to run on the server-side of the network by using the BlueField as a BGP router. The EVPN extension of BGP, supported by HBN, extends the L3 underlay network to multi-tenant environments with overlay L2 and L3 isolated networks.

The HBN solution packages a set of network functions inside a container which, itself, is packaged as a service pod to be run on BlueField Arm. At the core of HBN is the Linux networking BlueField acceleration driver Netlink-to-DOCA, or nl2docad. This daemon
seamlessly accelerates Linux networking using DOCA APIs to program specific packet processing rules in BlueField hardware.

The driver mirrors the Linux kernel routing and bridging tables into the BlueField hardware tables by discovering the configured Linux networking objects using the Linux Netlink API. Dynamic network flows, as learned by the Linux kernel networking stack, are also programmed by the driver into BlueField hardware by listening to Linux kernel networking events.

The following diagram captures an overview of HBN and the interactions between various components of HBN.
ifupdown2 is the interface manager which pushes all the interface related states to kernel.

The routing stack is implemented in FRR and pushes all the control states (EVPN MACs and routes) to kernel via netlink.

Kernel maintains the whole network state and relays the information using netlink. The kernel is also involved in the punt path and handling traffic that does not match any rules in the eSwitch.

nl2docad listens for the network state via netlink and invokes the DOCA interface to accelerate the flows in BlueField hardware tables. nl2docad also offloads these flows to eSwitch.

**Service Function Chaining**

HBN is a "bump-in-the-wire" service and requires specific network configuration on BlueField called service function chaining (SFC). SFC configuration is used to redirect network traffic, which is originated from or forwarded to the host or BlueField itself via the HBN data plane.

The diagram below shows the fully detailed default configuration for HBN with SFC.
In this setup, the HBN container is configured to use sub-function ports (SFs) instead of the actual uplinks, PFs and VFs. To illustrate, for example:

- **Uplinks** – use `p0_sf` instead of `p0`
- **PF** – use `pf0hpf_sf` instead of `pf0hpf`
- **VF** – use `pf0vf0_sf` instead of `pf0vf0`

The indirection layer between the SF and the actual ports is managed via a `br-hbn` OVS bridge automatically configured when the BFB image is installed on BlueField with HBN enabled. This indirection layer allows other services to be chained to existing SFs and provide additional functionality to transit traffic.

**HBN Service Release Notes**

The following subsections provide information on HBN service new features, interoperability, known issues, and bug fixes.

**Changes and New Features**

HBN 2.3.0 offers the following new features and updates:

- TBD
Supported Platforms and Interoperability

Supported BlueField Networking Platforms

HBN 2.2.0 has been validated on the following NVIDIA BlueField Networking Platforms:

- **BlueField-2 DPUs:**
  - BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
  - BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
  - BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
  - BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL

- **BlueField-3 DPUs:**
  - BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode)/HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
  - BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode)/NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
  - BlueField-3 B3240 P-Series Dual-slot FHHL DPU; 400GbE/NDR IB (default mode); Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled

- **BlueField-3 SuperNICs:**
  - BlueField-3 B3210L E-series FHHL SuperNIC, 100GbE (default mode)/HDR100 IB, Dual port QSFP112, PCIe Gen4.0 x16, 8 Arm cores, 16GB on-board DDR,
integrated BMC, Crypto Enabled

- BlueField-3 B3220L E-Series FHHL SuperNIC, 200GbE (default mode)/NDR200 IB, Dual-port QSFP112, PCIe Gen5.0 x16, 8 Arm cores, 16GB on-board DDR, integrated BMC, Crypto Enabled

- BlueField-3 B3140L E-Series FHHL SuperNIC, 400GbE/ NDR IB (default mode), Single-port QSFP112, PCIe Gen5.0 x16, 8 Arm cores, 16GB on-board DDR, integrated BMC, Crypto Enabled

- BlueField-3 B3140H E-series HHHL SuperNIC, 400GbE (default mode)/NDR IB, Single-port QSFP112, PCIe Gen5.0 x16, 8 Arm cores, 16GB on board DDR, integrated BMC, Crypto Enabled

Note

BlueField platforms with 8GB on-board DDR memory are currently not supported with HBN.

Supported BlueField OS

HBN 2.3.0 supports DOCA 2.8.0 (BSP 4.8.0) on Ubuntu 22.04 OS.

Verified Scalability Limits

HBN 2.8.0 has been tested to sustain the following maximum scalability limits:

<table>
<thead>
<tr>
<th>Limit</th>
<th>BlueField -2</th>
<th>BlueField -3</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTEP peers</td>
<td>2k</td>
<td>2k</td>
<td>Number of BlueFields (VTEPs) within a single overlay fabric (reachable in the underlay)</td>
</tr>
<tr>
<td>Limit</td>
<td>Blue Field -2</td>
<td>Blue Field -3</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>control plane) in the fabric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 VNIs/Overlay networks per BlueField</td>
<td>20</td>
<td>20</td>
<td>Total number of L2 VNIs in the fabric for L2 VXLAN use-case assuming every interface is associated with its own VLAN + L2 VNI</td>
</tr>
<tr>
<td>L3 VNIs/Overlay networks per BlueField</td>
<td>20</td>
<td>20</td>
<td>Total number of L3 VNIs in the fabric for L3 VXLAN use-case assuming every interface is associated with its own VLAN + L2 VNI + L3 VNI + VRF</td>
</tr>
<tr>
<td>BlueFields per a single L2 VNI network</td>
<td>2k</td>
<td>2k</td>
<td>Total number of DPUs, configured with the same L2 VNI (3 real DPUs, 2000 emulated VTEPs)</td>
</tr>
<tr>
<td>BlueFields per a single L3 VNI network</td>
<td>2k</td>
<td>2k</td>
<td>Total number of DPUs, configured with the same L3 VNI (3 real DPUs, 2000 emulated VTEPs)</td>
</tr>
<tr>
<td>Maximum number of local MAC/ARP entries per BlueField</td>
<td>20</td>
<td>20</td>
<td>Max total number of MAC/ARP entries learned from the host on the DPU</td>
</tr>
<tr>
<td>Maximum number of local BGP routes per BlueField</td>
<td>200</td>
<td>200</td>
<td>Max total number of BGP routes advertised by the host to the BlueField (BGP peering with the host): 100 IPv4 + 100 IPv6</td>
</tr>
<tr>
<td>Maximum number of remote L3 LPM routes (underlay)</td>
<td>2K</td>
<td>2K</td>
<td>IPv4 or IPv6 underlay LPM routes per BlueField (default + host routes + LPM)</td>
</tr>
<tr>
<td>Maximum number of EVPN type-2 entries</td>
<td>16K</td>
<td>16K</td>
<td>Remote overlay MAC/IP entries for compute peers stored on a single BlueField (L2 EVPN use case)</td>
</tr>
<tr>
<td>Maximum number of EVPN type-5 entries</td>
<td>16K</td>
<td>16K</td>
<td>Remote overlay L3 LPM entries for compute peers stored on a single BlueField (L3 EVPN use case)</td>
</tr>
<tr>
<td>Maximum number of PFs on the Host side</td>
<td>2</td>
<td>2</td>
<td>Total number of PFs visible to the host</td>
</tr>
<tr>
<td>Maximum number of VFs on the Host side</td>
<td>16</td>
<td>16</td>
<td>Total number of VFs created on the host</td>
</tr>
<tr>
<td>Maximum number of SFs on BlueField side</td>
<td>2</td>
<td>2</td>
<td>Total number of SF devices created on BlueField Arm</td>
</tr>
</tbody>
</table>
Known Issues

The following table lists the known issues and limitations for this release of HBN.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
<th>Workaround</th>
<th>Keyword</th>
<th>Reported in HBN version</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004191</td>
<td>Description: Due to security fixes on BlueField-2, the number of context switches increased by 20% which may result in user applications (e.g., nl2doca) running slower.</td>
<td>N/A</td>
<td>BlueField-2; performance</td>
<td>2.3.0</td>
</tr>
<tr>
<td>3769309</td>
<td>Description: A ping or other IP connectivity from a locally connected host in vrf-X to an interface IP address on the DPU/HBN itself in vrf-Y will not work, even if VRF route-leaking is enabled between these two VRFs.</td>
<td>N/A</td>
<td>IP</td>
<td>2.2.0</td>
</tr>
<tr>
<td>3886379</td>
<td>Description: Deleting and re-adding SR-IOV ports might result in some ports in br-hbn bridge going in error state.</td>
<td>If possible, an appropriate number of SR-IOV ports should be chosen at BFB install time. But if a change is made and if the system has this error, the host must undergo a power cycle to resolve the issue.</td>
<td>Bridge; SR-IOV</td>
<td>2.2.0</td>
</tr>
<tr>
<td>3835295</td>
<td>Description: Traffic entering HBN service on a host PF/VF main-interface and exiting on a sub-interface of the same PF/VF (and vice versa) is not hardware offloaded. Similarly, traffic entering HBN service on one sub-interface and exiting on another sub-interface of the same host PF/VF is also not hardware offloaded.</td>
<td>N/A</td>
<td>Hardware offload; interfaces</td>
<td>2.2.0</td>
</tr>
<tr>
<td>3772</td>
<td>Description: The DHCP relay gateway-interface IP address does not automatically pick up the IP address assigned to the associated VRF.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>Description</td>
<td>Workaround</td>
<td>Keyword</td>
<td>Reported in HBN version</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------------</td>
<td>---------</td>
<td>------------------------</td>
</tr>
<tr>
<td>55</td>
<td>Workaround: The gateway-interface IP address must be explicitly configured.</td>
<td></td>
<td>DHCP relay gateway; IP</td>
<td>2.2.0</td>
</tr>
<tr>
<td>52</td>
<td><strong>Keyword:</strong> DHCP relay gateway; IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Description: If NVUE-based routing policy (route map) configuration is used to associated route target extended communities with a EVPN route, only one route target can be specified.</td>
<td>Workaround: N/A</td>
<td>NVUE; route target</td>
<td>2.2.0</td>
</tr>
<tr>
<td>91</td>
<td><strong>Keyword:</strong> NVUE; route target</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Reported in HBN version: 2.2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><strong>Description:</strong> When the HBN container is coming up and applying a large configuration through the NVUE-startup service which includes entities used by DHCP relay (e.g., interfaces, SVIs and VRFs), the DHCP relay service may go into FATAL state. It can be observed using the following command:</td>
<td>Workaround: Restart the DHCP relay service which is in FATAL state using the command:</td>
<td>DHCP relay; fatal; container; restart</td>
<td>2.1.0</td>
</tr>
<tr>
<td>37</td>
<td>supervisorctl status</td>
<td>supervisorctl restart &lt;relay-service-name&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>grep isc-dhcp-relay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>isc-dhcp-relay-vrf11 RUNNING pid 2069, uptime 0:11:31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>isc-dhcp-relay-vrf12 RUNNING pid 2071, uptime 0:11:31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>isc-dhcp-relay-vrf13 FATAL Exited too quickly (process log may have details)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>isc-dhcp-relay-vrf14 FATAL Exited too quickly (process log may have details)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td><strong>Description:</strong> When the DPU boots up after issuing a &quot;reboot&quot; command from the DPU itself, some host-side interfaces may remain down.</td>
<td>Workaround:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1. Restart openibd:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>systemctl restart openibd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Recreate SR-IOV interfaces if they are needed.
3. Replay interface config. For example:
   ○ If using ifupdown2:

     ifreload -a

   ○ If using Netplan:

     netplan apply

Keyword: Reboot
Reported in HBN version: 1.5.0

<table>
<thead>
<tr>
<th>35</th>
<th>47</th>
<th>10</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: IPv6 stateless ACLs are not supported.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: IPv6 ACL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reported in HBN version: 1.5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>33</th>
<th>39</th>
<th>30</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Statistics for hardware-offloaded traffic are not reflected on SFs inside an HBN container.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: Look up the stats using <code>ip -s link show</code> on PFs outside of the HBN container. PFs would show Tx/Rx stats for traffic that is hardware-accelerated in the HBN container.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: Statistics; container</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reported in HBN version: 1.4.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>33</th>
<th>52</th>
<th>00</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: NVUE show, config, and apply commands malfunction if the nvued and nvued-startup services are not in the RUNNING and EXITED states respectively.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: NVUE commands</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reported in HBN version: 1.3.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>84</th>
<th>74</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: The command <code>nv show interface &lt;intf&gt; acl</code> does not show correct information if there are multiple ACLs bound to the interface.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: Use the command <code>nv show interface &lt;intf&gt;</code> to view the ACLs bound to an interface.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: ACLs</td>
<td>Reported in HBN version: 1.2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description: Deleting an NVUE user by removing their password file and restarting the <code>decrypt-user-add</code> service on the HBN container does not work.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: Either respawn the container after deleting the file, or delete the password file corresponding to the user by running <code>userdel -r username</code>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: User deletion</td>
<td>Reported in HBN version: 1.2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description: When a packet is encapsulated with a VXLAN header, it adds extra bytes which may cause the packet to exceed the MTU of link. Typically, the packet would be fragmented but its silently dropped and no fragmentation happens.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: Make sure that the MTU on the uplink port is always 50 bytes more than host ports so that even after adding VXLAN headers, ingress packets do not exceed the MTU.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: MTU; VXLAN</td>
<td>Reported in HBN version: 1.2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description: On VXLAN encapsulation, the DF flag is not propagated to the outer header. Such a packet may be truncated when forwarded in the kernel, and it may be dropped when hardware offloaded.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: Make sure that the MTU on the uplink port is always 50 bytes more than host ports so that even after adding VXLAN headers, ingress packets do not exceed the MTU.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: VXLAN</td>
<td>Reported in HBN version: 1.2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description: When stopping the container using the command <code>crictl stop</code> an error may be reported because the command uses a timeout of 0 which is not enough to stop all the processes in the HBN container.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workaround: Pass a timeout value when stopping the HBN container by running:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>crictl stop --timeout 60 &lt;hbn-container&gt;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyword: Timeout</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reported in HBN version: 1.2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Description:</strong> The same ACL rule cannot be applied in both the inbound and outbound direction on a port.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Workaround:</strong> N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Keyword:</strong> ACLs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reported in HBN version: 1.2.0</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reported in HBN version: 1.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> The system's time zone cannot be modified using NVUE in the HBN container.</td>
</tr>
<tr>
<td><strong>Workaround:</strong> The timezone can be manually changed by symlinking the <code>/etc/localtime</code> file to a binary time zone's identifier in the <code>/usr/share/zoneinfo</code> directory. For example:</td>
</tr>
<tr>
<td>sudo ln -sf /usr/share/zoneinfo/GMT /etc/localtime</td>
</tr>
<tr>
<td><strong>Keyword:</strong> Time zone; NVUE</td>
</tr>
<tr>
<td><strong>Reported in HBN version: 1.2.0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reported in HBN version: 1.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Auto-BGP functionality (where the ASN does not need to be configured but is dynamically inferred by the system based on the system's role as a leaf or spine device) is not supported on HBN.</td>
</tr>
<tr>
<td><strong>Workaround:</strong> If BGP is configured and used on HBN, the BGP ASN must be manually configured.</td>
</tr>
<tr>
<td><strong>Keyword:</strong> BGP</td>
</tr>
<tr>
<td><strong>Reported in HBN version: 1.2.0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reported in HBN version: 1.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Since checksum calculation is offloaded to the hardware (not done by the kernel), it is expected to see an incorrect checksum in the tcpdump for locally generated, outgoing packets. BGP keepalives and updates are some of the packets that show such incorrect checksum in tcpdump.</td>
</tr>
<tr>
<td><strong>Workaround:</strong> N/A</td>
</tr>
<tr>
<td><strong>Keyword:</strong> BGP</td>
</tr>
<tr>
<td><strong>Reported in HBN version: 1.2.0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reported in HBN version: 1.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> MAC addresses are not learned in the hardware but only in software. This may affect performance in pure L2 unicast traffic.</td>
</tr>
<tr>
<td><strong>Reported in HBN version: 1.2.0</strong></td>
</tr>
</tbody>
</table>
**Description:** Due to disabled backend foundation units, some NVUE commands return 500 INTERNAL SERVER ERROR/404 NOT FOUND. These commands are related to features or subsystems which are not supported on HBN.

**Workaround:** N/A

**Keyword:** MAC; L2

**Reported in HBN version:** 1.3.0

---

**Description:** NetworkManager and other services not directly related to HBN may display the following message in syslog:

```
"netlink: read: too many netlink events. Need to resynchronize platform cache"
```

The message has no functional impact and may be ignored.

**Workaround:** N/A

**Keyword:** Error

**Reported in HBN version:** 1.3.0

---

**Bug Fixes**

The following table lists the known issues which have been fixed for this release of HBN.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
<th>Fixed in HBN version: 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>38 73 50 6</td>
<td>Rarely, after deletion then creation of an interface, BGP peering over that interface may announce IPv6 routes with an IPv4-mapped IPv6 address as the next hop, which the BGP peer device at the other end can reject.</td>
<td>2.3.0</td>
</tr>
<tr>
<td>33 60 69 9</td>
<td>If it is required to decrease the default MTU on interfaces on which HBN operates, after the change is made on the BlueField as well as within HBN, the BlueField must be rebooted for the change to take effect properly.</td>
<td>2.3.0</td>
</tr>
<tr>
<td>35 38 16 7</td>
<td>An explicit restart of FRR service may be required if the BGP AS number is changed via NVUE.</td>
<td>2.3.0</td>
</tr>
<tr>
<td>38 64 08 0</td>
<td>When an interface is toggled off and on, its sub-interfaces lose their IPv6 addresses and do not get them back.</td>
<td>2.3.0</td>
</tr>
<tr>
<td>36 32 34 4</td>
<td>HBN interfaces on the BlueField side (outside the HBN container) may not get their proper MTU set from systemd-network.</td>
<td>2.2.0</td>
</tr>
<tr>
<td>37 60 86 9</td>
<td>Datapath flow with very low PPS may be deleted before aging time (60 sec) in large scale of number of routes (16K+).</td>
<td>2.2.0</td>
</tr>
<tr>
<td>37 70 99 2</td>
<td>It is not possible to configure an IPv6 default (::/0) static route using NVUE.</td>
<td>2.2.0</td>
</tr>
<tr>
<td>38 24</td>
<td>When the number of unique ECMP groups used is more than 6, it results in failure of programming prefixes using ECMP-groups greater than 6.</td>
<td>2.2.0</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Fixed in HBN version:</td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>88</td>
<td>Uniqueness is based on ECMP content, so if multiple routes have same next-hop paths, they just use 1 ECMP group.</td>
<td>2.2.0</td>
</tr>
<tr>
<td>89</td>
<td>Description: In an EVPN Symmetric Routing scenario, IPv6 traffic is not hardware offloaded.</td>
<td>2.2.0</td>
</tr>
<tr>
<td>37</td>
<td>Description: If many interfaces are participating in EVPN/routing, it is possible for the routing process to run out of memory.</td>
<td>2.1.0</td>
</tr>
<tr>
<td>31</td>
<td>Description: The DOCA HBN container takes about 1 minute longer to spawn, as compared to previous HBN release (1.4.0)</td>
<td>2.1.0</td>
</tr>
<tr>
<td>32</td>
<td>Description: TC rules are programmed by OVS to map uplink and host representor ports to HBN service. These rules are ageable and can result in packets needing to get software forwarded periodically to refresh the rules.</td>
<td>2.1.0</td>
</tr>
<tr>
<td>36</td>
<td>Description: The output of the command <code>nv show interface</code> does not display information about VRFs, VXLAN, and bridge.</td>
<td>2.0.0</td>
</tr>
<tr>
<td>34</td>
<td>Description: IPv6 OOB connectivity from the HBN container stops working if the br-mgmt interface on the DPU goes down. When going down, the br-mgmt interface loses its IPv6 address, which is used as the gateway address for the HBN container. If the br-mgmt interface comes back up, its IPv6 address is not added back and IPv6 OOB connectivity from the HBN container will not work</td>
<td>1.5.0</td>
</tr>
<tr>
<td>31</td>
<td>Description: ECMP selection for the underlay path uses the ingress port and identifies uplink ports via round robin. This may not result in uniform spread of the traffic.</td>
<td>1.4.0</td>
</tr>
</tbody>
</table>
Description: When reloading (ifreload) an empty /etc/network/interfaces file, the previously created interfaces are not deleted.
Fixed in HBN version: 1.4.0

Description: When an ACL is configured for IPv4 and L4 parameters (protocol tcp/udp, source, and destination ports) match, the ACL also matches IPv6 traffic with the specified L4 parameters.
Fixed in HBN version: 1.4.0

Description: Some DPUs experience an issue with the clock settings after installing a BlueField OS in an HBN setting in which the date reverts back to "Thu Sep 8, 2022".
Fixed in HBN version: 1.4.0

Description: If interfaces on which BGP unnumbered peering is configured are not defined in the /etc/network/interfaces configuration file, BGP peering does not get established on them.
Fixed in HBN version: 1.4.0

HBN Service Deployment

HBN Service Requirements

Info

Refer to the "HBN Service Release Notes" page for information on the specific hardware and software requirements for HBN.

The following subsections describe specific prerequisites for the BlueField before deploying the DOCA HBN Service.

Enabling BlueField DPU Mode
HBN requires BlueField to work in either DPU mode or zero-trust mode of operation. Information about configuring BlueField modes of operation can be found under "NVIDIA BlueField Modes of Operation".

### Enabling SFC

HBN requires SFC configuration to be activated on the BlueField before running the HBN service container. SFC allows for additional services/containers to be chained to HBN and provides additional data manipulation capabilities.

The following subsections provide additional information about SFC and instructions on enabling it during BlueField DOCA image installation.

### Deploying BlueField DOCA Image with SFC from Host

For DOCA image installation on BlueField, the user should follow the instructions under *NVIDIA DOCA Installation Guide for Linux* with the following extra notes to enable BlueField for HBN setup:

1. Make sure link type is set to ETH under the "Installing Software on Host" section.

2. Add the following parameters to the `bf.cfg` configuration file:
   1. Enable HBN specific OVS bridge on BlueField Arm by setting `ENABLE_BR_HBN=yes`.
   2. Define the uplink ports to be used by HBN `BR_HBN_UPLINKS='<port>'`.
   3. Include PF and VF ports to be used by HBN. The following example sets both PFs and 8 VFs on each uplink: `BR_HBN_REPS='pf0hpf,pf1hpf,pf0vf0-pf0vf7,pf1vf0-pf1vf7'`.

   **Note**
   
   Must include both ports (i.e., `p0,p1`) for dual-port BlueField devices and only `p0` for single-port BlueField devices.

   3. Include PF and VF ports to be used by HBN. The following example sets both PFs and 8 VFs on each uplink: `BR_HBN_REPS='pf0hpf,pf1hpf,pf0vf0-pf0vf7,pf1vf0-pf1vf7'`.
4. (Optional) Include SF devices to be created and connected to HBN bridge on the BlueField Arm side by setting `BR_HBN_SFS='pf0dpu1,pf0dpu3'`.

![Info]

*Info*

If nothing is provided, `pf0dpu1` and `pf0dpu3` are created by default.

![Warning]

*Warning*

While older formats of `bf.cfg` still work in this release, they will be deprecated over the next 2 releases. So, it's advisable to move to the new format to avoid any upgrade issues in future releases. The following is an example for the old `bf.cfg` format:

```plaintext
ENABLE_SFC_HBN=yes
NUM_VFs_PHYS_PORT0=12 # <num VFs supported by HBN on Physical Port 0> (valid range: 0-127) Default 14
NUM_VFs_PHYS_PORT1=2  # <num VFs supported by HBN on Physical Port 1> (valid range: 0-127) Default 0
```

3. Then run:

```plaintext
bfb-install -c bf.cfg -r rshim0 -b <BFB-image>
```

**Deploying BlueField DOCA Image with SFC Using PXE Boot**

To enable HBN SFC using a PXE installation environment with BFB content, use the following configuration for PXE:
The kickstart script (bash) should include the following lines:

```
bfnet=<IFNAME>:<IPADDR>:<NETMASK> or <IFNAME>:dhcp
bfks=<URL of the kickstart script>
```

The kickstart script should include the following lines:

```
cat >> /etc/bf.cfg << EOF
ENABLE_BR_HBN=yes
BR_HBN_UPLINKS='p0,p1'
BR_HBN_REPS='pf0hpf,pf1hpf,pf0vf0-pf0vf7,pf1vf0-pf1vf7'
BR_HBN_SFS='pf0dpu1,pf0dpu3'
EOF
```

The `/etc/bf.cfg` generated above is sourced by the BFB `install.sh` script.

**Note**

It is recommended to verify the accuracy of the BlueField's clock post-installation. This can be done using the following command:

```
$ date
```

Please refer to the known issues listed in the "NVIDIA DOCA Release Notes" for more information.

**Deploying HBN with Other Services**

When the HBN container is deployed by itself, BlueField Arm is configured with 3k huge pages. If it is deployed with other services, the actual number of huge-pages must be adjusted based on the requirements of those services. For example, SNAP or NVMesh
need approximately 1k huge pages. So if HBN is running with either of these services on the same BlueField, the total number of huge pages must be set to 4k (3k for HBN and 1k for SNAP or NVMe).

To do that, add the following parameters to the $bf.cfg$ configuration file alongside other desired parameters.

```
HUGEPAGE_COUNT=4096
```

⚠️  **Warning**

This should be performed only on a BlueField-3 running with 32G of memory. Doing this on 16G system may cause memory issues for various applications on BlueField Arm.

## Launching HBN Service

### HBN Service Container Deployment

HBN service is available on NGC, NVIDIA’s container catalog. For information about the deployment of DOCA containers on top of the BlueField, refer to NVIDIA DOCA Container Deployment Guide.

### Downloading DOCA Container Resource File

Pull the latest DOCA container resource as a *.zip file from NGC and extract it to the <resource> folder (doca_container_configs_2.7.0v1 in this example):

```
wget https://api.ngc.nvidia.com/v2/resources/nvidia/doca/doca_container_configs/versions/2.7.0v1/zip -O doca_container_configs_2.7.0v1.zip
```
Running HBN Preparation Script

The HBN script (hbn-dpu-setup.sh) performs the following steps on BlueField Arm which are required for HBN service to run:

1. Sets the BlueField to DPU mode if needed.
3. Sets up interface MTU if needed.
4. Sets up mount points between BlueField Arm and HBN container for logs and configuration persistency.
5. Sets up various paths as needed by supervisord and other services inside container.

The script is located in <resource>/scripts/doca_hbn/<hbn_version>/ folder, which is downloaded as part of the DOCA Container Resource.

**Note**

To achieve the desired configuration on HBN's first boot, before running preparation script, users can update default NVUE or flat (network interfaces and FRR) configuration files, which are located in <resource>/scripts/doca_hbn/<hbn_version>/.

- For NVUE-based configuration:
  - etc/nvue.d/startup.yaml
- For flat-files based configuration:
  - etc/network/interfaces
  - etc/frr/frr.conf
Run the following commands to execute the hbn-dpu-setup.sh script:

```bash
cd <resource>/scripts/doca_hbn/2.2.0/
chmod +x hbn-dpu-setup.sh
sudo ./hbn-dpu-setup.sh
```

**Note**

After running the script, perform BlueField system-level reset.

### Spawning HBN Container

HBN container .yaml configuration is called doca_hbn.yaml and it is located in `<resource>/configs/<doca_version>/` directory. To spawn the HBN container, simply copy the doca_hbn.yaml file to the `/etc/kubelet.d` directory:

```bash
cd <resource>/configs/2.7.0/
sudo cp doca_hbn.yaml /etc/kubelet.d/
```

Kubelet automatically pulls the container image from NGC and spawns a pod executing the container. The DOCA HBN Service starts executing right away.

### Verifying HBN Container is Running

To inspect the HBN container and verify if it is running correctly:
1. Check HBN pod and container status and logs:

   1. Examine the currently active pods and their IDs (it may take up to 20 seconds for the pod to start):

      ```
      sudo crictl pods
      ```

   2. View currently active containers and their IDs:

      ```
      sudo crictl ps
      ```

   3. Examine logs of a given container:

      ```
      sudo crictl logs
      ```

   4. Examine kubelet logs if something did not work as expected:

      ```
      sudo journalctl -u kubelet@mgmt
      ```

2. Log into the HBN container:

   ```
   sudo crictl exec -it $(crictl ps | grep hbn | awk '{print $1;}') bash
   ```

3. While logged into HBN container, verify that the `frr`, `nl2doca`, and `neighmgr` services are running:

   ```
   (hbn-container)$ supervisorctl status frr
   (hbn-container)$ supervisorctl status nl2doca
   (hbn-container)$ supervisorctl status neighmgr
   ```
4. Users may also examine various logs under `/var/log` inside the HBN container.

**HBN Default Deployment Configuration**

The HBN service comes with four types of configurable interfaces:

- Two uplinks (`p0_sf`, `p1_sf`)
- Two PF port representors (`pf0hpf_sf`, `pf1hpf_sf`)
- User-defined number of VFs (i.e., `pf0vf0_sf`, `pf0vf1_sf`, ..., `pf1vf0_sf`, `pf1vf1_sf`, ...)
- Two interfaces to connect to services running on BlueField, outside of the HBN container (`pf0dpu1_sf` and `pf0dpu3_sf`)

The *_sf suffix indicates that these are sub-functions and are different from the physical uplinks (i.e., PFs, VFs). They can be viewed as virtual interfaces from a virtualized BlueField.

Each of these interfaces is connected outside the HBN container to the corresponding physical interface, see section "Service Function Chaining" (SFC) for more details.

The HBN container runs as an isolated namespace and does not see any interfaces outside the container (`oob_net0`, `real uplinks and PFs, *_sf_r representors`).
pf0dpu1_sf and pf0dpu3_sf are special interfaces for HBN to connect to services running on BlueField. Their counterparts pf0dpu0_sf and pf0dpu2_sf respectively are located outside the HBN container. See section "Connecting to DOCA Services to HBN on BlueField Arm" for deployment considerations when using the pf0dpu1.sf or pf0dpu3.sf interface in HBN.

eth0 is equivalent to the oob_net0 interface in the HBN container. It is part of the management VRF of the container. It is not configurable via NVUE and does not need any configuration from the user. See section "MGMT VRF Inside HBN Container" for more details on this interface and the management VRF.

HBN Deployment Considerations
SF Interface State Tracking

When HBN is deployed with SFC, the interface state of the following network devices is propagated to their corresponding SFs:

- Uplinks – p0, p1
- PFs – pf0hpf, pf1hpf
- VFs – pf0vfX, pf1vfX where X is the VF number

For example, if the p0 uplink cable gets disconnected:

- p0 transitions to DOWN state with NO-CARRIER (default behavior on Linux); and
- p0 state is propagated to p0_sf whose state also becomes DOWN with NO-CARRIER

After p0 connection is reestablished:

- p0 transitions to UP state; and
- p0 state is propagated to p0_sf whose state becomes UP

Interface state propagation only happens in the uplink/PF/VF-to-SF direction.

A daemon called sfc-state-propagation runs on BlueField, outside of the HBN container, to sync the state. The daemon listens to netlink notifications for interfaces and transfers the state to SFs.

SF Interface MTU

In the HBN container, all the interfaces MTU are set to 9216 by default. MTU of specific interfaces can be overwritten using flat-files configuration or NVUE.

On BlueField side (i.e., outside of the HBN container), the MTU of the uplinks, PFs and VFs interfaces are also set to 9216. This can be changed by modifying `/etc/systemd/network/30-hbn-mtu.network` or by adding a new configuration file in the `/etc/systemd/network` for specific directories.

To reload this configuration, execute `systemctl restart systemd-networkd`. 
Connecting to DOCA Services to HBN on BlueField Arm

There are various SF ports (named pf0dpuX_sf, where X is [0..n]) on BlueField Arm, which can be used to run any services on BlueField and use HBN to provide network connectivity. These ports are always created and connected in pairs of even and odd numbered ports, where even numbered ports are on BlueField side and odd numbered port are on the HBN side. For example, pf0dpu0_sf can be used by another service running on BlueField Arm to connect to HBN port pf0dpu1_sf.

Traffic between BlueField and the outside world is hardware-accelerated when the HBN side port is an L3 interface or access-port using switch virtual interface (SVI). So, it is treated the same way as PF or VF ports from a traffic handling standpoint.

**Info**

There are 2 SF port pairs created by default on BlueField Arm side so there can be 2 separate DOCA services running at same time.

Disabling BlueField Uplinks

The uplink ports must be always kept administratively up for proper operation of HBN. Otherwise, the NVIDIA® ConnectX® firmware would bring down the corresponding representor port which would cause data forwarding to stop.

**Note**

Change in operational status of uplink (e.g., carrier down) would result in traffic being switched to the other uplink.
When using ECMP failover on the two uplink SFs, locally disabling one uplink does not result in traffic switching to the second uplink. Disabling local link in this case means to set one uplink admin DOWN directly on BlueField.

To test ECMP failover scenarios correctly, the uplink must be disabled from its remote counterpart (i.e., execute admin DOWN on the remote system’s link which is connected to the uplink).

**HBN NVUE User Credentials**

The preconfigured default user credentials are as follows:

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvidia</td>
<td>nvidia</td>
</tr>
</tbody>
</table>

NVUE user credentials can be added post installation:

1. This can be done by specifying additional `--username` and `--password` to the HBN startup script (refer to "Running HBN Preparation Script"). For example:

   ```bash
   sudo ./hbn-dpu-setup.sh -u newuser -p newpassword
   ```

2. After executing this script, respawn the container or start the `decrypt-user-add` script inside running HBN container:

   ```bash
   supervisorctl start decrypt-user-add
   decrypt-user-add: started
   ```

The script creates a new user in the HBN container:

```bash
cat /etc/passwd | grep newuser
newuser:x:1001:1001::/home/newuser:/bin/bash
```
HBN NVUE Interface Classification

<table>
<thead>
<tr>
<th>Interface</th>
<th>Interface Type</th>
<th>NVUE Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0_sf</td>
<td>Uplink representor</td>
<td>swp</td>
</tr>
<tr>
<td>p1_sf</td>
<td>Uplink representor</td>
<td>swp</td>
</tr>
<tr>
<td>lo</td>
<td>Loopback</td>
<td>loopback</td>
</tr>
<tr>
<td>pf0hpf_sf</td>
<td>Host representor</td>
<td>swp</td>
</tr>
<tr>
<td>pf1hpf_sf</td>
<td>Host representor</td>
<td>swp</td>
</tr>
<tr>
<td>pf0vfx_sf</td>
<td>VF representor</td>
<td>swp</td>
</tr>
<tr>
<td>pf1vfx_sf</td>
<td>VF representor</td>
<td>swp</td>
</tr>
</tbody>
</table>

HBN Files Persistence

The following directories are mounted from BlueField Arm to the HBN container namespace and are persistent across HBN service restarts and BlueField reboots:

<table>
<thead>
<tr>
<th></th>
<th>BlueField Arm Mount Point</th>
<th>HBN Container Mount Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration file</td>
<td>/var/lib/hbn/etc/network/</td>
<td>/etc/network/</td>
</tr>
<tr>
<td>Configuration file</td>
<td>/var/lib/hbn/etc/frr/</td>
<td>/etc/frr/</td>
</tr>
<tr>
<td>Configuration file</td>
<td>/var/lib/hbn/etc/nvue.d/</td>
<td>/etc/nvue.d/</td>
</tr>
<tr>
<td>Configuration file</td>
<td>/var/lib/hbn/etc/supervisor/conf.d/</td>
<td>/etc/supervisor/conf.d/</td>
</tr>
<tr>
<td>Support and log file</td>
<td>/var/lib/hbn/var/lib/nvue/</td>
<td>/var/lib/nvue/</td>
</tr>
<tr>
<td>Support and log file</td>
<td>/var/lib/hbn/var/support/</td>
<td>/var/support/</td>
</tr>
<tr>
<td>Support and log file</td>
<td>/var/log/doca/hbn/</td>
<td>/var/log/hbn/</td>
</tr>
</tbody>
</table>

SR-IOV Support in HBN
Creating SR-IOV VFs on Host

The first step to use SR-IOV is to create Virtual Functions (VFs) on the host server.

VFs can be created using the following command:

```
sudo echo N > /sys/class/net/<host-rep>/device/sriov_numvfs
```

Where:

- `<host-rep>` is one of the two host representors (e.g., `ens1f0` or `ens1f1`)
- \(0 \leq N \leq 16\) is the desired total number of VFs
  - Set \(N=0\) to delete all the VFs on \(0 \leq N \leq 16\)
  - \(N=16\) is the maximum number of VFs supported on HBN across all representors

Automatic Creation of VF Representors and SF Devices on BlueField

VFs created on the host must have corresponding VF representor devices and SF devices for HBN on BlueField side. For example:

- `ens1f0vf0` is the first SR-IOV VF device from the first host representor; this interface is created on the host server
- `pf0vf0` is the corresponding VF representor device to `ens1f0vf0`; this device is present on the BlueField Arm side and automatically created at the same time as `ens1f0vf0` is created by the user on the host side
- `pf0vf0_sf` is the corresponding SF device for `pf0vf0` which is used to connect the VF to HBN pipeline

The creation of the SF device for VFs is done ahead of time when provisioning the BlueField and installing the DOCA image on it, see section "Enabling SFC" to see how to select how many SFs to create ahead of time.
The SF devices for VFs (i.e., pfXvfY) are pre-mapped to work with the corresponding VF representors when these are created with the command from the previous step.

**Management VRF**

Two management VRFs are automatically configured for HBN when BlueField is deployed with SFC:

- The first management VRF is outside the HBN container on BlueField. This VRF provides separation between out-of-band (OOB) traffic (via oob_net0 or tmfifo_net0) and data-plane traffic via uplinks and PFs.

- The second management VRF is inside the HBN container and provides similar separation. The OOB traffic (via eth0) is isolated from the traffic via the *_sf interfaces.

**MGMT VRF on BlueField Arm**

The management (mgmt) VRF is enabled by default when the BlueField is deployed with SFC (see section "Enabling SFC"). The mgmt VRF provides separation between the OOB management network and the in-band data plane network.

The uplinks and PFs/VFs use the default routing table while the oob_net0 (OOB Ethernet port) and the tmifo_net0 netdevices use the mgmt VRF to route their packets.

When logging in either via SSH or the console, the shell is by default in mgmt VRF context. This is indicated by a mgmt added to the shell prompt:

```bash
root@bf2:mgmt:/home/ubuntu#
root@bf2:mgmt:/home/ubuntu# ip vrf identify
mgmt.
```

When logging into the HBN container with crictl, the HBN shell will be in the default VRF. Users must switch to MGMT VRF manually if OOB access is required. Use `ip vrf exec` to do so.
The user must run `ip vrf exec mgmt` to perform operations requiring OOB access (e.g., `apt-get update`).

Network devices belonging to the mgmt VRF can be listed with the `vrf` utility:

```
root@bf2:mgmt:/home/ubuntu# vrf link list

VRF: mgmt
----------------------
tmfifo_net0      UP             00:1a:ca:ff:ff:03 <BROADCAST,MULTICAST,UP,LOWER_UP>
oob_net0         UP             08:c0:eb:c0:5a:32 <BROADCAST,MULTICAST,UP,LOWER_UP>
```

```
root@bf2:mgmt:/home/ubuntu# vrf help
vrf <OPTS>

VRF domains:
  vrf list

Links associated with VRF domains:
  vrf link list [vrf-name]

Tasks and VRF domain association:
  vrf task exec [vrf-name] <command>
  vrf task list [vrf-name]
  vrf task identify <pid>

  NOTE: This command affects only AF_INET and AF_INET6 sockets opened by the
        command that gets exec'ed. Specifically, it has *no* impact on netlink
        sockets (e.g., `ip command`).
```

To show the routing table for the default VRF, run:

```
root@bf2:mgmt:/home/ubuntu# ip route show
```

To show the routing table for the mgmt VRF, run:
MGMT VRF Inside HBN Container

Inside the HBN container, a separate mgmt VRF is present. Similar commands as those listed under section "MGMT VRF on BlueField Arm" can be used to query management routes.

The *_sf interfaces use the default routing table while the eth0 (OOB) uses the mgmt VRF to route out-of-band packets out of the container. The OOB traffic gets NATed through the oob_net0 interface on BlueField Arm, ultimately using the BlueField OOB's IP address.

When logging into the HBN container via crictl, the shell enters the default VRF context by default. Switching to the mgmt VRF can be done using the command `ip vrf exec mgmt <cmd>`.

Existing Services in MGMT VRF on BlueField Arm

On the BlueField Arm, outside the HBN container, a set of existing services run in the mgmt VRF context as they need OOB network access:

- containerd
- kubelet
- ssh
- docker

These services can be restarted and queried for their status using the command `systemctl` while adding `@mgmt` to the original service name. For example:

- To restart containerd:
  
  ```bash
  root@bf2:mgmt:/home/ubuntu# systemctl restart containerd@mgmt
  ```
To query containerd status:

```
root@bf2:mgmt:/home/ubuntu# systemctl status containerd@mgmt
```

**Note**

The original version of these services (without @mgmt) are not used and must not be started.

---

**Running New Service in MGMT VRF on BlueField Arm**

If a service needs OOB access to run, it can be added to the set of services running in mgmt VRF context. Adding such a service is only possible on the BlueField Arm (i.e., outside the HBN container).

To add a service to the set of mgmt VRF services:

1. Add it to `/etc/vrf/systemd.conf` (if it is not present already). For example, NTP is already listed in this file.

2. Run the following:

```
root@bf2:mgmt:/home/ubuntu# systemctl daemon-reload
```

3. Stop and disable to the non-VRF version of the service to be able to start the mgmt VRF one:

```
root@bf2:mgmt:/home/ubuntu# systemctl stop ntp
root@bf2:mgmt:/home/ubuntu# systemctl disable ntp
root@bf2:mgmt:/home/ubuntu# systemctl enable ntp@mgmt
```
HBN Service Configuration

To start configuring HBN, log into the HBN container:

```
root@bf2:mgmt:/home/ubuntu# systemctl start ntp@mgmt
```

```
sudo crictl exec -it $(crictl ps | grep hbn | awk '{print $1;}') bash
```

**General Network Configuration**

**Flat Files Configuration**

Add network interfaces and FRR configuration files to HBN to achieve the desired configuration:

- `/etc/network/interfaces`

- `/etc/frr/frr.conf; /etc/frr/daemons`

**Note**

Refer to NVIDIA® Cumulus® Linux documentation for more information.
NVUE Configuration

This section assumes familiarity with NVIDIA user experience (NVUE) Cumulus Linux documentation. The following subsections, only expand on HBN-specific aspects of NVUE.

NVUE Service

HBN installs NVUE by default and enables NVUE service at boot.

NVUE REST API

HBN enables REST API by default.

Users may run the cURL commands from the command line. Use the default HBN username nvidia and password nvidia.

To change the default password of the nvidia user or add additional users for NVUE access, refer to section "HBN NVUE User Credentials".

REST API example:

curl -u 'nvidia:nvidia' --insecure https://<mgmt_ip>:8765/nvue_v1/vrf/default/router/bgp
{
    "configured-neighbors": 2,
    "established-neighbors": 2,
    "router-id": "10.10.10.201"
}
**Note**

For information about using the NVUE REST API, refer to the [NVUE API documentation](#).

---

**NVUE CLI**

For information about using the NVUE CLI, refer to the [NVUE CLI documentation](#).

---

**NVUE Startup Configuration File**

When the network configuration is saved using NVUE, HBN writes the configuration to the `/etc/nvue.d/startup.yaml` file.

Startup configuration is applied by following the supervisor daemon at boot time. `nvued-startup` will appear in `EXITED` state after applying the startup configuration.

```
# supervisorctl status nvued-startup
nvued-startup         EXITED   Apr 17 10:04 AM
```

**Note**

`nv config apply startup` applies the yaml configuration saved at `/etc/nvue.d/`.

---

**Note**

For more information, refer to the [NVUE CLI documentation](#).
HBN Configuration Examples

HBN Default Configuration

After a fresh HBN installation, the default `/etc/network/interfaces` file would contain only the declaration of the two uplink SFs and a loopback interface.

```bash
source /etc/network/interfaces.d/*.intf

auto lo
iface lo inet loopback

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf
```

FRR configuration files would also be present under `/etc/frr/` but no configuration would be enabled.

Layer-3 Routing

Native Routing with BGP and ECMP

HBN supports unicast routing with BGP and ECMP for IPv4 and IPv6 traffic. ECMP is achieved by distributing traffic using hash calculation based on the source IP, destination IP, and protocol type of the IP header.
**Info**

For TCP and UDP packets, it also includes source port and destination port.

---

**ECMP Example**

ECMP is implemented any time routes have multiple paths over uplinks or host ports. For example, 20.20.20.0/24 has 2 paths using both uplinks, so a path is selected based on a hash of the IP headers.

```
20.20.20.0/24 proto bgp metric 20
   nexthop via 169.254.0.1 dev p0_sf weight 1 onlink <<<<< via uplink p0_sf
   nexthop via 169.254.0.1 dev p1_sf weight 1 onlink <<<<< via uplink p1_sf
```

---

**Info**

HBN supports up to 16 paths for ECMP.

---

**Sample NVUE Configuration for Native Routing**

```
nv set interface lo ip address 10.10.10.1/32
nv set interface lo ip address 2010:10:10::1/128
nv set interface vlan100 type svi
nv set interface vlan100 vlan 100
nv set interface vlan100 base-interface br_default
nv set interface vlan100 ip address 2030:30:30::1/64
nv set interface vlan100 ip address 30.30.30.1/24
nv set bridge domain br_default vlan 100
```
Sample Flat Files Configuration for Native Routing

Example /etc/network/interfaces configuration:

```plaintext
auto lo
iface lo inet loopback
    address 10.10.10.1/32
    address 2010:10:10::1/128

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf

auto pf0hpf_sf
iface pf0hpf_sf
    bridge-access 100

auto pf1hpf_sf
iface pf1hpf_sf
    bridge-access 100
```

nv set interface pf0hpf_sf,pf1hpf_sf bridge domain br_default access 100
nv set vrf default router bgp router-id 10.10.10.1
nv set vrf default router bgp autonomous-system 65501
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf default router bgp neighbor p0_sf remote-as external
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p0_sf address-family ipv4-unicast enable on
nv set vrf default router bgp neighbor p0_sf address-family ipv6-unicast enable on
nv set vrf default router bgp neighbor p1_sf remote-as external
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf address-family ipv4-unicast enable on
nv set vrf default router bgp neighbor p1_sf address-family ipv6-unicast enable on
auto vlan100
iface vlan100
    address 2030:30:30::1/64
    address 30.30.30.1/24
    vlan-raw-device br_default
    vlan-id 100

auto br_default
iface br_default
    bridge-ports pf0hpf_sf pf1hpf_sf
    bridge-vlan-aware yes
    bridge-vids 100
    bridge-pvid 1

Example /etc/frr/daemons configuration:

bgpd=yes
vtysh_enable=yes

FRR Config file @ /etc/frr/frr.conf -
!frr version 7.5+cl5.3.0u0
frr defaults datacenter
hostname BLUEFIELD2
log syslog informational
no zebra nexthop kernel enable
!
router bgp 65501
    bgp router-id 10.10.10.1
    bgp bestpath as-path multipath-relax
    neighbor p0_sf interface remote-as external
    neighbor p0_sf advertisement-interval 0
    neighbor p0_sf timers 3 9
    neighbor p0_sf timers connect 10
    neighbor p1_sf interface remote-as external
    neighbor p1_sf advertisement-interval 0
    neighbor p1_sf timers 3 9
    neighbor p1_sf timers connect 10
!
address-family ipv4 unicast
    redistribute connected
Direct Routing on Host-facing Interfaces

Host-facing interfaces (PFs and VFs) are not restricted to be part of the bridge for routing. HBN supports L3-only configuration with direct routing on host-facing PFs and VFs.

Sample NVUE Configuration

```
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
address-family ipv6 unicast
redistribute connected
neighbor p0_sf activate
neighbor p1_sf activate
maximum-paths 64
maximum-paths ibgp 64
exit-address-family
!
line vty
!
end
```

```
nv set interface pf0hpf_sf ip address 30.30.11.1/24
nv set interface pf0hpf_sf ip address 2030:30:11::1/64
nv set interface pf0vf0_sf ip address 30.30.13.1/24
nv set interface pf0vf0_sf ip address 2030:30:13::1/64
```

Sample Flat File Configuration

```
auto pf0hpf_sf
iface pf0hpf_sf
   address 2030:30:11::1/64
   address 30.30.11.1/24

auto pf0vf0_sf
```

DOCA Services
BGP Peering with the Host

HBN supports the ability to establish a BGP session between the host and the HBN service running on BlueField Arm and allow the host to announce arbitrary route prefixes through the BlueField into the underlay fabric. The host can use any standard BGP protocol stack implementation to establish BGP peering with HBN.

Traffic to and from endpoints on the host gets offloaded.

Note

Both IPv4 and IPv6 unicast AFI/SAFI are supported.

It is possible to apply route filtering for these prefixes to limit the potential security impact in this configuration.

Sample NVUE Configuration for Host BGP Peering

The following code block shows configuration to peer to host at 45.3.0.4 and 2001:cafe:1ead::4. The BGP session can be established using IPv4 or IPv6 address.

Note

Either of these sessions can support IPv4 unicast and IPv6 unicast AFI/SAFI.
NVUE configuration for peering with host:

```plaintext
dv set vrf default router bgp autonomous-system 63642
dv set vrf default router bgp enable on
dv set vrf default router bgp neighbor 45.3.0.4 nexthop-connected-check off
dv set vrf default router bgp neighbor 45.3.0.4 peer-group dpu_host
dv set vrf default router bgp neighbor 45.3.0.4 type numbered
dv set vrf default router bgp neighbor 2001:cafe:1ead::4 nexthop-connected-check off
dv set vrf default router bgp neighbor 2001:cafe:1ead::4 peer-group dpu_host
dv set vrf default router bgp neighbor 2001:cafe:1ead::4 type numbered
dv set vrf default router bgp peer-group dpu_host address-family ipv4-unicast enable on
dv set vrf default router bgp peer-group dpu_host address-family ipv6-unicast enable on
dv set vrf default router bgp peer-group dpu_host remote-as external
```

**Sample Flat Files Configuration for Host BGP peering**

The following block shows configuration to peer to host at 45.3.0.4 and 2001:cafe:1ead::4. The BGP session can be established using IPv4 or IPv6 address.

`frr.conf` file:

```plaintext
router bgp 63642
  bgp router-id 27.0.0.4
  bgp bestpath as-path multipath-relax
  neighbor dpu_host peer-group
  neighbor dpu_host remote-as external
  neighbor dpu_host advertisement-interval 0
  neighbor dpu_host timers 3 9
  neighbor dpu_host timers connect 10
  neighbor dpu_host disable-connected-check
  neighbor fabric peer-group
  neighbor fabric remote-as external
  neighbor fabric advertisement-interval 0
  neighbor fabric timers 3 9
  neighbor fabric timers connect 10
  neighbor 45.3.0.4 peer-group dpu_host
  neighbor 2001:cafe:1ead::4 peer-group dpu_host
  neighbor p0_sf interface peer-group fabric
  neighbor p1_sf interface peer-group fabric
```
Sample FRR configuration on the Host

Any BGP implementation can be used on the host to peer to HBN and advertise endpoints. The following is an example using FRR BGP:

Sample FRR configuration on the host:

```
!  address-family ipv4 unicast
      neighbor dpu_host activate
!
address-family ipv6 unicast
      neighbor dpu_host activate

bf2-s12# sh run
Building configuration...

Current configuration:
!
  frr version 7.2.1
  frr defaults traditional
  hostname bf2-s12
  no ip forwarding
  no ipv6 forwarding
!
  router bgp 1000008
    !
  router bgp 1000008 vrf v_200_2000
    neighbor 45.3.0.2 remote-as external
    neighbor 2001:cafe:1ead::2 remote-as external
    !
    address-family ipv4 unicast
      redistribute connected
      exit-address-family
    !
  address-family ipv6 unicast
      redistribute connected
      neighbor 45.3.0.2 activate
      neighbor 2001:cafe:1ead::2 activate
      exit-address-family
```
Sample interfaces configuration on the host:

```bash
root@bf2-s12:/home/cumulus# ifquery -a
auto lo
if lo inet loopback
    address 27.0.0.7/32
    address 2001:c000:10ff:f00d::7/128
auto v_200_2000
if v_200_2000
    address 60.1.0.1
    address 60.1.0.2
    address 60.1.0.3
    address 2001:60:1::1
    address 2001:60:1::2
    address 2001:60:1::3
    vrf-table auto
auto ens1f0np0
if ens1f0np0
    address 45.3.0.4/24
    address 2001:cafe:1ead::4/64
    gateway 45.3.0.1
    gateway 2001:cafe:1ead::1
    vrf v_200_2000
    hwaddress 00:03:00:08:00:12
    mtu 9162
```

**VRF Route Leaking**

VRFs are typically used when multiple independent routing and forwarding tables are desirable. However, users may want to reach destinations in one VRF from another VRF, as in the following cases:
- To make a service, such as a firewall available to multiple VRFs

- To enable routing to external networks or the Internet for multiple VRFs, where the external network itself is reachable through a specific VRF

Route leaking can be used to reach remote destinations as well as directly connected destinations in another VRF. Multiple VRFs can import routes from a single source VRF, and a VRF can import routes from multiple source VRFs. This can be used when a single VRF provides connectivity to external networks or a shared service for other VRFs. It is possible to control the routes leaked dynamically across VRFs with a route map.

When route leaking is used:

- The redistribute command (not network command) must be used in BGP to leak non-BGP routes (connected or static routes)

- It is not possible to leak routes between the default and non-default VRF

**Note**

Ping or other IP traffic from a locally connected host in vrfX to a local interface IP address on the BlueField/HBN in vrfY does not work, even if VRF route-leaking is enabled between these two VRFs.

In the following example commands, routes in the BGP routing table of VRF BLUE dynamically leak into VRF RED:

```
nv set vrf RED router bgp address-family ipv4-unicast route-import from-vrf list BLUE
nv config apply
```

The following example commands delete leaked routes from VRF BLUE to VRF RED:

```
nv unset vrf RED router bgp address-family ipv4-unicast route-import from-vrf list BLUE
nv config apply
```
To exclude certain prefixes from the import process, configure the prefixes in a route map.

The following example configures a route map to match the source protocol BGP and imports the routes from VRF BLUE to VRF RED. For the imported routes, the community is 11:11 in VRF RED.

```plaintext
nv set vrf RED router bgp address-family ipv4-unicast route-import from-vrf list BLUE
nv set router policy route-map BLUEtoRED rule 10 match type ipv4
nv set router policy route-map BLUEtoRED rule 10 match source-protocol bgp
nv set router policy route-map BLUEtoRED rule 10 action permit
nv set router policy route-map BLUEtoRED rule 10 set community 11:11
nv set vrf RED router bgp address-family ipv4-unicast route-import from-vrf route-map BLUEtoRED
nv config
```

To check the status of the VRF route leaking, run:

- **NVUE command:**

  ```plaintext
  nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import
  ```

- **Vtysh command:**

  ```plaintext
  show ip bgp vrf <vrf-name> ipv4|ipv6 unicast route-leak command.
  ```

- **For example:**

  ```plaintext
  nv show vrf RED router bgp address-family ipv4-unicast route-import
    operational   applied
    --------------  ----------
    from-vrf       enable    on
    route-map      BLUEtoRED
  [list] BLUE   BLUE
  ```
To show more detailed status information, the following NVUE commands are available:

- `nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import from-vrf`
- `nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import from-vrf list`
- `nv show vrf <vrf-name> router bgp address-family ipv4-unicast route-import from-vrf list <leak-vrf-id>`

To view the BGP routing table, run:

- NVUE command:
  - `nv show vrf <vrf-name> router bgp address-family ipv4-unicast`

  Vtysh command:
  - `show ip bgp vrf <vrf-name> ipv4|ipv6 unicast`

To view the FRR IP routing table, run:

- Vtysh command:
  - `show ip route vrf <vrf-name>`
  - Or:
  - `net show route vrf <vrf-name>`
VLAN Subinterfaces

A VLAN subinterface is a VLAN device on an interface. The VLAN ID appends to the parent interface using dot (.) VLAN notation which is a standard way to specify a VLAN device in Linux.

For example:

- A VLAN with ID 100 which is a subinterface of p0.sf is annotated as p0.sf.100
- The subinterface p0.sf.100 only receives packets that have a VLAN 100 tag on port p0.sf
- Any packets transmitted from p0.sf.100 would have VLAN tag 100

In HBN, VLAN subinterfaces can be created on uplink ports as well as on the host-facing PF and VF ports. A VLAN subinterface only receives traffic tagged for that VLAN.

Note

VLAN subinterfaces are L3 interfaces and should not be added to a bridge.

In the following example, uplink subinterface on p0.sf with VLAN ID 10 and a host facing subinterface on VF ports pf1vf0.sf with VLAN ID 999 are created. The host-facing subinterface is also assigned with IPv4 and IPv6 addresses.

Subinterface configuration using NVUE commands:

```
nv set interface p0.sf.10 base-interface p0.sf
```
Ethernet Virtual Private Network – EVPN

HBN supports VXLAN with EVPN control plane for intra-subnet bridging (L2) services for IPv4 and IPv6 traffic in the overlay.

For the underlay, only IPv4 or BGP unnumbered configuration is supported.

Note

HBN supports VXLAN encapsulation only over uplink parent interfaces.

Single VXLAN Device
With a single VXLAN device, a set of VXLAN network identifiers (VNIs) represents a single device model. The single VXLAN device has a set of attributes that belong to the VXLAN construct. Individual VNIs include VLAN-to-VNI mapping which allows users to specify which VLANs are associated with which VNIs. A single VXLAN device simplifies the configuration and reduces the overhead by replacing multiple traditional VXLAN devices with a single VXLAN device.

Users may configure a single VXLAN device automatically with NVUE, or manually by editing the `/etc/network/interfaces` file. When users configure a single VXLAN device with NVUE, NVUE creates a unique name for the device in the following format using the bridge name as the hash key: vxlan<id>.

This example configuration performs the following steps:

1. Creates a single VXLAN device (vxlan21).
2. Maps VLAN 10 to VNI 10 and VLAN 20 to VNI 20.
3. Adds the VXLAN device to the default bridge.

```
cumulus@leaf01:~$ nv set bridge domain bridge vlan 10 vni 10
cumulus@leaf01:~$ nv set bridge domain bridge vlan 20 vni 20
cumulus@leaf01:~$ nv set nve vxlan source address 10.10.10.1
cumulus@leaf01:~$ nv config apply
```

Alternately, users may edit the file `/etc/network/interfaces` as follows, then run the `ifreload -a` command to apply the SVD configuration.

```
auto lo
iface lo inet loopback
  vxlan-local-tunnelip 10.10.10.1

auto vxlan21
iface vxlan21
  bridge-vlan-vni-map 10=10 20=20
  bridge-learning off

auto bridge
iface bridge
```
bridge-vlan-aware yes  
bridge-ports vxlan21 pf0hpf_sf pf1hpf_sf  
bridge-vids 10 20  
bridge-pvid 1

Note

Users may not use a combination of single and traditional VXLAN devices.

Sample Switch Configuration for EVPN

The following is a sample NVUE config for underlay switches (NVIDIA® Spectrum® with Cumulus Linux) to enable EVPN deployments with HBN.

It assumes that the uplinks on all BlueField devices are connected to ports swp1-4 on the switch.

nv set evpn enable on
nv set router bgp enable on

nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on

nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 63640
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp1 peer-group fabric
nv set vrf default router bgp neighbor swp1 type unnumbered
nv set vrf default router bgp neighbor swp2 peer-group fabric
nv set vrf default router bgp neighbor swp2 type unnumbered
nv set vrf default router bgp neighbor swp3 peer-group fabric
nv set vrf default router bgp neighbor swp3 type unnumbered
nv set vrf default router bgp neighbor swp4 peer-group fabric
nv set vrf default router bgp neighbor swp4 type unnumbered
Layer-2 EVPN

Sample NVUE Configuration for L2 EVPN

The following is a sample NVUE configuration which has L2-VNIs (2000, 2001) for EVPN bridging on BlueField.

```
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp router-id 27.0.0.10

nv set interface lo ip address 2001:c000:10ff:f00d::10/128
nv set interface lo ip address 27.0.0.10/32
nv set interface lo type loopback
nv set interface swp1,swp2,swp3,swp4 type swp

nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default vlan 200 vni 2000 flooding enable auto
nv set bridge domain br_default vlan 200 vni 2000 mac-learning off
nv set bridge domain br_default vlan 201 vni 2001 flooding enable auto
nv set bridge domain br_default vlan 201 vni 2001 mac-learning off

nv set evpn enable on
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.4
nv set router bgp enable on
nv set system global anycast-mac 44:38:39:42:42:07
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on

nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 63642
nv set vrf default router bgp enable on
```
Sample Flat Files Configuration for L2 EVPN

nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast policy outbound route-map MY_ORIGIN_ASPATH_ONLY
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast policy outbound route-map MY_ORIGIN_ASPATH_ONLY
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp router-id 27.0.0.4

nv set interface lo ip address 2001:c000:10ff:f00d::4/128
nv set interface lo ip address 27.0.0.4/32
nv set interface lo type loopback
nv set interface p0_sf,p1_sf, pf0hpf_sf, pf1hpf_sf type swp
nv set interface pf0hpf_sf bridge domain br_default access 200
nv set interface pf1hpf_sf bridge domain br_default access 201

nv set interface vlan200-201 base-interface br_default
nv set interface vlan200-201 ip ipv4 forward on
nv set interface vlan200-201 ip ipv6 forward on
nv set interface vlan200-201 ip vrr enable on
nv set interface vlan200-201 ip vrr state up
nv set interface vlan200-201 link mtu 9050
nv set interface vlan200-201 type svi
nv set interface vlan200 ip address 2001:cafe:1ead::3/64
nv set interface vlan200 ip address 45.3.0.2/24
nv set interface vlan200 ip vrr address 2001:cafe:1ead::1/64
nv set interface vlan200 ip vrr address 45.3.0.1/24
nv set interface vlan200 vlan 200
nv set interface vlan201 ip address 2001:cafe:1ead::1::3/64
nv set interface vlan201 ip address 45.3.1.2/24
nv set interface vlan201 ip vrr address 2001:cafe:1ead::1::1/64
nv set interface vlan201 ip vrr address 45.3.1.1/24
nv set interface vlan201 vlan 201
The following is a sample flat files configuration which has L2-VNIs (vx-2000, vx-2001) for EVPN bridging on BlueField.

This file is located at /etc/network/interfaces:

```plaintext
auto lo
iface lo inet loopback
  address 2001:c000:10ff:f00d::4/128
  address 27.0.0.4/32
  vxlan-local-tunnelip 27.0.0.4

auto p0_sf
iface p0_sf

auto p1_sf
iface p1_sf

auto pf0hpf_sf
iface pf0hpf_sf
  bridge-access 200

auto pf1hpf_sf
iface pf1hpf_sf
  bridge-access 201

auto vlan200
iface vlan200
  address 2001:cafe:1ead::3/64
  address 45.3.0.2/24
  mtu 9050
  address-virtual 00:00:5e:00:01:01 2001:cafe:1ead::1/64 45.3.0.1/24
  vlan-raw-device br_default
  vlan-id 200

auto vlan201
iface vlan201
  address 2001:cafe:1ead:1::3/64
  address 45.3.1.2/24
  mtu 9050
  address-virtual 00:00:5e:00:01:01 2001:cafe:1ead:1::1/64 45.3.1.1/24
  vlan-raw-device br_default
  vlan-id 201
```
This file tells the frr package which daemon to start and is located at /etc/frr/daemons:

```
auto vxlan48
iface vxlan48
  bridge-vlan-vni-map 200=2000 201=2001 217=2017
  bridge-learning off
auto br_default
iface br_default
  bridge-ports pf0hpf_sf pf1hpf_sf vxlan48
  bridge-vlan-aware yes
  bridge-vids 200 201
  bridge-pvid 1

bgpd=yes
ospfd=no
ospf6d=no
isisd=no
pimd=no
ldpd=no
vrrpd=no
fabricd=no
nhrpdp=no
eigrpd=no
babeld=no
sharpd=no
fabricd=no
ripngd=no
ripd=no

vtysh_enable=yes
zebra_options=" -M cumulus_mlag -M snmp -A 127.0.0.1 -s 90000000"
bgpdpd_options=" -M snmp -A 127.0.0.1"
ospfd_options=" -M snmp -A 127.0.0.1"
ospf6d_options=" -M snmp -A 127.0.0.1"
ripd_options=" -A 127.0.0.1"
ripngd_options="-A ::1"
isisd_options=" -A 127.0.0.1"
pimd_options=" -A 127.0.0.1"
ldpd_options=" -A 127.0.0.1"
```
nhrpd_options=" -A 127.0.0.1"
eigrpd_options=" -A 127.0.0.1"
babeld_options=" -A 127.0.0.1"
sharpd_options=" -A 127.0.0.1"
pbrd_options=" -A 127.0.0.1"
staticd_options=" -A 127.0.0.1"
fabricd_options=" -A 127.0.0.1"
vrprd_options=" -A 127.0.0.1"

frr_profile="datacenter"

FRR configuration file is located at /etc/frr/frr.conf:

!---- Cumulus Defaults ----
frr defaults datacenter
log syslog informational
no zebra nexthop kernel enable
vrf default
outer bgp 63642 vrf default
bgp router-id 27.0.0.4
bgp bestpath as-path multipath-relax
timers bgp 3 9
bgp deterministic-med
! Neighbors
neighbor fabric peer-group
neighbor fabric remote-as external
neighbor fabric timers 3 9
neighbor fabric timers connect 10
neighbor fabric advertisement-interval 0
neighbor p0_sf interface peer-group fabric
neighbor p1_sf interface peer-group fabric
address-family ipv4 unicast
maximum-paths ibgp 64
maximum-paths 64
distance bgp 20 200 200
neighbor fabric activate
exit-address-family
address-family ipv6 unicast
maximum-paths ibgp 64
maximum-paths 64
distance bgp 20 200 200
neighbor fabric activate
Layer-3 EVPN with Symmetric Routing

In distributed symmetric routing, each VXLAN endpoint (VTEP) acts as a layer-3 gateway, performing routing for its attached hosts. However, both the ingress VTEP and egress VTEP route the packets (similar to traditional routing behavior of routing to a next-hop router). In a VXLAN encapsulated packet, the inner destination MAC address is the router MAC address of the egress VTEP to indicate that the egress VTEP is the next hop and that it must also perform the routing.

All routing happens in the context of a tenant (VRF). For a packet that the ingress VTEP receives from a locally attached host, the SVI interface corresponding to the VLAN determines the VRF. For a packet that the egress VTEP receives over the VXLAN tunnel, the VNI in the packet has to specify the VRF. For symmetric routing, this is a VNI corresponding to the tenant and is different from either the source VNI or the destination VNI. This VNI is a layer-3 VNI or interconnecting VNI. The regular VNI, which maps a VLAN, is the layer-2 VNI.

For more details about this, refer to the Cumulus Linux User Manual.

Info

HBN uses a one-to-one mapping between an L3 VNI and a tenant (VRF).
In an EVPN symmetric routing configuration, when the switch announces a type-2 (MAC/IP) route, in addition to containing two VNIs (L2 and L3 VNIs), the route also contains separate route targets (RTs) for L2 and L3. The L3 RT associates the route with the tenant VRF. By default, this is auto-derived using the L3 VNI instead of the L2 VNI. However, this is configurable.

For EVPN symmetric routing, users must perform the configuration listed in the following subsections. Optional configuration includes configuring a route distinguisher (RD) and RTs for the tenant VRF, and advertising the locally-attached subnets.

**Sample NVUE Configuration for L3 EVPN**

If using NVUE to configure EVPN symmetric routing, the following is a sample configuration using NVUE commands:

```bash
nv set bridge domain br_default vlan 111 vni 1000111
nv set bridge domain br_default vlan 112 vni 1000112
nv set bridge domain br_default vlan 213 vni 1000213
nv set bridge domain br_default vlan 214 vni 1000214
nv set evpn enable on
nv set interface lo ip address 6.0.0.19/32
nv set interface lo type loopback
nv set interface p0_sf description ‘alias p0_sf to leaf-21 swp3’
nv set interface p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf1hpf_sf,pf1vf0_sf type swp
nv set interface p1_sf description ‘alias p1_sf to leaf-22 swp3’
nv set interface pf0hpf_sf bridge domain br_default access 111
nv set interface pf0hpf_sf description ‘alias pf0hpf_sf to host-211 ens2f0np0’
 nv set interface pf0vf0_sf bridge domain br_default access 112
nv set interface pf0vf0_sf description ‘alias pf0vf0_sf to host-211 ens2f0np0v0’
```
nv set interface pf1hpf_sf bridge domain br_default access 213
nv set interface pf1hpf_sf description 'alias pf1hpf_sf to host-211 ens2f1np1'

nv set interface pf1vf0_sf bridge domain br_default access 214
nv set interface pf1vf0_sf description 'alias pf1vf0_sf to host-211 ens2f1np0v0'

nv set interface vlan111 ip address 60.1.1.21/24
nv set interface vlan111 ip vrr address 60.1.1.250/24

nv set interface vlan111 ip vrr address 2060:1:1:21/64
nv set interface vlan111 ip vrr address 2060:1:1:250/64

nv set interface vlan111 vlan 111

nv set interface vlan111,213 ip vrf vrf2
nv set interface vlan111-112,213-214 ip vrr enable on

nv set interface vlan111-112,213-214 ip vrr mac-address 00:00:5e:00:01:01

nv set interface vlan112 ip address 50.1.1.21/24

nv set interface vlan112 ip address 2050:1:1:21/64
nv set interface vlan112 ip vrr address 2060:1:1:250/64

nv set interface vlan112 ip vrr address 2050:1:1:250/64

nv set interface vlan112 vlan 112

nv set interface vlan112,214 ip vrf vrf1

nv set interface vlan213 ip address 60.1.210.21/24

nv set interface vlan213 ip address 2060:1:1:210:21/64
nv set interface vlan213 ip vrr address 60.1.210.250/24

nv set interface vlan213 ip vrr address 2060:1:1:210:250/64

nv set interface vlan213 ip vrr address 2050:1:1:210:250/64

nv set interface vlan213 vlan 213

nv set interface vlan214 ip address 50.1.210.21/24

nv set interface vlan214 ip vrr address 2050:1:1:210:21/64

nv set interface vlan214 ip vrr address 50.1.210.250/24

nv set interface vlan214 ip vrr address 2050:1:1:210:250/64

nv set interface vlan214 vlan 214

nv set nve vxlan arp-nd-suppress on

nv set nve vxlan enable on

nv set nve vxlan source address 6.0.0.19

nv set platform

nv set router bgp enable on

nv set router policy route-map ALLOW_LOBR rule 10 action permit

nv set router policy route-map ALLOW_LOBR rule 10 match interface lo

nv set router policy route-map ALLOW_LOBR rule 20 action permit

nv set router policy route-map ALLOW_LOBR rule 20 match interface br_default

nv set router policy route-map ALLOW_VRF1 rule 10 action permit

nv set router policy route-map ALLOW_VRF1 rule 10 match interface vrf1

nv set router policy route-map ALLOW_VRF2 rule 10 action permit

nv set router policy route-map ALLOW_VRF2 rule 10 match interface vrf2
nv set router vrr enable on
nv set system global system-mac 00:01:00:00:1e:03
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast multipaths ebgp 16
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected route-map ALLOW_LOBR
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 650019
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor p0_sf address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp neighbor p0_sf address-family l2vpn-evpn enable on
nv set vrf default router bgp neighbor p0_sf peer-group TOR_LEAF_SPINE
nv set vrf default router bgp neighbor p0_sf remote-as external
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p0_sf address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp neighbor p0_sf address-family l2vpn-evpn enable on
nv set vrf default router bgp neighbor p0_sf peer-group TOR_LEAF_SPINE
nv set vrf default router bgp neighbor p0_sf remote-as external
nv set vrf default router bgp neighbor p1_sf address-family l2vpn-evpn enable on
nv set vrf default router bgp neighbor p1_sf peer-group TOR_LEAF_SPINE
nv set vrf default router bgp neighbor p1_sf address-family l2vpn-evpn enable on
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp path-selection routerid-compare on
nv set vrf default router bgp peer-group TOR_LEAF_SPINE address-family ipv4-unicast enable on
nv set vrf default router bgp router-id 6.0.0.19
nv set vrf vrf1 evpn enable on
nv set vrf vrf1 evpn vni 104001
nv set vrf vrf1 loopback ip address 50.1.21.21/32
nv set vrf vrf1 loopback ip address 2050:50:50:21::1/128
nv set vrf vrf1 router bgp address-family ipv4-unicast enable on
nv set vrf vrf1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf vrf1 router bgp address-family ipv4-unicast redistribute connected route-map ALLOW_VRF1
nv set vrf vrf1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf vrf1 router bgp address-family ipv6-unicast enable on
nv set vrf vrf1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf vrf1 router bgp address-family ipv6-unicast redistribute connected route-map ALLOW_VRF1
nv set vrf vrf1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf vrf1 router bgp autonomous-system 650019
nv set vrf vrf1 router bgp enable on
nv set vrf vrf1 router bgp router-id 50.1.21.21
nv set vrf vrf2 evpn enable on
nv set vrf vrf2 evpn vni 104002
nv set vrf vrf2 loopback ip address 60.1.21.21/32
nv set vrf vrf2 loopback ip address 2060:60:60:21::1/128
nv set vrf vrf2 router bgp address-family ipv4-unicast enable on
Sample Flat Files Configuration for L3 EVPN

The following is a sample flat files configuration which has L2 VNIs and L3 VNIs for EVPN bridging and symmetric routing on BlueField.

This file is located at /etc/network/interfaces:

```plaintext
auto lo
iface lo inet loopback
   address 6.0.0.19/32
   vxlan-local-tunnelip 6.0.0.19

auto vrf1
iface vrf1
   address 2050:50:50:21::21/128
   address 50.1.21.21/32
   vrf-table auto

auto vrf2
iface vrf2
   address 2060:60:60:21::21/128
   address 60.1.21.21/32
   vrf-table auto

auto p0_sf
iface p0_sf
   alias alias p0_sf to leaf-21 swp3

auto p1_sf
```
iface p1_sf
  alias alias p1_sf to leaf-22 swp3

auto pf0hpf_sf
iface pf0hpf_sf
  alias alias pf0hpf_sf to host-211 ens2f0np0
  bridge-access 111

auto pf0vf0_sf
iface pf0vf0_sf
  alias alias pf0vf0_sf to host-211 ens2f0np0v0
  bridge-access 112

auto pf1hpf_sf
iface pf1hpf_sf
  alias alias pf1hpf_sf to host-211 ens2f1np1
  bridge-access 213

auto pf1vf0_sf
iface pf1vf0_sf
  alias alias pf1vf0_sf to host-211 ens2f1np0v0
  bridge-access 214

auto vlan111
iface vlan111
  address 2060:1:1:1::21/64
  address 60.1.1.21/24
  address-virtual 00:00:5e:00:01:01 2060:1:1:1::250/64 60.1.1.250/24
  hwaddress 00:01:00:00:1e:03
  vrf vrf2
  vlan-raw-device br_default
  vlan-id 111

auto vlan112
iface vlan112
  address 2050:1:1:1::21/64
  address 50.1.1.21/24
  address-virtual 00:00:5e:00:01:01 2050:1:1:1::250/64 50.1.1.250/24
  hwaddress 00:01:00:00:1e:03
  vrf vrf1
  vlan-raw-device br_default
  vlan-id 112

auto vlan213
iface vlan213  
  address 2060:1:1:210::21/64  
  address 60.1.210.21/24  
  address-virtual 00:00:5e:00:01:01 2060:1:1:210::250/64 60.1.210.250/24  
  hwaddress 00:01:00:00:1e:03  
  vrf vrf2  
  vlan-raw-device br_default  
  vlan-id 213

auto vlan214  
iface vlan214  
  address 2050:1:1:210::21/64  
  address 50.1.210.21/24  
  address-virtual 00:00:5e:00:01:01 2050:1:1:210::250/64 50.1.210.250/24  
  hwaddress 00:01:00:00:1e:03  
  vrf vrf1  
  vlan-raw-device br_default  
  vlan-id 214

auto vlan4058_l3  
iface vlan4058_l3  
  vrf vrf1  
  vlan-raw-device br_default  
  address-virtual none  
  vlan-id 4058

auto vlan4059_l3  
iface vlan4059_l3  
  vrf vrf2  
  vlan-raw-device br_default  
  address-virtual none  
  vlan-id 4059

auto vxlan48  
iface vxlan48  
  bridge-vlan-vni-map 111=1000111 112=1000112 213=1000213 214=1000214 4058=104001 4059=104002  
  bridge-learning off

auto br_default  
iface br_default  
  bridge-ports pf0hf0 sf pf0vf0 sf pf1hf0 sf pf1vf0 sf vxlan48  
  hwaddress 00:01:00:00:1e:03  
  bridge-vlan-aware yes
FRR configuration is located at `/etc/frr/frr.conf`:

```
bridge-vids 111 112 213 214
bridge-pvid 1

fr version 8.4.3
fr defaults datacenter
hostname doca-hbn-service-bf3-s05-1-ipmi
log syslog informational
no zebra nexthop kernel enable
service integrated-vtysh-config
!
vrf vrf1
  vni 104001
  exit-vrf
!
vrf vrf2
  vni 104002
  exit-vrf
!
rtr bgp 650019
  bgp router-id 6.0.0.19
  bgp bestpath as-path multipath-relax
  bgp bestpath compare-routerid
  neighbor TOR_LEAF_SPINE peer-group
  neighbor TOR_LEAF_SPINE advertisement-interval 0
  neighbor TOR_LEAF_SPINE timers 3 9
  neighbor TOR_LEAF_SPINE timers connect 10
  neighbor p0_sf interface peer-group TOR_LEAF_SPINE
  neighbor p0_sf remote-as external
  neighbor p0_sf advertisement-interval 0
  neighbor p0_sf timers 3 9
  neighbor p0_sf timers connect 10
  neighbor p1_sf interface peer-group TOR_LEAF_SPINE
  neighbor p1_sf remote-as external
  neighbor p1_sf advertisement-interval 0
  neighbor p1_sf timers 3 9
  neighbor p1_sf timers connect 10
!
address-family ipv4 unicast
  redistribute connected route-map ALLOW_LOBR
  maximum-paths 16
```
maximum-paths ibgp 64
.exit-address-family

! address-family l2vpn evpn
  neighbor p0_sf activate
  neighbor p1_sf activate
  advertise-all-vni
.exit-address-family
.exit

! router bgp 650019 vrf vrf1
  bgp router-id 50.1.21.21
  ! address-family ipv4 unicast
    redistribute connected route-map ALLOW_VRF1
    maximum-paths 64
    maximum-paths ibgp 64
.exit-address-family

! address-family ipv6 unicast
  redistribute connected route-map ALLOW_VRF1
  maximum-paths 64
  maximum-paths ibgp 64
.exit-address-family

! address-family l2vpn evpn
  advertise ipv4 unicast
  advertise ipv6 unicast
.exit-address-family
.exit

! router bgp 650019 vrf vrf2
  bgp router-id 60.1.21.21
  ! address-family ipv4 unicast
    redistribute connected route-map ALLOW_VRF2
    maximum-paths 64
    maximum-paths ibgp 64
.exit-address-family

! address-family ipv6 unicast
  redistribute connected route-map ALLOW_VRF2
  maximum-paths 64
  maximum-paths ibgp 64
Multi-hop eBGP Peering for EVPN (Route Server in Symmetric EVPN Routing)

eBGP multi-hop peering for EVPN support in a route server-like role in EVPN topology, allows the deployment of EVPN on any cloud that supports IP transport.

Route servers and BF/HBN VTEPs are connected via the IP cloud. That is:

- Switches in the cloud provider need not be EVPN-aware
- Switches in the provider fabric provide IPv4 and IPv6 transport and do not have to support EVPN

Sample Route Server Configuration for EVPN

```yaml
exit-address-family
!
address-family l2vpn evpn
advertise ipv4 unicast
advertise ipv6 unicast
exit-address-family
exit
!
route-map ALLOW_LOBR permit 10
match interface lo
exit
!
route-map ALLOW_LOBR permit 20
match interface br_default
exit
!
route-map ALLOW_VRF1 permit 10
match interface vrf1
exit
!
route-map ALLOW_VRF2 permit 10
match interface vrf2
exit
```
The following is a sample configuration of an Ubuntu server running FRR 9.0 stable, configured as EVPN route server and an HBN VTEP that is peering to two spine switches for IP connectivity and 3 Route servers for EVPN overlay control.

```
root@sn1:/home/cumulus# uname -a
Linux sn1 5.15.0-88-generic #98-Ubuntu SMP Mon Oct 215:18:56 UTC 2023 x86_64 x86_64 x86_64
GNU/Linux
root@sn1:/home/cumulus# dpkg -l frr
Desired=Unknown/Install/Remove/Purge/Hold
  Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
  / Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
  ||/ Name           Version               Architecture Description
  ++)==============-=====================-============-

 ii  frr 9.0.1-0-ubuntu22.04.1 amd64   FRRouting suite of internet protocols (BGP, OSPF, IS-IS, ...)
root@sn1:/home/cumulus#
```

**FRR configuration (frr.conf):**

```
root@sn1:/home/cumulus# sh run
Building configuration...

Current configuration:
!
frr version 9.0.1
frr defaults datacenter
hostname sn1
no ip forwarding
no ipv6 forwarding
service integrated-vtysh-config
!
router bgp 4200065507
bgp router-id 6.0.0.7
timers bgp 60 180
neighbor rclients peer-group
neighbor rclients remote-as external
neighbor rclients ebgp-multihop 10
neighbor rclients update-source lo
neighbor rclients advertisement-interval 0
neighbor rclients timers 3 9
```
neighbor rclients timers connect 10
neighbor rcsuper peer-group
neighbor rcsuper remote-as external
neighbor rcsuper advertisement-interval 0
neighbor rcsuper timers 3 9
neighbor rcsuper timers connect 10
neighbor swp1 interface peer-group rcsuper
bgp listen range 6.0.0.0/24 peer-group rclients

address-family ipv4 unicast
redistribute connected
neighbor fabric route-map pass in
neighbor fabric route-map pass out
no neighbor rclients activate
maximum-paths 64
maximum-paths ibgp 64
exit-address-family

address-family l2vpn evpn
neighbor rclients activate
neighbor rcsuper activate
exit-address-family
exit

route-map pass permit 10
set community 11:11 additive
exit

end
sn1#

Interfaces configuration (/etc/network/interfaces):

root@sn1:/home/cumulus# ifquery -a
auto lo
iface lo inet loopback
  address 6.0.0.7/32

auto lo
iface lo inet loopback

auto swp1
Sample HBN Configuration for Deployments with EVPN Route Server

root@doca-hbn-service-bf2-s12-1-ipmi:/tmp# nv config show -o commands
nv set bridge domain br_default vlan 101 vni 10101
nv set bridge domain br_default vlan 102 vni 10102
nv set bridge domain br_default vlan 201 vni 10201
nv set bridge domain br_default vlan 202 vni 10202
nv set evpn enable on
nv set evpn route-advertise svi-ip off
nv set interface ilan3200 ip vrf internet1
nv set interface ilan3200 vlan 3200
nv set interface ilan3200,slan3201,_vlan101-201,_vlan201-202,_vlan3001-3002 base-interface br_default
nv set interface ilan3200,slan3201,_vlan101-201,_vlan201-202,_vlan3001-3002 type svi
nv set interface lo ip address 6.0.0.13/32
nv set interface lo ip address 2001::13/128
nv set interface lo type loopback
nv set interface
p0_sf,p1_sf,p0hf_pf_sf,p0vf0_sf,p0vf1_sf,p0vf2_sf,p0vf3_sf,p0vf4_sf,p0vf5_sf,p0vf6_sf,p0vf7_sf,p0vf8_sf
:type swp
nv set interface pf0vf0_sf bridge domain br_default access 101
nv set interface pf0vf1_sf bridge domain br_default access 102
nv set interface pf0vf2_sf bridge domain br_default access 201
nv set interface pf0vf3_sf bridge domain br_default access 202
nv set interface slan3201 ip vrf special1
nv set interface slan3201 vlan 3201
nv set interface vlan101 ip address 21.1.0.13/16
nv set interface vlan101 ip address 2020:0:1:13/64
nv set interface vlan101 ip vrr address 21.1.0.250/16
nv set interface vlan101 ip vrr address 2020:0:1:1:250/64
nv set interface vlan101 ip vrr mac-address 00:00:01:00:00:65
nv set interface vlan101 vlan 101
nv set interface vlan101-102,201-202 ip vrr enable on
nv set interface vlan101-102,3001 ip vrf tenant1
nv set interface vlan102 ip address 21.2.0.13/16
nv set interface vlan102 ip address 2020:0:1:2::13/64
nv set interface vlan102 ip vrr address 21.2.0.250/16
nv set interface vlan102 ip vrr address 2020:0:1:2:250/64
nv set interface vlan102 ip vrr mac-address 00:00:01:00:00:66
nv set interface vlan102 vlan 102
nv set interface vlan201 ip address 22.1.0.13/16
nv set interface vlan201 ip address 2020:0:2:1::13/64
nv set interface vlan201 ip vrr address 21.2.0.250/16
nv set interface vlan201 ip vrr address 2020:0:2:1::250/64
nv set interface vlan201 ip vrr mac-address 00:00:02:00:00:c9
nv set interface vlan201 vlan 201
nv set interface vlan201-202,3002 ip vrf tenant2
nv set interface vlan202 ip address 22.2.0.13/16
nv set interface vlan202 ip address 2020:0:2:2::13/64
nv set interface vlan202 ip vrr address 22.2.0.250/16
nv set interface vlan202 ip vrr address 2020:0:2:2:2::250/64
nv set interface vlan202 ip vrr mac-address 00:00:02:00:00:ca
nv set interface vlan202 vlan 202
nv set interface vlan3001 vlan 3001
nv set interface vlan3002 vlan 3002
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan source address 6.0.0.13
nv set platform
nv set router bgp autonomous-system 4200065011
nv set router bgp enable on
nv set router bgp router-id 6.0.0.13
nv set router vrr enable on
nv set system config snippet
nv set system global
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 6.0.0.7 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.7 type numbered
nv set vrf default router bgp neighbor 6.0.0.8 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.8 type numbered
nv set vrf default router bgp neighbor 6.0.0.9 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.9 type numbered
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on

nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group rservers address-family ipv4-unicast enable off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group rservers multihop-ttl 3
nv set vrf default router bgp peer-group rservers remote-as external
nv set vrf default router bgp peer-group rservers update-source lo

nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 42000
nv set vrf internet1 loopback ip address 8.1.0.13/32
nv set vrf internet1 loopback ip address 2008:0:1::13/64
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn enable on

nv set vrf special1 evpn enable on
nv set vrf special1 evpn vni 42001
nv set vrf special1 loopback ip address 9.1.0.13/32
nv set vrf special1 loopback ip address 2009:0:1::13/64
nv set vrf special1 router bgp address-family ipv4-unicast enable on
nv set vrf special1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf special1 router bgp address-family ipv4-unicast route-export to-evpn enable on

nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 30001
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp router-id 6.0.0.13
nv set vrf tenant2 evpn enable on
nv set vrf tenant2 evpn vni 30002
nv set vrf tenant2 router bgp address-family ipv4-unicast enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant2 router bgp enable on
nv set vrf tenant2 router bgp router-id 6.0.0.13
Verifying BGP sessions in HBN:

doca-hbn-service-bf2-s12-1-ipmi# sh bgp sum

IPv4 Unicast Summary (VRF default):
BGP router identifier 6.0.0.13, local AS number 4200065011 vrf-id 0
BGP table version 20
RIB entries 21, using 4032 bytes of memory
Peers 2, using 40 KiB of memory
Peer groups 2, using 128 bytes of memory

<table>
<thead>
<tr>
<th>Neighbor</th>
<th>V</th>
<th>AS</th>
<th>MsgRcvd</th>
<th>MsgSent</th>
<th>TblVer</th>
<th>InQ</th>
<th>OutQ</th>
<th>Up/Down</th>
<th>State/PfxRcd</th>
<th>PfxSnt</th>
<th>Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>spine11(p0_sf)</td>
<td>4</td>
<td>65201</td>
<td>30617</td>
<td>30620</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1d01h30m</td>
<td>9</td>
<td>11</td>
<td>N/A</td>
</tr>
<tr>
<td>spine12(p1_sf)</td>
<td>4</td>
<td>65201</td>
<td>30620</td>
<td>30623</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1d01h30m</td>
<td>9</td>
<td>11</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Total number of neighbors 2

IPv6 Unicast Summary (VRF default):
BGP router identifier 6.0.0.13, local AS number 4200065011 vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 2, using 40 KiB of memory
Peer groups 2, using 128 bytes of memory

<table>
<thead>
<tr>
<th>Neighbor</th>
<th>V</th>
<th>AS</th>
<th>MsgRcvd</th>
<th>MsgSent</th>
<th>TblVer</th>
<th>InQ</th>
<th>OutQ</th>
<th>Up/Down</th>
<th>State/PfxRcd</th>
<th>PfxSnt</th>
<th>Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>spine11(p0_sf)</td>
<td>4</td>
<td>65201</td>
<td>30617</td>
<td>30620</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1d01h30m</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>spine12(p1_sf)</td>
<td>4</td>
<td>65201</td>
<td>30620</td>
<td>30623</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1d01h30m</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Total number of neighbors 2

L2VPN EVPN Summary (VRF default):
BGP router identifier 6.0.0.13, local AS number 4200065011 vrf-id 0
BGP table version 0
RIB entries 79, using 15 KiB of memory
Peers 3, using 60 KiB of memory
Peer groups 2, using 128 bytes of memory

<table>
<thead>
<tr>
<th>Neighbor</th>
<th>V</th>
<th>AS</th>
<th>MsgRcvd</th>
<th>MsgSent</th>
<th>TblVer</th>
<th>InQ</th>
<th>OutQ</th>
<th>Up/Down</th>
<th>State/PfxRcd</th>
<th>PfxSnt</th>
<th>Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn1(6.0.0.7)</td>
<td>4</td>
<td>4200065507</td>
<td>31410</td>
<td>31231</td>
<td>0</td>
<td>0</td>
<td>0:00:27:51</td>
<td>69</td>
<td>95</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>sn2(6.0.0.8)</td>
<td>4</td>
<td>4200065508</td>
<td>31169</td>
<td>31062</td>
<td>0</td>
<td>0</td>
<td>0:02:34:47</td>
<td>69</td>
<td>95</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
The command output shows that the HBN has BGP sessions with spine switches exchanging IPv4/IPv6 unicast. BGP sessions with route servers sn1, sn2, and sn3 only exchanging L2VPN EVPN AFI/SAFI.

**Downstream VNI (DVNI)**

Downstream VNI (symmetric EVPN route leaking) allows users to leak remote EVPN routes without having the source tenant VRF locally configured. A common use case is where upstream switches learn the L3VNI from downstream leaf switches and impose the learned L3VNI to the traffic VXLAN routed to the associated VRF. This eliminates the need to configure L3VNI-SVi interfaces on all leaf switches and enables shared service and hub-and-spoke scenarios.

To configure access to a shared service in a specific VRF, users must:

1. Configure route-target import statements, effectively leaking routes from remote tenants to the shared VRF.

2. Import shared VRF's route-target at the remote nodes.

The route target import or export statement takes the following format:

```
route-target import | export <asn>:<vni>
```

For example:

```
route-target import 65101:6000
```
For route target import statements, users can use `route-target import ANY:<vni>` for NVUE commands or `route-target import *:<vni>` in the `/etc/frr/frr.conf` file. ANY in NVUE commands or the asterisk (*) in the `/etc/frr/frr.conf` file use any ASN (a utonomous system number) as a wildcard.

The NVUE commands are as follows:

1. To configure a route import statement:

   ```
   nv set vrf <vrf> router bgp route-import from-evpn route-target <asn>:<vni>
   ```

2. To configure a route export statement:

   ```
   nv set vrf <vrf> router bgp route-export from-evpn route-target <asn>:<vni>
   ```

Important considerations when implementing DVNI configuration:

- EVPN symmetric mode supports downstream VNI with L3 VNIs and single VXLAN devices only
- You can configure multiple import and export route targets in a VRF
- You cannot leak (import) overlapping tenant prefixes into the same destination VRF

**Note**

If symmetric EVPN configuration is using automatic import/export (which is often the case), when DVNI is configured, automatic import of tenant's VNI is disabled, isolating VRF from the tenant. User must specifically add 'route-target import auto' in such cases to avoid the problem.

**DVNI Configurations for Shared Internet Service**
Configuration example here considers a scenario where External/Internet connectivity is available via a firewall (FW), which is connected to a shared VRF (vrf external in this example).

The routes on super spine switches have external VRF configured in which the route-targets from remote tenants are imported.

On BlueField devices with HBN, a local tenant VRF imports route-target corresponding to the shared external VRF.

Diagram attachment access error: cannot display diagram

L3VNI:

<table>
<thead>
<tr>
<th>Tenant</th>
<th>L3VNI</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenant1</td>
<td>30001</td>
<td>On HBN VTEPs</td>
</tr>
<tr>
<td>tenant2</td>
<td>30002</td>
<td>On HBN VTEPs</td>
</tr>
<tr>
<td>tenant3</td>
<td>30003</td>
<td>On HBN VTEPs</td>
</tr>
<tr>
<td>tenant4</td>
<td>30004</td>
<td>On HBN VTEPs</td>
</tr>
<tr>
<td>tenant5</td>
<td>30005</td>
<td>On HBN VTEPs</td>
</tr>
<tr>
<td>tenant6</td>
<td>30006</td>
<td>On HBN VTEPs</td>
</tr>
<tr>
<td>external</td>
<td>60000</td>
<td>Configured on superspines and connects to external world</td>
</tr>
</tbody>
</table>

On BlueField devices with HBN, every tenant VRF on HBN one must import VNI of shared external VRF:

```
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
```
On super spine switches (SS1 in this example), every remote tenant VRF that needs access to shared services has to be leaked to the shared external VRF.

All super spines in this case need this configuration.

**DVNI Leaked Routes in VRF Table of HBN**

- **Info**
  
  Each super spine here is advertising reachability providing 4-way overlay ECMP.
Kernel table for all tenant VRFs, showing the imported shared service:

```bash
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# ip -4 route show table all 6.0.0.4/32
6.0.0.4 table tenant1 proto bgp metric 20
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48 weight 1
    onlink
6.0.0.4 table tenant2 proto bgp metric 20
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48 weight 1
    onlink
6.0.0.4 table tenant3 proto bgp metric 20
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48 weight 1
    onlink
6.0.0.4 table tenant4 proto bgp metric 20
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.13 ttl 0 tos 0 via 6.0.0.13 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.14 ttl 0 tos 0 via 6.0.0.14 dev vxlan48 weight 1
    onlink
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.15 ttl 0 tos 0 via 6.0.0.15 dev vxlan48 weight 1
    onlink
6.0.0.4 table tenant5 proto bgp metric 20
    nexthop  encap ip id 60000 src 0.0.0.0 dst 6.0.0.12 ttl 0 tos 0 via 6.0.0.12 dev vxlan48 weight 1
    onlink
```
FRR RIB table:

root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# vtysh

Hello, this is FRRouting (version 8.4.3).

doca-hbn-service-bf3-s06-1-ipmi# sh ip route vrf tenant1
Codes: K - kernel route, C - connected, S - static, R - RIP, 
O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP, 
T - Table, A - Babel, D - SHARP, F - PBR, f - OpenFabric, 
Z - FRR, 
> - selected route, * - FIB route, q - queued, r - rejected, b - backup 
t - trapped, o - offload failure

VRF tenant1:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:36
B>* 6.6.0.12/32 [20/0] via 6.6.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
 * via 6.6.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
 * via 6.6.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
 * via 6.6.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
B>* 6.6.0.13/32 [20/0] via 6.6.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
B>* 6.6.0.14/32 [20/0] via 6.6.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
B>* 6.6.0.15/32 [20/0] via 6.6.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:38
VRF default:
B>* 6.0.0.6/32 [20/0] via 600.0.6, vlan4052_l3 onlink, weight 1, 00:05:37
   * via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.7/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:05:48
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:05:48
B>* 6.0.0.8/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:05:38
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:05:38
B>* 6.0.0.9/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:05:28
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:05:28
B>* 6.0.0.10/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:49
B>* 6.0.0.11/32 [20/0] via fe80::202:ff00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.12/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.13/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.14/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:06:47
B>* 6.0.0.15/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1_sf, weight 1, 00:06:47
C>* 6.0.0.16/32 is directly connected, lo, 00:10:42
B>* 6.0.0.18/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1 sf, weight 1, 00:06:47
B>* 6.0.0.20/32 [20/0] via fe80::202:ff00:1f, p0_sf, weight 1, 00:06:47
   * via fe80::202:ff00:27, p1 sf, weight 1, 00:06:47

Codes: K - kernel route, C - connected, S - static, R - RIP,
   O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
   T - Table, A - Babel, D - SHARP, F - PBR, f - OpenFabric,
   Z - FRR,
   > - selected route, * - FIB route, q - queued, r - rejected, b - backup
   t - trapped, o - offload failure
B>* 192.168.0.0/24 [20/0] via fe80::202:ff:fe00:1f, p0_sf, weight 1, 00:05:48
    * via fe80::202:ff:fe00:27, p1_sf, weight 1, 00:05:48

VRF internet1:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 8.1.0.6/32 [20/0] via 6.0.0.6, vlan4004_l3 onlink, weight 1, 00:05:43
C>* 8.1.0.16/32 is directly connected, internet1, 00:10:42
B>* 8.1.0.18/32 [20/0] via 6.0.0.18, vlan4004_l3 onlink, weight 1, 00:05:43
B>* 8.1.0.20/32 [20/0] via 6.0.0.20, vlan4004_l3 onlink, weight 1, 00:05:43

VRF mgmt:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
C>* 10.88.0.0/16 is directly connected, eth0, 00:10:42

VRF special1:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 9.1.0.6/32 [20/0] via 6.0.0.6, vlan4033_l3 onlink, weight 1, 00:05:43
C>* 9.1.0.16/32 is directly connected, special1, 00:10:42
B>* 9.1.0.18/32 [20/0] via 6.0.0.18, vlan4033_l3 onlink, weight 1, 00:05:43
B>* 9.1.0.20/32 [20/0] via 6.0.0.20, vlan4033_l3 onlink, weight 1, 00:05:43

VRF tenant1:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
* via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 7.1.0.6/32 [20/0] via 6.0.0.6, vlan4052_l3 onlink, weight 1, 00:05:43
C>* 7.1.0.16/32 is directly connected, tenant1, 00:10:42
B>* 7.1.0.18/32 [20/0] via 6.0.0.18, vlan4052_l3 onlink, weight 1, 00:05:43
B>* 7.1.0.20/32 [20/0] via 6.0.0.20, vlan4052_l3 onlink, weight 1, 00:05:43
C>* 21.1.0.0/16 is directly connected, vlan101, 00:10:42
C* 21.1.0.0/16 [0/1024] is directly connected, vlan101-v0, 00:10:42
C* 21.2.0.0/16 [0/1024] is directly connected, vlan102-v0, 00:10:42
C>* 21.2.0.0/16 is directly connected, vlan102, 00:10:42
B>* 101.12.4/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 101.13.4/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 101.14.4/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 101.15.4/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
VRF tenant2:
K> 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B> 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
 * via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
 * via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
 * via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 7.2.0.6/32 [20/0] via 6.0.0.6, vlan4037_l3 onlink, weight 1, 00:05:43
C> 7.2.0.16/32 is directly connected, tenant2, 00:10:42
B> 7.2.0.18/32 [20/0] via 6.0.0.18, vlan4037_l3 onlink, weight 1, 00:05:43
B> 7.2.0.20/32 [20/0] via 6.0.0.20, vlan4037_l3 onlink, weight 1, 00:05:43
C 22.1.0.0/16 [0/1024] is directly connected, vlan201-v0, 00:10:42
C> 22.1.0.0/16 is directly connected, vlan201, 00:10:42
C> 22.2.0.0/16 [0/1024] is directly connected, vlan202-v0, 00:10:42
C> 22.2.0.16/32 is directly connected, tenant3, 00:10:42
B> 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44

VRF tenant3:
K> 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B> 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
 * via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
 * via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
 * via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 6.6.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 7.3.0.6/32 [20/0] via 6.0.0.6, vlan4022_l3 onlink, weight 1, 00:05:43
C> 7.3.0.16/32 is directly connected, tenant3, 00:10:42
B> 7.3.0.18/32 [20/0] via 6.0.0.18, vlan4022_l3 onlink, weight 1, 00:05:43
B> 7.3.0.20/32 [20/0] via 6.0.0.20, vlan4022_l3 onlink, weight 1, 00:05:43
C> 23.17.0.0/16 is directly connected, pf0vf4_sf.3, 00:10:42
B> 23.19.0.0/16 [20/0] via 6.0.0.18, vlan4022_l3 onlink, weight 1, 00:05:43
B> 23.21.0.0/16 [20/0] via 6.0.0.20, vlan4022_l3 onlink, weight 1, 00:05:43
B> 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B> 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
VRF tenant4:
K= 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B= 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
   * via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
   * via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
   * via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 7.4.0.6/32 [20/0] via 6.0.0.6, vlan4017_l3 onlink, weight 1, 00:05:43
C= 7.4.0.16/32 is directly connected, tenant4, 00:10:42
B= 7.4.0.18/32 [20/0] via 6.0.0.18, vlan4017_l3 onlink, weight 1, 00:05:43
B= 7.4.0.20/32 [20/0] via 6.0.0.20, vlan4017_l3 onlink, weight 1, 00:05:43
C= 24.17.0.0/16 is directly connected, pf0vf4_sf, 00:10:42
B= 24.19.0.0/16 [20/0] via 6.0.0.18, vlan4017_l3 onlink, weight 1, 00:05:43
B= 24.21.0.0/16 [20/0] via 6.0.0.20, vlan4017_l3 onlink, weight 1, 00:05:43
B= 101.12.4/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 101.13.4/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 101.14.4/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 101.15.4/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44

VRF tenant5:
K= 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B= 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
   * via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
   * via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
   * via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 6.0.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 7.5.0.6/32 [20/0] via 6.0.0.6, vlan4046_l3 onlink, weight 1, 00:05:43
C= 7.5.0.16/32 is directly connected, tenant5, 00:10:42
B= 7.5.0.18/32 [20/0] via 6.0.0.18, vlan4046_l3 onlink, weight 1, 00:05:43
B= 7.5.0.20/32 [20/0] via 6.0.0.20, vlan4046_l3 onlink, weight 1, 00:05:43
C= 25.17.0.0/16 is directly connected, pf0vf4_sf, 00:10:42
B= 25.19.0.0/16 [20/0] via 6.0.0.18, vlan4046_l3 onlink, weight 1, 00:05:43
B= 25.21.0.0/16 [20/0] via 6.0.0.20, vlan4046_l3 onlink, weight 1, 00:05:43
B= 101.12.4/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 101.13.4/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 101.14.4/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B= 101.15.4/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
VRF tenant6:
K>* 0.0.0.0/0 [255/8192] unreachable (ICMP unreachable), 00:10:42
B>* 6.0.0.4/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
  * via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
  * via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.0.0.12/32 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.0.0.13/32 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.0.0.14/32 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 6.0.0.15/32 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 7.6.0.6/32 [20/0] via 6.0.0.6, vlan4041_l3 onlink, weight 1, 00:05:43
C>* 7.6.0.16/32 is directly connected, tenant6, 00:10:42
B>* 7.6.0.18/32 [20/0] via 6.0.0.18, vlan4041_l3 onlink, weight 1, 00:05:43
B>* 7.6.0.20/32 [20/0] via 6.0.0.20, vlan4041_l3 onlink, weight 1, 00:05:43
C>* 26.17.0.0/16 is directly connected, pfovf4_sf.6, 00:10:42
B>* 26.19.0.0/16 [20/0] via 6.0.0.18, vlan4041_l3 onlink, weight 1, 00:05:43
B>* 26.21.0.0/16 [20/0] via 6.0.0.20, vlan4041_l3 onlink, weight 1, 00:05:43
B>* 101.12.4.0/24 [20/0] via 6.0.0.12, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 101.13.4.0/24 [20/0] via 6.0.0.13, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 101.14.4.0/24 [20/0] via 6.0.0.14, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
B>* 101.15.4.0/24 [20/0] via 6.0.0.15, vxlan48 (vrf default) onlink, label 60000, weight 1, 00:05:44
doca-hbn-service-bf3-s06-1-ipmi#

DVNI Debugging

BGP/Zebra debug:

May  7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf tenant1: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafe52d4650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May  7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new pi dest 0xaaaafe52d4650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May  7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf tenant2: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafe52d4650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May  7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new pi dest 0xaaaafe52d4650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May  7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf tenant3: import evpn prefix [5]:[0]:[32]:[6.0.0.4] parent 0xaaaafe52d4650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May  7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new pi dest 0xaaaafe52d4650 (l 2) pi 0xaaaafe5ae400 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf tenant4: import evpn prefix [5]:[0]:[32]:6.0.0.4 parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new pi dest 0xaaaafef519fb0 (l 2) pi 0xaaaafef675e40 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf tenant5: import evpn prefix [5]:[0]:[32]:6.0.0.4 parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new pi dest 0xaaaafef55ae50 (l 2) pi 0xaaaafef5482f0 (l 1, f 0x4010)
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [GKC5Y-XBAX9] vrf tenant6: import evpn prefix [5]:[0]:[32]:6.0.0.4 parent 0xaaaafda63a90 flags 0x410
May 7 20:59:49 doca-hbn-service-bf3-s06-1-ipmi bgpd[1775018]: [KZNVF-SX7KT] ... new pi dest 0xaaaafdaf3590 (l 2) pi 0xaaaafef48fbf0 (l 1, f 0x4010)

DVNI table:

root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# cat /cumulus/nl2docad/run/software-tables/15
{
  "table": {
    "id": 15,
    "name": "HAL Downstream-VNI Table ",
    "count": 1,
    "records": [
      {
        "vni": 60000,
        "fid": 4098,
        "mark-for-del": 0,
        "vtep-users": {
          "count": 4,
          "vtep-user-list": [
            {
              "dest-vtep": "6.0.0.12",
              "dest-mac": "44:38:39:f0:00:12",
              "is-dmac-null": 0,
              "ref-cnt": 36
            },
            {
              "dest-vtep": "6.0.0.14",
              "dest-mac": "44:38:39:f0:00:14",
              "is-dmac-null": 0,
              "ref-cnt": 36
            }
          ]
        }
      }
    ]
  }
Sample DVNI Configuration

HBN configuration example for BlueField devices:

```bash
root@doca-hbn-service-bf3-s06-1-ipmi:/tmp# nv config show -o commands
nv set bridge domain br_default vlan 101 vni 10101
nv set bridge domain br_default vlan 102 vni 10102
nv set bridge domain br_default vlan 201 vni 10201
nv set bridge domain br_default vlan 202 vni 10202
nv set evpn enable on
nv set evpn route-advertise svi-ip off
nv set interface ilan3200 ip vrf internet1
nv set interface ilan3200 vlan 3200
nv set interface ilan3200,slan3201,vlan101-102,201-202,3001-3006 base-interface br_default
nv set interface ilan3200,slan3201,vlan101-102,201-202,3001-3006 type svi
nv set interface lo ip address 6.0.0.16/32
nv set interface lo ip address 2001::16/128
nv set interface lo type loopback
nv set interface p0_sf,p1_sf,pf0hpf_sf,pf0vf0_sf,pf0vf1_sf,pf0vf2_sf,pf0vf3_sf,pf0vf4_sf,pf0vf5_sf,pf0vf6_sf,pf0vf7_sf,pf0vf8_sf type swp
```
nv set interface pf0vf0_sf bridge domain br_default access 101
nv set interface pf0vf1_sf bridge domain br_default access 102
nv set interface pf0vf2_sf bridge domain br_default access 201
nv set interface pf0vf3_sf bridge domain br_default access 202
nv set interface pf0vf4_sf.3 ip address 23.17.0.16 /16
nv set interface pf0vf4_sf.3 ip address 2020:0:3:17::16/64
nv set interface pf0vf4_sf.3 vlan 3
nv set interface pf0vf4_sf.3,vlan3003 ip vrf tenant3
nv set interface pf0vf4_sf.3-6 base-interface pf0vf4_sf
nv set interface pf0vf4_sf.3-6 type sub
nv set interface pf0vf4_sf.4 ip address 24.17.0.16 /16
nv set interface pf0vf4_sf.4 ip address 2020:0:4:17::16/64
nv set interface pf0vf4_sf.4 vlan 4
nv set interface pf0vf4_sf.4,vlan3004 ip vrf tenant4
nv set interface pf0vf4_sf.5 ip address 25.17.0.16 /16
nv set interface pf0vf4_sf.5 ip address 2020:0:5:17::16/64
nv set interface pf0vf4_sf.5 vlan 5
nv set interface pf0vf4_sf.5,vlan3005 ip vrf tenant5
nv set interface pf0vf4_sf.6 ip address 26.17.0.16 /16
nv set interface pf0vf4_sf.6 ip address 2020:0:6:17::16/64
nv set interface pf0vf4_sf.6 vlan 6
nv set interface pf0vf4_sf.6,vlan3006 ip vrf tenant6
nv set interface slan3201 ip vrf special1
nv set interface slan3201 vlan 3201
nv set interface vlan101 ip address 21.1.0.16 /16
nv set interface vlan101 ip address 2020:0:1:1::16/64
nv set interface vlan101 ip vrr address 21.1.0.250 /16
nv set interface vlan101 ip vrr address 2020:0:1:1::250/64
nv set interface vlan101 ip vrr mac-address 00:00:01:00:00:65
nv set interface vlan101 vlan 101
nv set interface vlan101-102,201-202 ip vrr enable on
nv set interface vlan101-102,3001 ip vrf tenant1
nv set interface vlan102 ip address 21.2.0.16 /16
nv set interface vlan102 ip address 2020:0:1:2::16/64
nv set interface vlan102 ip vrr address 21.2.0.250 /16
nv set interface vlan102 ip vrr address 2020:0:1:2::250/64
nv set interface vlan102 ip vrr mac-address 00:00:01:00:00:66
nv set interface vlan102 vlan 102
nv set interface vlan201 ip address 22.1.0.16 /16
nv set interface vlan201 ip address 2020:0:2:1::16/64
nv set interface vlan201 ip vrr address 22.1.0.250 /16
nv set interface vlan201 ip vrr address 2020:0:2:1::250/64
nv set interface vlan201 ip vrr mac-address 00:00:02:00:00:c9
nv set interface vlan201 vlan 201
nv set interface vlan201-202,3002 ip vrf tenant2
nv set interface vlan202 ip address 22.2.0.16/16
nv set interface vlan202 ip address 2020:0:2::16/64
nv set interface vlan202 ip vrr address 22.2.0.250/16
nv set interface vlan202 ip vrr address 2020:0:2::250/64
nv set interface vlan202 ip vrr mac-address 00:00:02:00:00:ca
nv set interface vlan202 vlan 202
nv set interface vlan3001 vlan 3001
nv set interface vlan3002 vlan 3002
nv set interface vlan3003 vlan 3003
nv set interface vlan3004 vlan 3004
nv set interface vlan3005 vlan 3005
nv set interface vlan3006 vlan 3006
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan source address 6.0.0.16
nv set platform
nv set router bgp autonomous-system 65011
nv set router bgp enable on
nv set router bgp router-id 6.0.0.16
nv set router vrr enable on
nv set system config snippet
nv set system global
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 6.0.0.7 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.7 type numbered
nv set vrf default router bgp neighbor 6.0.0.8 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.8 type numbered
nv set vrf default router bgp neighbor 6.0.0.9 peer-group rservers
nv set vrf default router bgp neighbor 6.0.0.9 type numbered
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric bfd detect-multiplier 3
nv set vrf default router bgp peer-group fabric bfd enable on
nv set vrf default router bgp peer-group fabric bfd min-rx-interval 1000
nv set vrf default router bgp peer-group fabric bfd min-tx-interval 1000
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group rservers address-family ipv4-unicast enable off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group rservers multihop-ttl 10
nv set vrf default router bgp peer-group rservers remote-as external
nv set vrf default router bgp peer-group rservers update-source lo
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 42000
nv set vrf internet1 loopback ip address 8.1.0.16/32
nv set vrf internet1 loopback ip address 2008::1:16/64
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp enable on
nv set vrf special1 evpn enable on
nv set vrf special1 evpn vni 42001
nv set vrf special1 loopback ip address 9.1.0.16/32
nv set vrf special1 loopback ip address 2009:0:1:16/64
nv set vrf special1 router bgp address-family ipv4-unicast enable on
nv set vrf special1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf special1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf special1 router bgp address-family ipv6-unicast enable on
nv set vrf special1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf special1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf special1 router bgp enable on
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 30001
nv set vrf tenant1 loopback ip address 7.1.0.16/32
nv set vrf tenant1 loopback ip address 2007:0:1:16/64
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp neighbor 21.1.0.17 peer-group hostgroup
nv set vrf tenant1 router bgp neighbor 21.1.0.17 type numbered
nv set vrf tenant1 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp peer-group hostgroup address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp peer-group hostgroup remote-as external
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant1 router bgp router-id 6.0.0.16
nv set vrf tenant2 evpn enable on
nv set vrf tenant2 evpn vni 30002
nv set vrf tenant2 loopback ip address 7.2.0.16/32
nv set vrf tenant2 loopback ip address 2007::2::16/64
nv set vrf tenant2 router bgp address-family ipv4-unicast enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant2 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant2 router bgp address-family ipv6-unicast enable on
nv set vrf tenant2 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant2 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant2 router bgp enable on
nv set vrf tenant2 router bgp neighbor 22.1.0.17 peer-group hostgroup
nv set vrf tenant2 router bgp neighbor 22.1.0.17 type numbered
nv set vrf tenant2 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant2 router bgp peer-group hostgroup address-family ipv6-unicast enable on
nv set vrf tenant2 router bgp peer-group hostgroup remote-as external
nv set vrf tenant2 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant2 router bgp route-import from-evpn route-target auto
nv set vrf tenant2 router bgp router-id 6.0.0.16
nv set vrf tenant3 evpn enable on
nv set vrf tenant3 evpn vni 30003
nv set vrf tenant3 loopback ip address 7.3.0.16/32
nv set vrf tenant3 loopback ip address 2007::3::16/64
nv set vrf tenant3 router bgp address-family ipv4-unicast enable on
nv set vrf tenant3 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant3 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant3 router bgp address-family ipv6-unicast enable on
nv set vrf tenant3 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant3 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant3 router bgp enable on
nv set vrf tenant3 router bgp neighbor 23.17.0.17 peer-group hostgroup
nv set vrf tenant3 router bgp neighbor 23.17.0.17 type numbered
nv set vrf tenant3 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant3 router bgp peer-group hostgroup address-family ipv6-unicast enable on
nv set vrf tenant3 router bgp peer-group hostgroup remote-as external
nv set vrf tenant3 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant3 router bgp route-import from-evpn route-target auto
nv set vrf tenant3 router bgp router-id 6.0.0.16
nv set vrf tenant3 table auto
nv set vrf tenant4 evpn enable on
nv set vrf tenant4 evpn enable on
nv set vrf tenant4 evpn vni 30004
nv set vrf tenant4 loopback ip address 7.4.0.16/32
nv set vrf tenant4 loopback ip address 2007:0:4::16/64
nv set vrf tenant4 router bgp address-family ipv4-unicast enable on
nv set vrf tenant4 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant4 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant4 router bgp address-family ipv6-unicast enable on
nv set vrf tenant4 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant4 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant4 router bgp enable on
nv set vrf tenant4 router bgp neighbor 24.17.0.17 peer-group hostgroup
nv set vrf tenant4 router bgp neighbor 24.17.0.17 type numbered
nv set vrf tenant4 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant4 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant4 router bgp peer-group hostgroup address-family ipv6-unicast enable on
nv set vrf tenant4 router bgp peer-group hostgroup remote-as external
nv set vrf tenant4 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant4 router bgp route-import from-evpn route-target auto
nv set vrf tenant4 router bgp route-id 6.0.0.16
nv set vrf tenant4 table auto
nv set vrf tenant5 evpn enable on
nv set vrf tenant5 evpn vni 30005
nv set vrf tenant5 loopback ip address 7.5.0.16/32
nv set vrf tenant5 loopback ip address 2007:0:5::16/64
nv set vrf tenant5 router bgp address-family ipv4-unicast enable on
nv set vrf tenant5 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant5 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant5 router bgp address-family ipv6-unicast enable on
nv set vrf tenant5 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant5 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant5 router bgp enable on
nv set vrf tenant5 router bgp neighbor 25.17.0.17 peer-group hostgroup
nv set vrf tenant5 router bgp neighbor 25.17.0.17 type numbered
nv set vrf tenant5 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant5 router bgp peer-group hostgroup address-family ipv4-unicast enable on
nv set vrf tenant5 router bgp peer-group hostgroup address-family ipv6-unicast enable on
nv set vrf tenant5 router bgp peer-group hostgroup remote-as external
nv set vrf tenant5 router bgp route-import from-evpn route-target ANY:60000
nv set vrf tenant5 router bgp route-import from-evpn route-target auto
nv set vrf tenant5 router bgp route-id 6.0.0.16
nv set vrf tenant5 table auto
nv set vrf tenant6 evpn enable on
nv set vrf tenant6 evpn vni 30006
nv set vrf tenant6 loopback ip address 7.6.0.16/32
nv set vrf tenant6 loopback ip address 2007:0:6::16/64
nv set vrf tenant6 router bgp address-family ipv4-unicast enable on
SS1 switch configuration example:

```bash
root@superspine1:mgmt:/home/cumulus# nv config show -o commands
nv set bridge domain br_default vlan 101 vni 10101
nv set bridge domain br_default vlan 102 vni 10102
nv set bridge domain br_default vlan 201 vni 10201
nv set bridge domain br_default vlan 202 vni 10202
nv set evpn enable on
nv set interface eth0 ip address 192.168.0.15/24
nv set interface eth0 ip gateway 192.168.0.2
nv set interface eth0 type eth
nv set interface lo ip address 6.0.0.12/32
nv set interface lo ip address 2001::12/128
nv set interface lo type loopback
nv set interface swp1-6 type swp
nv set interface swp6 ip address 101.12.4.12/24
nv set interface swp6 ip address 2101:12::4:12/112
nv set interface swp6 ip vrf external
nv set nve vxlans arp-nd-suppress on
nv set nve vxlans enable on
nv set nve vxlans source address 6.0.0.12
nv set platform
nv set router bgp autonomous-system 65300
nv set router bgp enable on
nv set router bgp router-id 6.0.0.12
```
nv set system config snippet
nv set system global system-mac 44:38:39:f0:00:12
nv set system hostname superspine1
nv set system ssh-server permit-root-login enabled
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp1 peer-group fabric
nv set vrf default router bgp neighbor swp1 type unnumbered
nv set vrf default router bgp neighbor swp2 peer-group fabric
nv set vrf default router bgp neighbor swp2 type unnumbered
nv set vrf default router bgp neighbor swp3 peer-group rservers
nv set vrf default router bgp neighbor swp3 type unnumbered
nv set vrf default router bgp neighbor swp4 peer-group rservers
nv set vrf default router bgp neighbor swp4 type unnumbered
nv set vrf default router bgp neighbor swp5 peer-group rservers
nv set vrf default router bgp neighbor swp5 type unnumbered
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric bfd detect-multiplier 3
nv set vrf default router bgp peer-group fabric bfd enable on
nv set vrf default router bgp peer-group fabric bfd min-rx-interval 1000
nv set vrf default router bgp peer-group fabric bfd min-tx-interval 1000
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group rservers address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rservers address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group rservers remote-as external
nv set vrf external evpn enable on
nv set vrf external evpn vni 60000
nv set vrf external loopback ip address 6.6.0.12/32
nv set vrf external loopback ip address 2006:0:6::12/64
nv set vrf external router bgp address-family ipv4-unicast enable on
nv set vrf external router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf external router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf external router bgp address-family ipv6-unicast enable on
nv set vrf external router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf external router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf external router bgp address-family l2vpn-evpn enable on
nv set vrf external router bgp enable on
nv set vrf external router bgp neighbor swp6 peer-group peer-group-fw
nv set vrf external router bgp neighbor swp6 type unnumbered
A DPU running the HBN service can be deployed in the role of a border gateway using a combination of HBN features, specifically, EVPN symmetric routing, downstream VNI, VRF route-leaking, and VLAN sub-interfaces. Such a border gateway can do the northbound traffic handoff (to external networks or the Internet) for one or more tenants. In this gateway configuration, the BlueField's uplinks must carry both the tenant traffic which would be in the "overlay" and VXLAN-encapsulated, as well as traffic to and from the external network or Internet, which would be direct-routed in the "underlay". This is accomplished by configuring and running VXLAN-EVPN on the uplink interfaces while configuring and using additional VLAN sub-interfaces on those same uplinks for the traffic to and from external networks. These VLAN sub-interfaces would be configured into an Internet or external VRF for separation from the VXLAN-encapsulated traffic which is carried over the default VRF.

With a BlueField running HBN able to act as a border gateway, there is no longer a dependence on physical switches and routers to terminate VXLAN traffic and perform this role, hence the requirements on the underlying network is simply to provide end-to-end IP/UDP connectivity and facilitate the setup of overlay networks on top. Additionally, multiple border gateways can be easily deployed in the network, including dedicated gateways per tenant or shared gateways for groups of tenants.

Note

```
root@superspine1:mgmt:/home/cumulus#
```
For more details and configuration of some of the key features that together enable the border gateway functionality, refer to sections on Downstream VNIs and VLAN Subinterfaces.

**Gateway Application Example**

The following topology diagram and associated configuration snippets show two different use cases of border gateway deployment:

- **tenant1** is an example of a tenant hosted on a server(s) with a non-gateway BlueField, using a dedicated border gateway on BlueField Gw-HBN1 for Internet connectivity. Traffic flow to and from the Internet for this tenant is marked in pink.

- **gw_tenant1** is an example of a tenant hosted on a server(s) with a gateway BlueField. In this case, the border gateway for this tenant is provided by BlueField Gw-HBN2. Traffic flow to and from the Internet for this tenant is depicted in blue.
L3 VNI Origin Map

<table>
<thead>
<tr>
<th>HBN</th>
<th>VRF</th>
<th>L3 VNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>gw-hbn1 and gw-hbn2</td>
<td>internet1</td>
<td>10000</td>
</tr>
<tr>
<td>gw-hbn1 and gw-hbn2</td>
<td>gw_tenant1</td>
<td>30000</td>
</tr>
<tr>
<td>tenant-hbn3 and tenant-hbn4</td>
<td>tenant1</td>
<td>20000</td>
</tr>
</tbody>
</table>

Configuration Snippet for Internet VRF

- Internet VRF is established in BGP sessions using sub-interface features with underlay switches (i.e., p0_sf.60 and p1_sf.60)
The Internet VRF also imports all the tenant VRFs (local and remote) using the downstream VNI feature with from-EVPN syntax.

```
nv set interface p0.sf.60,p1.sf.60,vlan10 ip vrf internet1
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 10000
    nv set vrf internet1 loopback ip address 6.2.0.1/32
nv set vrf internet1 loopback ip address 2001:cafe:feed::1/128
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family l2vpn-evpn enable on
nv set vrf internet1 router bgp autonomous-system 65552
nv set vrf internet1 router bgp enable on
nv set vrf internet1 router bgp neighbor p0.sf.60 capabilities source-address internet1
nv set vrf internet1 router bgp neighbor p0.sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p0.sf.60 type unnumbered
nv set vrf internet1 router bgp neighbor p1.sf.60 capabilities source-address internet1
nv set vrf internet1 router bgp neighbor p1.sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p1.sf.60 type unnumbered
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv4-unicast enable on
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv6-unicast enable on
nv set vrf internet1 router bgp peer-group l3_pg1 remote-as external
nv set vrf internet1 router bgp route-export to-evpn route-target 65552:10000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:20000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:30000
nv set vrf internet1 router bgp route-import from-evpn route-target auto
nv set vrf internet1 router bgp router-id 27.0.0.5
```

**Configuration Snippet for Gateway Local Tenant**

- `gw_tenant` is stretched across 2 gateway and connected using L3 VNI
- `gw_tenant` has multiple SVIs, which are represented as vlan30 and vlan31 SVIs
- Internet L3 VNI is imported using DVNI. The example also explicitly adds route targets using auto.
gw_tenant VRF:

```
nv set interface vlan30-31 ip vrf gw_tenant1
nv set vrf gw_tenant1 evpn enable on
nv set vrf gw_tenant1 evpn vni 30000
nv set vrf gw_tenant1 loopback ip address 15.3.0.1/32
nv set vrf gw_tenant1 loopback ip address 2001:bad:c0de::1/128
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf gw_tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf gw_tenant1 router bgp autonomous-system 65552
nv set vrf gw_tenant1 router bgp enable on
nv set vrf gw_tenant1 router bgp route-export to-evpn route-target 65552:30000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target auto
nv set vrf gw_tenant1 router bgp router-id 27.0.0.5
```

**Configuration Snippet for Remote Tenant**

- tenant1 is stretched across 2 remote HBN VTEP and connected using L3 VNI
- tenant1 is importing Internet L3 VNI routes in tenant1 and adding its own using route-target auto

Tenant VRF:

```
nv set interface vlan20-21 ip vrf tenant1
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 20000
nv set vrf tenant1 loopback ip address 15.1.0.1/32
nv set vrf tenant1 loopback ip address 2001:c001:c0de::1/128
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
```
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf tenant1 router bgp autonomous-system 6300656
nv set vrf tenant1 router bgp enable on
nv set vrf tenant1 router bgp route-export to-evpn route-target 6300656:20000
nv set vrf tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf tenant1 router bgp route-import from-evpn route-target auto
nv set vrf tenant1 router bgp router-id 27.0.0.17

HBN Accelerated Routing Plan

The following subsections pick a few IP endpoints from the code snippets above and examine their route distribution.

- The gateway devices have a remote tenant
- Internet route is injected using the default originator from the exit node

Gateway-1 Route Info

- BGP sharing the uplink via a sub-interface feature in the Internet VRF.

  ```
  root@hbn:# ip -4 route show vrf internet1 default
default proto bgp metric 20
  nexthop via 169.254.0.1 dev p0_sf.60 weight 1 onlink
  nexthop via 169.254.0.1 dev p1_sf.60 weight 1 onlink
  
  root@hbn:# ip -6 route show vrf internet1 default
default proto bgp metric 20 pref medium
  nexthop via fe80::202:ff:fe00:1b dev p0_sf.60 weight 1
  nexthop via fe80::202:ff:fe00:23 dev p1_sf.60 weight 1
  ```

- Local Tenant routing information: The Internet is reached using L3 VNI via a peer gateway.

  ```
  root@hbn:# ip -4 route show vrf gw_tenant1 default
  ```
Remote tenant routing reachability via gateway1 using DVNI CFG.

Considering an IP endpoint from the remote tenant1 VRF on Tenant-HBN3.

Tenant-HBN3 Route Info

- IP endpoint as gateway1 VRF loopback and DVNI handoff for the VNI is reaching the gateway1 node.

- Internet VRF default route is reaching the remote tenant VRF.
Gateway and Tenant Complete Configuration Example

Gateway-1 Full Configuration

default proto bgp metric 20
    nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.5 ttl 0 tos 0 via 27.0.0.5 dev vxlan48 weight 1 onlink
    nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.7 ttl 0 tos 0 via 27.0.0.7 dev vxlan48 weight 1 onlink

root@hbn:/# ip -6 route show vrf tenant1 default
default proto bgp metric 20 pref medium
    nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.5 ttl 0 tos 0 via ::ffff:27.0.0.5 dev vxlan48 weight 1 onlink
    nexthop encap ip id 10000 src 0.0.0.0 dst 27.0.0.7 ttl 0 tos 0 via ::ffff:27.0.0.7 dev vxlan48 weight 1 onlink

nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 10,30-31
nv set evpn enable on
nv set interface lo ip address 27.0.0.5/32
nv set interface lo ip address 2001:c001:ff:f00d::5/128
nv set interface lo type loopback
nv set interface p0_sf,p1_sf, pf0hpf_sf, pf0vf0_sf, pf0vf10.sf, pf0vf11.sf, pf0vf12.sf, pf0vf1.sf, pf0vf2.sf, pf0vf3.sf, pf0vf4.sf, pf0v type swp
nv set interface p0_sf.60 base-interface p0_sf
nv set interface p0_sf.60,p1_sf.60 type sub
nv set interface p0_sf.60,p1_sf.60 vlan 60
nv set interface p0_sf.60,p1_sf.60,vlan10 ip vrf internet1
nv set interface p1_sf.60 base-interface p1_sf
nv set interface pf0hpf_sf bridge domain br_default access 30
nv set interface pf0vf0_sf bridge domain br_default access 31
nv set interface vlan10 ip address 12.2.0.1/24
nv set interface vlan10 ip address 2001:c001:d00d::1/96
nv set interface vlan10 vlan 10
nv set interface vlan10,30-31 ip ipv4 forward on
nv set interface vlan10,30-31 ip ipv6 forward on
nv set interface vlan10,30-31 type svi
nv set interface vlan30 ip address 45.3.0.1/24
nv set interface vlan30 ip address 2001:b055:b00c::1/96
nv set interface vlan30 vlan 30
nv set interface vlan30-31 ip vrf gw_tenant1
nv set interface vlan31 ip address 45.3.1.1/24
nv set interface vlan31 ip address 2001:b055:b00c::1:0/96
nv set interface vlan31 vlan 31
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.5
nv set platform
nv set router bgp enable on
nv set system config snippet
nv set system global anycast-mac 44:38:39:42:42:17
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 65552
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0_sf capabilities source-address lo
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf capabilities source-address lo
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.5
nv set vrf gw_tenant1 evpn enable on
nv set vrf gw_tenant1 evpn vni 30000
nv set vrf gw_tenant1 loopback ip address 15.3.0.1/32
nv set vrf gw_tenant1 loopback ip address 2001:bad:c0de::1/128
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf gw_tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf gw_tenant1 router bgp autonomous-system 65552
nv set vrf gw_tenant1 router bgp enable on
nv set vrf gw_tenant1 router bgp route-export to-evpn route-target 65552:30000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target auto
nv set vrf gw_tenant1 router bgp router-id 27.0.0.5
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 10000
nv set vrf internet1 loopback ip address 6.2.0.1/32
nv set vrf internet1 loopback ip address 2001:cafe:feed::1/128
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family l2vpn-evpn enable on
nv set vrf internet1 router bgp autonomous-system 65552
nv set vrf internet1 router bgp enable on
nv set vrf internet1 router bgp neighbor p0_sf.60 capabilities source-address internet1	nv set vrf internet1 router bgp neighbor p0_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p0_sf.60 type unnumbered
nv set vrf internet1 router bgp neighbor p1_sf.60 capabilities source-address internet1
Gateway-2 Full Configuration

```plaintext	nv set bridge domain br_default encap 802.1Q	nv set bridge domain br_default type vlan-aware	nv set bridge domain br_default untagged 1	nv set bridge domain br_default vlan 10,30-31	nv set evpn enable on	nv set interface lo ip address 27.0.0.7/32	nv set interface lo ip address 2001:c001:ff:f00d::7/128	nv set interface lo type loopback
nv set interface p0.sf,p1.sf,pf0hpf_sf,pf0vf0_sf,pf0vf10_sf,pf0vf11_sf,pf0vf12_sf,pf0vf1_sf,pf0vf2_sf,pf0vf3_sf,pf0vf4_sf,pf0v type swp	nv set interface p0.sf.60 base-interface p0_sf
nv set interface p0.sf.60,p1.sf.60 type sub
nv set interface p0.sf.60,p1.sf.60 vlan 60
nv set interface p0.sf.60,p1.sf.60,vlan10 ip vrf internet1
nv set interface p1.sf.60 base-interface p1_sf
nv set interface pf0hpf_sf bridge domain br_default access 30
nv set interface pf0vf0_sf bridge domain br_default access 31
nv set interface vlan10 ip address 12.2.1.1/24
nv set interface vlan10 ip address 2001:c001:d00d::1:0:1/96
nv set interface vlan10 vlan 10
nv set interface vlan10,30-31 ip ipv4 forward on
nv set interface vlan10,30-31 ip ipv6 forward on
nv set interface vlan10,30-31 type svi
nv set interface vlan30 ip address 45.3.2.1/24
nv set interface vlan30 ip address 2001:b055:b00c::2:0:1/96
nv set interface vlan30 vlan 30
```

DOCA Services
nv set interface vlan30-31 ip vrf gw_tenant1
nv set interface vlan31 ip address 45.3.3.1/24
nv set interface vlan31 ip address 2001:b055:b00c::3:0:1/96
nv set interface vlan31 vlan 31
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.7
nv set platform
nv set router bgp enable on
nv set system config snippet
nv set system global anycast-mac 44:38:39:42:42:19
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 65554
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0_sf capabilities source-address lo
nv set vrf default router bgp neighbor p0_sf peer-group fabric
nv set vrf default router bgp neighbor p0_sf type unnumbered
nv set vrf default router bgp neighbor p1_sf capabilities source-address lo
nv set vrf default router bgp neighbor p1_sf peer-group fabric
nv set vrf default router bgp neighbor p1_sf type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.7
nv set vrf gw_tenant1 evpn enable on
nv set vrf gw_tenant1 evpn vni 30000
nv set vrf gw_tenant1 loopback ip address 15.3.0.2/32
nv set vrf gw_tenant1 loopback ip address 2001:bad:c0de::2/128
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf gw_tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf gw_tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf gw_tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf gw_tenant1 router bgp autonomous-system 65554
nv set vrf gw_tenant1 router bgp enable on
nv set vrf gw_tenant1 router bgp route-export to-evpn route-target 65554:30000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target ANY:10000
nv set vrf gw_tenant1 router bgp route-import from-evpn route-target auto
nv set vrf gw_tenant1 router bgp router-id 27.0.0.7
nv set vrf internet1 evpn enable on
nv set vrf internet1 evpn vni 10000
nv set vrf internet1 loopback ip address 6.2.0.2/32
nv set vrf internet1 loopback ip address 2001:cafe:feed::2/128
nv set vrf internet1 router bgp address-family ipv4-unicast enable on
nv set vrf internet1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family ipv6-unicast enable on
nv set vrf internet1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf internet1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf internet1 router bgp address-family l2vpn-evpn enable on
nv set vrf internet1 router bgp autonomous-system 65554
nv set vrf internet1 router bgp enable on
nv set vrf internet1 router bgp neighbor p0_sf.60 capabilities source-address internet1
nv set vrf internet1 router bgp neighbor p0_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p0_sf.60 type unnumbered
nv set vrf internet1 router bgp neighbor p1_sf.60 capabilities source-address internet1
nv set vrf internet1 router bgp neighbor p1_sf.60 peer-group l3_pg1
nv set vrf internet1 router bgp neighbor p1_sf.60 type unnumbered
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv4-unicast enable on
nv set vrf internet1 router bgp peer-group l3_pg1 address-family ipv6-unicast enable on
nv set vrf internet1 router bgp peer-group l3_pg1 remote-as external
nv set vrf internet1 router bgp route-export to-evpn route-target 65554:10000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:20000
nv set vrf internet1 router bgp route-import from-evpn route-target ANY:30000
nv set vrf internet1 router bgp route-import from-evpn route-target auto
nv set vrf internet1 router bgp router-id 27.0.0.7

Tenant-HBN-3 Full Configuration

nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 20-21
nv set evpn enable on
nv set interface lo ip address 27.0.0.17/32
nv set interface lo ip address 2001:c001:ff:f00d::11/128
nv set interface lo type loopback
nv set interface p0-1,pf0hpf,pf0vf0-12,pf1hpf,pf1vf0-4 type swp
nv set interface pf0hpf bridge domain br_default access 20
nv set interface pf0vf0 bridge domain br_default access 21
nv set interface vlan20 ip address 45.1.0.1/24
nv set interface vlan20 ip address 2001:c001:b00c::1/96
nv set interface vlan20 vlan 20
nv set interface vlan20-21 ip ipv4 forward on
nv set interface vlan20-21 ip ipv6 forward on
nv set interface vlan20-21 ip vrf tenant1
nv set interface vlan20-21 type svi
nv set interface vlan21 ip address 45.1.1.1/24
nv set interface vlan21 ip address 2001:c001:b00c::1:0:1/96
nv set interface vlan21 vlan 21
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.17
nv set platform
nv set router bgp enable on
nv set system global anycast-mac 44:38:39:42:42:21
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 6300656
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0 capabilities source-address lo
nv set vrf default router bgp neighbor p0 peer-group fabric
nv set vrf default router bgp neighbor p0 type unnumbered
nv set vrf default router bgp neighbor p1 capabilities source-address lo
nv set vrf default router bgp neighbor p1 peer-group fabric
nv set vrf default router bgp neighbor p1 type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.17
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 20000
nv set vrf tenant1 loopback ip address 15.1.0.1/32
nv set vrf tenant1 loopback ip address 2001:c001:c0de::1/128
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf tenant1 router bgp autonomous-system 6300656
nv set vrf tenant1 router bgp enable on
Tenant-HBN-4 Full Configuration

nv set bridge domain br_default encap 802.1Q
nv set bridge domain br_default type vlan-aware
nv set bridge domain br_default untagged 1
nv set bridge domain br_default vlan 20-21
nv set evpn enable on
nv set interface lo ip address 27.0.0.19/32
nv set interface lo ip address 2001:c001:ff00::13/128
nv set interface lo type loopback
nv set interface p0-1,pf0hpf,pf0vfo-12,pf1hpf,pf1vf0-4 type swp
nv set interface pf0hpf bridge domain br_default access 20
nv set interface pf0vfo bridge domain br_default access 21
nv set interface vlan20 ip address 45.1.2.1/24
nv set interface vlan20 ip address 2001:c001:b00c::2:0:1/96
nv set interface vlan20 vlan 20
nv set interface vlan20-21 ip ipv4 forward on
nv set interface vlan20-21 ip ipv6 forward on
nv set interface vlan20-21 ip vrf tenant1
nv set interface vlan20-21 type svi
nv set interface vlan21 ip address 45.1.3.1/24
nv set interface vlan21 ip address 2001:c001:b00c::3:0:1/96
nv set interface vlan21 vlan 21
nv set nve vxlan arp-nd-suppress on
nv set nve vxlan enable on
nv set nve vxlan mac-learning off
nv set nve vxlan source address 27.0.0.19
nv set platform
nv set router bgp enable on
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf default router bgp address-family ipv6-unicast enable on
nv set vrf default router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf default router bgp address-family l2vpn-evpn enable on
nv set vrf default router bgp autonomous-system 6300658
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor 27.0.0.11 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.11 type numbered
nv set vrf default router bgp neighbor 27.0.0.12 peer-group rs_client
nv set vrf default router bgp neighbor 27.0.0.12 type numbered
nv set vrf default router bgp neighbor p0 capabilities source-address lo
nv set vrf default router bgp neighbor p0 peer-group fabric
nv set vrf default router bgp neighbor p0 type unnumbered
nv set vrf default router bgp neighbor p1 capabilities source-address lo
nv set vrf default router bgp neighbor p1 peer-group fabric
nv set vrf default router bgp neighbor p1 type unnumbered
nv set vrf default router bgp path-selection multipath aspath-ignore on
nv set vrf default router bgp peer-group fabric address-family ipv4-unicast enable on
nv set vrf default router bgp peer-group fabric address-family ipv6-unicast enable on
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group fabric address-family l2vpn-evpn enable off
nv set vrf default router bgp peer-group fabric remote-as external
nv set vrf default router bgp peer-group fabric timers connection-retry 5
nv set vrf default router bgp peer-group fabric timers hold 30
nv set vrf default router bgp peer-group fabric timers keepalive 10
nv set vrf default router bgp peer-group rs_client address-family ipv4-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family ipv6-unicast enable off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn add-path-tx off
nv set vrf default router bgp peer-group rs_client address-family l2vpn-evpn enable on
nv set vrf default router bgp peer-group rs_client multihop-ttl 5
nv set vrf default router bgp peer-group rs_client remote-as external
nv set vrf default router bgp peer-group rs_client timers connection-retry 5
nv set vrf default router bgp peer-group rs_client timers hold 30
nv set vrf default router bgp peer-group rs_client timers keepalive 10
nv set vrf default router bgp router-id 27.0.0.19
nv set vrf tenant1 evpn enable on
nv set vrf tenant1 evpn vni 20000
nv set vrf tenant1 loopback ip address 15.1.0.2/32
nv set vrf tenant1 loopback ip address 2001:c001:c0de::2/128
nv set vrf tenant1 router bgp address-family ipv4-unicast enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv4-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast redistribute connected enable on
nv set vrf tenant1 router bgp address-family ipv6-unicast route-export to-evpn enable on
nv set vrf tenant1 router bgp address-family l2vpn-evpn enable on
nv set vrf tenant1 router bgp autonomous-system 6300658
nv set vrf tenant1 router bgp enable on
Access Control Lists

Access Control Lists (ACLs) are a set of rules that are used to filter network traffic. These rules are used to specify the traffic flows that must be permitted or blocked at networking device interfaces. There are two types of ACLs:

- **Stateless ACLs** – rules that are applied to individual packets. They inspect each packet individually and permit/block the packets based on the packet header information and the match criteria specified by the rule.

- **Stateful ACLs** – rules that are applied to traffic sessions/connections. They inspect each packet with respect to the state of the session/connection to which the packet belongs to determine whether to permit/block the packet.

**Stateless ACLs**

HBN supports configuration of stateless ACLs for IPv4 packets, IPv6 packets, and Ethernet (MAC) frames. The following examples depict how stateless ACLs are configured for each case, with NVUE and with flat files (cl-acltool).

**NVUE Examples for Stateless ACLs**

**NVUE IPv4 ACLs Example**

The following is an example of an ingress IPv4 ACL that permits DHCP request packets ingressing on the `pf0hpf_sf` port towards the DHCP server:

```
root@hbn01-host01:~# nv set acl acl1_ingress type ipv4
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip protocol udp
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip dest-port 67
```
Bind the ingress IPv4 ACL to host representor port pf0hpf_sf of BlueField in the inbound direction:

code
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 match ip source-port 68
root@hbn01-host01:~# nv set acl acl1_ingress rule 100 action permit

The following is an example of an egress IPv4 ACL that permits DHCP reply packets egressing out of the pf0hpf_sf port towards the DHCP client:

code
root@hbn01-host01:~# nv set acl acl2_egress type ipv4
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip protocol udp
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip dest-port 68
root@hbn01-host01:~# nv set acl acl2_egress rule 200 match ip source-port 67
root@hbn01-host01:~# nv set acl acl2_egress rule 200 action permit

Bind the egress IPv4 ACL to host representor port pf0hpf_sf of BlueField in the outbound direction:

code
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl2_egress outbound
root@hbn01-host01:~# nv config apply

NVUE IPv6 ACLs Example

The following is an example of an ingress IPv6 ACL that permits traffic with matching dest-ip and protocol tcp ingress on port pf0hpf_sf:

code
root@hbn01-host01:~# nv set acl acl5_ingress type ipv6
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 match ip protocol tcp
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 match ip dest-ip 48:2034::80:9
root@hbn01-host01:~# nv set acl acl5_ingress rule 100 action permit
Bind the ingress IPv6 ACL to host representor port pf0hpf_sf of BlueField in the inbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl5_ingress inbound
root@hbn01-host01:~# nv config apply
```

The following is an example of an egress IPv6 ACL that permits traffic with matching source-ip and protocol tcp egressing out of port pf0hpf_sf:

```
root@hbn01-host01:~# nv set acl acl6_egress type ipv6
root@hbn01-host01:~# nv set acl acl6_egress rule 101 match ip protocol tcp
root@hbn01-host01:~# nv set acl acl6_egress rule 101 match ip source-ip 48:2034::80:9
root@hbn01-host01:~# nv set acl acl6_egress rule 101 action permit
```

Bind the egress IPv6 ACL to host representor port pf0hpf_sf of BlueField in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl6_egress outbound
root@hbn01-host01:~# nv config apply
```

**NVUE MAC ACLs Example**

The following is an example of an ingress MAC ACL that permits traffic with matching source-mac and dest-mac ingressing to port pf0hpf_sf:

```
root@hbn01-host01:~# nv set acl acl3_ingress type mac
root@hbn01-host01:~# nv set acl acl3_ingress rule 1 match mac source-mac 00:00:00:00:00:0a
root@hbn01-host01:~# nv set acl acl3_ingress rule 1 match mac dest-mac 00:00:00:00:00:0b
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl3_ingress inbound
```

Bind the ingress MAC ACL to host representor port pf0hpf_sf of BlueField in the inbound direction:
The following is an example of an egress MAC ACL that permits traffic with matching source-mac and dest-mac egressing out of port pf0hpf_sf:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl4_egress type mac
root@hbn01-host01:~# nv set acl acl4_egress rule 2 match mac source-mac 00:00:00:00:00:0b
root@hbn01-host01:~# nv set acl acl4_egress rule 2 match mac dest-mac 00:00:00:00:00:0a
root@hbn01-host01:~# nv set acl acl4_egress rule 2 action permit
```

Bind the egress MAC ACL to host representor port pf0hpf_sf of BlueField in the outbound direction:

```
root@hbn01-host01:~# nv set interface pf0hpf_sf acl acl4_egress outbound
root@hbn01-host01:~# nv config apply
```

**Flat Files (cl-acltool) Examples for Stateless ACLs**

For the same examples cited above, the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`. The rules in `/etc/cumulus/acl/policy.d/<rule_name.rules>` are configured using Linux iptables/ip6tables/ebtables.

**Flat Files IPv4 ACLs Example**

The following example configures an ingress IPv4 ACL rule matching with DHCP request under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL acl1_ingress in dir inbound on interface pf1vf1_sf ##
-t filter -A FORWARD -m physdev --physdev-in pf1vf1_sf -p udp --sport 68 --dport 67 -j ACCEPT
```
The following example configures an egress IPv4 ACL rule matching with DHCP reply under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the egress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[iptables]
## ACL acl2_egress in dir outbound on interface pf1vf1_sf ##
-t filter -A FORWARD -m physdev --physdev-out pf1vf1_sf -p udp --sport 67 --dport 68 -j ACCEPT
```

**Flat File IPv6 ACLs Example**

The following example configures an ingress IPv6 ACL rule matching with dest-ip and tcp protocol under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ip6tables]
## ACL acl5_ingress in dir inbound on interface pf0hpf_sf ##
-t filter -A FORWARD -m physdev --physdev-in pf0hpf_sf -d 48:2034::80:9 -p tcp -j ACCEPT
```

The following example configures an egress IPv6 ACL rule matching with source-ip and tcp protocol under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the egress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ip6tables]
## ACL acl6_egress in dir outbound on interface pf0hpf_sf ##
-t filter -A FORWARD -m physdev --physdev-out pf0hpf_sf -s 48:2034::80:9 -p tcp -j ACCEPT
```

**Flat Files MAC ACLs Example**

The following example configures an ingress MAC ACL rule matching with source-mac and dest-mac under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with the ingress interface as the host representor of BlueField followed by invoking `cl-acltool -i`:

```
[ebtables]
## ACL acl3_ingress in dir inbound on interface pf0hpf_sf ##
```
The following example configures an egress MAC ACL rule matching with `source-mac` and `dest-mac` under `/etc/cumulus/acl/policy.d/<rule_name.rules>` with egress interface as host representor of BlueField followed by invoking `cl-acltool -i`:

```
[tfilter]
-A FORWARD -m physdev --physdev-in pf0hpf_sf -s 00:00:00:00:00:0a/ff:ff:ff:ff:ff:ff -d 00:00:00:00:00:0b/ff:ff:ff:ff:ff:ff -j ACCEPT
```

**Stateful ACLs**

Stateful ACLs facilitate monitoring and tracking traffic flows to enforce per-flow traffic filtering (unlike stateless ACLs which filter traffic on a per-packet basis). HBN supports stateful ACLs using reflexive ACL mechanism. Reflexive ACL mechanism is used to allow initiation of connections from "within" the network to "outside" the network and allow only replies to the initiated connections from "outside" the network (or vice versa).

HBN supports stateful ACL configuration for IPv4 traffic.

Stateful ACLs can be applied for native routed traffic (north-south underlay routed traffic in EVPN deployments), EVPN bridged traffic (east-west overlay bridged/L2 traffic in EVPN deployments) and EVPN routed traffic (east-west overlay routed traffic in EVPN deployments). Stateful ACLs applied for native routed traffic are called "Native-L3 stateful ACLs". Stateful ACLs applied for EVPN bridged traffic and EVPN routed traffic are called "EVPN-L2 stateful ACLs" and "EVPN-L3 stateful ACLs", respectively.

Stateful ACLs in HBN are disabled by default. To enable stateful ACL functionality, use the following NVUE commands:

```
root@hbn03-host00:~# nv set system reflexive-acl enable
root@hbn03-host00:~# nv config apply
```
If using flat-file configuration (and not NVUE), edit the file `/etc/cumulus/nl2docad.d/acl.conf` and set the knob `rflx.reflexive_acl_enable` to `TRUE`. To apply this change, execute:

```bash
root@hbn03-host00:~# supervisorctl start nl2doca-reload
```

**NVUE Example for Native-L3 Stateful ACLs**

The following is an example of allowing HTTP (TCP) connection originated by the host, where BlueField is hosted, to an HTTP server (with the IP address 11.11.11.11) on an external network. Two sets of ACLs matching with CONNTRACK state must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with CONNTRACK state of NEW, ESTABLISHED and bind it to the SVI in the inbound direction.

- Configure an ACL rule matching TCP/HTTP connection/flow details with CONNTRACK state of ESTABLISHED and bind it to the SVI in the outbound direction.

Native-L3 stateful ACLs should be bound to an SVI interface. In this example, SVI interface is `vlan101`.

1. Configure the ingress ACL rule:

```bash
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 action permit
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack new
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-ip 11.11.11.11/32
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-port 80
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type ipv4
```

2. Bind this ACL to the SVI interface in the inbound direction:

```bash
root@hbn03-host00:~# nv set interface vlan101 acl allow_tcp_conn_from_host inbound
```
3. Configure the egress ACL rule:

```bash
root@hbn03-host00:~# nv config apply
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 action permit
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match conntrack established
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match ip protocol tcp
root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server type ipv4
root@hbn03-host00:~# nv config apply
```

4. Bind this ACL to the SVI interface in the outbound direction:

```bash
root@hbn03-host00:~# nv set interface vlan101 acl allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

---

**Note**

If virtual router redundancy (VRR) is set, L3 stateful ACLs must be bound to all the related SVI interfaces. For example, if VRR is configured on SVI `vlan101` as follows in the `/etc/network/interfaces` file:

```plaintext
auto vlan101
iface vlan101
    address 45.3.1.2/24
    address-virtual 00:00:5e:00:01:01 45.3.1.1/24
    vlan-raw-device br_default
    vlan-id 101
```

With this configuration, two SVI interfaces, `vlan101` and `vlan101-v0` would be created in the system:
Flat Files (cl-acltool) Example for Native-L3 Stateful ACLs

For the same NVUE example for Native-L3 stateful ACLs cited above (HTTP server at IP address 11.11.11.11 on an external network), the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i` to install the rules in BlueField hardware.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the SVI followed by invoking `cl-acltool -i`:

```
[jptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface vlan101 ##
-t filter -A FORWARD -i vlan101 -p tcp -d 11.11.11.11/32 --dport 80 -m conntrack --ctstate EST,NEW -m connmark ! --mark 7998 -j CONNMARK --set-mark 7999
-t filter -A FORWARD -i vlan101 -p tcp --dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT
```

In this case, stateful ACLs must be bound to both SVI interfaces (vlan101 and vlan101-v0). In the stateful ACL described in the current section, the binding would be:

```
root@hbn03-host00:~# ip -br addr show | grep vlan101
vlan101@br_default UP 45.3.1.2/24 fe80::204:4bff:fe8a:f100/64
vlan101-v0@vlan101 UP 45.3.1.1/24 metric 1024
fe80::200:5eff:fe00:101/64

root@hbn03-host00:~# nv set interface vlan101,vlan101-v0 acl allow_tcp_conn_from_host inbound
root@hbn03-host00:~# nv set interface vlan101,vlan101-v0 acl allow_tcp_resp_from_server outbound
root@hbn03-host00:~# nv config apply
```
2. Configure an egress ACL rule matching the TCP flow and CONNTRACK state of ESTABLISHED, RELATED under /etc/cumulus/acl/policy.d/stateful_acl.rules file with the egress interface as SVI followed by invoking cl-acctool -i:

```
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on interface vlan101 ##
-t filter -A FORWARD -o vlan101 -p tcp -s 11.11.11.11/32 --sport 80 -m conntrack --ctstate EST -j CONNMARK --set-mark 7998
-t filter -A FORWARD -o vlan101 -p tcp -s 11.11.11.11/32 --sport 80 -m conntrack --ctstate EST -j ACCEPT
```

Note
As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (-j CONNMARK --set-mark <value>) for egress ACL rules are protocol dependent: 7998 for TCP, 7996 for UDP, and 7994 for ICMP.

NVUE Example for EVPN-L2 Stateful ACLs

The following is an example allowing HTTP (TCP) connection originated by the host, hosting BlueField, to an HTTP server (with the IP address 192.168.5.5) accessible on the EVPN bridged network (L2 stretch). Two sets of ACLs matching with CONNTRACK state

Note
As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (-j CONNMARK --set-mark <value>) for ingress ACL rules are protocol dependent: 7999 for TCP, 7997 for UDP, and 7995 for ICMP.
must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of NEW, ESTABLISHED, and bind it to the host interface in the inbound direction

- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of ESTABLISHED, and bind it to the host interface in the outbound direction

EVPN-L2 stateful ACLs should be bound to a host interface. In this example, the host interface is pf1vf7_sf.

1. Configure the ingress ACL rule:

   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 action permit
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack new
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack established
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-ip 192.168.5.5/32
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-port 80
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip protocol tcp
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type ipv4

2. Bind this ACL to the host interface in the inbound direction:

   root@hbn03-host00:~# nv set interface pf1vf7_sf acl allow_tcp_conn_from_host inbound
   root@hbn03-host00:~# nv config apply

3. Configure the egress ACL rule:

   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 action permit
   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match conntrack established
   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match ip protocol tcp
4. Bind this ACL to the host interface in the outbound direction:

```
root@hbn03-host00:~# nv set interface pf1vf7_sf acl allow_tcp Resp_from_server outbound
root@hbn03-host00:~# nv config apply
```

### Flat Files (cl-acltool) Example for EVPN-L2 Stateful ACLs

For the same NVUE EPVN-L2 stateful ACLs example cited above (HTTP server at IP address 192.168.5.5 accessible over bridged network), the following are the corresponding ACL rules which must be configured under `/etc/cumulus/acl/policy.d/<rule_name.rules>` followed by invoking `cl-acltool -i`.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the ingress interface as the host representor of BlueField, followed by invoking `cl-acltool -i`:

```
[jptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface pf1vf7_sf ##
-A FORWARD -m physdev --physdev-in pf1vf7_sf -p tcp -d 192.168.5.5/32 --dport 80 -m conntrack --ctstate EST,NEW -m connmark ! --mark 9998 -j CONNMARK --set-mark 9999
-A FORWARD -m physdev --physdev-in pf1vf7_sf -p tcp -d 192.168.5.5/32 --dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT
```

---

**Note**

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-`
2. Configure an egress ACL rule matching with TCP and CONNTRACK state of ESTABLISHED, RELATED under `/etc/cumulus/acl/policy.d/stateful_acl.rules` with the egress interface as the host representor of BlueField, followed by invoking `cl-acltool -i`:

```bash
[iptables]
## ACL allow_tcp_resp_from_server in dir outbound on interface pf1vf7_sf ##
-t filter -A FORWARD -m physdev --physdev-out pf1vf7_sf -p tcp -s 192.168.5.5/32 --sport 80 -m conntrack --ctstate EST -j CONNMARK --set-mark 9998
-t filter -A FORWARD -m physdev --physdev-out pf1vf7_sf -p tcp -s 192.168.5.5/32 --sport 80 -m conntrack --ctstate EST -j ACCEPT
```

**Note**

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`-j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 9998 for TCP, 9996 for UDP, and 9994 for ICMP.

**NVUE Example for EVPN-L3 Stateful ACLs**

The following is an example allowing an HTTP (TCP) connection originated by the host, hosting BlueField, to an HTTP server (with the IP address 21.1.1.2) accessible on the EVPN routed network (EVPN Symmetric Routing). Two sets of ACLs matching with CONNTRACK state must be configured for a CONNTRACK entry to be established in the kernel which would be offloaded to hardware:

- Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of NEW, ESTABLISHED, and bind it to the host interface in the inbound direction
Configure an ACL rule matching TCP/HTTP connection/flow details with a CONNTRACK state of ESTABLISHED, and bind it to the host interface in the outbound direction.

EVPN-L3 stateful ACLs should be bound to an SVI interface. In this example, the SVI interface is vlan105.

1. Configure the ingress ACL rule:

   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 action permit
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack new
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match conntrack established
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-ip 21.1.1.2/32
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip dest-port 80
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host rule 11 match ip protocol tcp
   root@hbn03-host00:~# nv set acl allow_tcp_conn_from_host type ipv4

2. Bind this ACL to the host interface in the inbound direction:

   root@hbn03-host00:~# nv set interface vlan105 acl allow_tcp_conn_from_host inbound
   root@hbn03-host00:~# nv config apply

3. Configure the egress ACL rule:

   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 action permit
   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match conntrack established
   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server rule 21 match ip protocol tcp
   root@hbn03-host00:~# nv set acl allow_tcp_resp_from_server type ipv4

4. Bind this ACL to the host interface in the outbound direction:

   root@hbn03-host00:~# nv set interface vlan105 acl allow_tcp_resp_from_server outbound
Flat Files (cl-acltool) Example for EVPN-L3 Stateful ACLs

For the same NVUE EVPN-L3 stateful ACLs example cited under "NVUE Example for EVPN-L3 Stateful ACLs" (HTTP server at IP address 21.1.1.2 accessible over EVPN routed overlay network), the following are the corresponding ACL rules which must be configured under /etc/cumulus/acl/policy.d/<rule_name.rules> followed by invoking cl-acltool -i.

1. Configure an ingress ACL rule matching with TCP flow details and CONNTRACK state of NEW, ESTABLISHED under /etc/cumulus/acl/policy.d/stateful_acl.rules file with the ingress interface as the SVI interface, followed by invoking cl-acltool -i:

```
[iptables]
## ACL allow_tcp_conn_from_host in dir inbound on interface vlan105 ##
-t filter -A FORWARD -i vlan105 -p tcp -d 21.1.1.2/32 --dport 80 -m conntrack --ctstate EST,NEW -m connmark ! --mark 7998 -j CONNMARK --set-mark 7999
-t filter -A FORWARD -i vlan105 -p tcp -d 21.1.1.2/32 --dport 80 -m conntrack --ctstate EST,NEW -j ACCEPT
```

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (-j CONNMARK --set-mark <value>) for ingress ACL rules are protocol dependent: 7999 for TCP, 7997 for UDP, and 7995 for ICMP.

2. Configure an egress ACL rule matching with TCP and CONNTRACK state of ESTABLISHED, RELATED under /etc/cumulus/acl/policy.d/stateful_acl.rules file with the egress interface as the SVI interface, followed by invoking cl-acltool -i:

```
[iptables]
```
DHCP Relay on HBN

DHCP is a client server protocol that automatically provides IP hosts with IP addresses and other related configuration information. A DHCP relay (agent) is a host that forwards DHCP packets between clients and servers. DHCP relays forward requests and replies between clients and servers that are not on the same physical subnet.

DHCP relay can be configured using either flat file (supervisord configuration) or through NVUE.

Configuration

HBN is a non-systemd based container. Therefore, the DHCP relay must be configured as explained in the following subsections.

Flat File Configuration (Supervisord)

The HBN initialization script installs default configuration files on BlueField in `/var/lib/hbn/etc/supervisor/conf.d/`. BlueField directory is mounted to `/etc/supervisor/conf.d` which achieves configuration persistence.

## ACL allow_tcp_resp_from_server in dir outbound on interface vlan105 ##
- `t filter -A FORWARD -o vlan105 -p tcp -s 21.1.1.2/32 --sport 80 -m conntrack --ctstate EST -j CONNMARK --set-mark 7998`
- `t filter -A FORWARD -o vlan105 -p tcp -s 21.1.1.2/32 --sport 80 -m conntrack --ctstate EST -j ACCEPT`

Note

As shown above, an additional rule must be configured with CONNMARK action. The CONNMARK values (`j CONNMARK --set-mark <value>`) for egress ACL rules are protocol dependent: 7998 for TCP, 7996 for UDP, and 7994 for ICMP.
By default, DHCP relay is disabled. Default configuration applies to one instance of DHCPv4 relay and DHCPv6 relay in the default VRF.

**NVUE Configuration**

The user can use NVUE to configure and maintain DHCPv4 and DHCPv6 relays with CLI and REST API. NVUE generates all the required configurations and maintains the relay service.

**DHCPv4 Relay Configuration**

**NVUE Example**

The following configuration starts a relay service which listens for the DHCP messages on p0_sf, p1_sf, and vlan482 and relays the requests to DHCP server 10.89.0.1 with gateway-interface as lo.

```
nv set service dhcp-relay default gateway-interface lo
nv set service dhcp-relay default interface p0_sf
nv set service dhcp-relay default interface p1_sf
nv set service dhcp-relay default interface vlan482 downstream
nv set service dhcp-relay default server 10.89.0.1
```

**Flat Files Example**

```
[program: isc-dhcp-relay-default]
command = /usr/sbin/dhcrelay --nl -d -i p0_sf -i p1_sf -id vlan482 -U lo 10.89.0.1
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```
Where:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>Network interface to listen on for requests and replies</td>
</tr>
<tr>
<td>-iu</td>
<td>Upstream network interface</td>
</tr>
<tr>
<td>-id</td>
<td>Downstream network interface</td>
</tr>
<tr>
<td>-U [address]%%i fname</td>
<td>Gateway IP address interface. Use %% for IP%%ifiename. % is used as an escape character.</td>
</tr>
<tr>
<td>--loglevel-debug</td>
<td>Debug logging. Location: /var/log/syslog.</td>
</tr>
<tr>
<td>-a</td>
<td>Append an agent option field to each request before forwarding it to the server with default values for circuit-id and remote-id</td>
</tr>
<tr>
<td>-r remote-id</td>
<td>Set a custom remote ID string (max of 255 chars). To use this option, you must also enable the -a option.</td>
</tr>
<tr>
<td>--use-pif-circuit-id</td>
<td>Set the underlying physical interface which receives the packet as the circuit-id. To use this option you must also enable the -a option.</td>
</tr>
</tbody>
</table>

**DHCPv4 Relay Option 82**

**NVUE Example**

The following NVUE command is used to enable option 82 insertion in DHCP packets with default values:

```
nv set service dhcp-relay default agent enable on
```

To provide a custom remote-id (e.g., host10) using NVUE:

```
nv set service dhcp-relay default agent remote-id host10
```
To use the underlying physical interface on which the request is received as circuit-id using NVUE:

```plaintext
nv set service dhcp-relay default agent use-pif-circuit-id enable on
```

**Flat Files Example**

```plaintext
[program: isc-dhcp-relay-default]
command = /usr/sbin/dhcrelay --nl -d -i p0_sf -i p1_sf -id vlan482 -U lo -a --use-pif-circuit-id -r host10 10.89.0.1
autostart = true
autorestart = unexpected
startsecs = 3
startretries = 3
exitcodes = 0
stopsignal = TERM
stopwaitsecs = 3
```

**DHCPv6 Relay Configuration**

**NVUE Example**

The following NVUE command starts the DHCPv6 Relay service which listens for DHCPv6 requests on vlan482 and sends relayed DHCPv6 requests towards p0_sf and p1_sf.

```plaintext
nv set service dhcp-relay6 default interface downstream vlan482
nv set service dhcp-relay6 default interface upstream p0_sf
nv set service dhcp-relay6 default interface upstream p1_sf
```

**Flat Files Example**

```plaintext
[program: isc-dhcp-relay6-default]
command = /usr/sbin/dhcrelay --nl -6 -d -l vlan482 -u p0_sf -u p1_sf
autostart = true
```
Where:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l [address]%%ifname[#index]</td>
<td>Downstream interface. Use %% for IP%%ifname. % is used as escape character.</td>
</tr>
<tr>
<td>-u [address]%%ifname</td>
<td>Upstream interface. Use %% for IP%%ifname. % is used as escape character.</td>
</tr>
<tr>
<td>-6</td>
<td>IPv6</td>
</tr>
<tr>
<td>--loglevel-debug</td>
<td>Debug logging located at /var/log/syslog</td>
</tr>
</tbody>
</table>

**DHCP Relay and VRF Considerations**

DHCP relay can be spawned inside a VRF context to handle the DHCP requests in that VRF. There can only be 1 instance each of DHCPv4 relay and DHCPv6 relay per VRF. To achieve that, the user can follow these guidelines:

- DHCPv4 on default VRF:
  
  ```bash
  /usr/sbin/dhcrelay --nl -i <interface> -U [address]%%<interface> <server_ip>
  ```

- DHCPv4 on VRF:
  
  ```bash
  /usr/sbin/ip vrf exec <vrf> /usr/sbin/dhcrelay --nl -i <interface> -U [address]%%<interface> <server_ip>
  ```
• DHCPv6 on default VRF:

```
/usr/sbin/dhcprelay --nl -6 -l <interface> -u <interface>
```

• DHCPv6 on VRF:

```
/usr/sbin/ip vrf exec <vrf> /usr/sbin/dhcprelay --nl -6 -l <interface> -u <interface>
```

**HBN Service Troubleshooting**

**HBN Container Stuck in init-sfs**

The HBN container starts as init-sfs and should transition to doca-hbn within 2 minutes as can be seen using crictl ps. But sometimes it may remain as init-sfs.

This can happen if interface p0_sf is missing. Run the command ip -br link show dev p0_sf in BlueField and inside the container to check if p0_sf is present or not. If its missing, make sure the firmware is upgraded to the latest version. Perform BlueField system-level reset for the new firmware to take effect.

**Host-side PF/VF Down After BlueField Reboot**

In general, the host can use any interface manager to manage host interfaces belonging to BlueField. When the host uses an interface manager other than Netplan or NetworkManager, some ports may remain down after BlueField reboot.

Apply the following workaround if interfaces stay down:

1. Restart openibd:
2. Recreate SR-IOV interfaces if they are needed.

3. Replay interface config. For example:

   - If using ifupdown2:

     ```bash
     ifreload -a
     ```

   - If using Netplan:

     ```bash
     netplan apply
     ```

**BGP Session not Establishing**

One of the main causes of a BGP session not getting established is a mismatch in MTU configuration. Make sure the MTU on all interfaces is the same. For example, if BGP is failing on p0, check and verify that there is a matching MTU value for p0, p0_sf_r, p0_sf, and the remote peer of p0.

**Generating Support Dump**

HBN support dump can be generated using the `cl-support` command, inside the HBN container:

```
root@bf2:/tmp# cl-support
```

Please send `/var/support/cl_support_bf2-s02-1-ipmi_20221025_180508.txz` to Cumulus support

The generated dump would be available in `/var/support` in the HBN container and would contain any process core dump as well as log files.
The `/var/support` directory is also mounted on the BlueField Arm side at `/var/lib/hbn/var/support`.

**SFC Troubleshooting**

To troubleshoot flows going through SFC interfaces, the first step is to disable the `nl2doca` service in the HBN container:

```
root@bf2:/tmp# supervisorctl stop nl2doca
nl2doca: stopped
```

Stopping `nl2doca` effectively stops hardware offloading and switches to software forwarding. All packets would appear on `tcpdump` capture on BlueField interfaces.

`tcpdump` can be performed on SF interfaces as well as VLAN, VXLAN, and uplinks to determine where a packet gets dropped or which flow a packet is taking.

**General nl2doca Troubleshooting**

The following steps can be used to make sure the `nl2doca` daemon is up and running:

1. Make sure there are no errors in the `nl2doca` log file at `/var/log/hbn/nl2docad.log`.

2. To check the status of the `nl2doca` daemon under supervisor, run:

```
supervisorctl status nl2doca
```

3. Use `ps` to check that the actual `nl2doca` process is running:

```
ps -eaf | grep nl2doca
root    18   1  0 06:31 ? 00:00:00 /bin/bash /usr/bin/nl2doca-docker-start
root   1437  18  0 06:31 ? 00:05:49 /usr/sbin/nl2docad
```

4. The core file should be in `/var/support/core/`.
5. Check if the `/cumulus/nl2docad/run/stats/punt` is accessible. Otherwise, `nl2doca` may be stuck and should be restarted:

```
supervisorctl restart nl2doca
```

**nl2doca Offload Troubleshooting**

If a certain traffic flow does not work as expected, disable `nl2doca` (i.e., disable hardware offloading):

```
supervisorctl stop nl2doca
```

With hardware offloading disabled, you can confirm it is an offloading issue if the traffic starts working. If it is not an offloading issue, use `tcpdump` on various interfaces to see where the packet gets dropped.

Offloaded entries can be checked in following files, which contain the programming status of every IP prefix and MAC address known to system.

- Bridge entries are available in the file `/cumulus/nl2docad/run/software-tables/17`. It includes all the MAC addresses in the system including local and remote MAC addresses.

**Example format:**

```
- flow-entry: 0xaaab0cef4190
  flow-pattern:
    fid: 112
    dst mac: 00:00:5e:00:01:01
  flow-actions:
    SET VRF: 2
    OUTPUT-PD-PORT: 20(TO_RTR_INTF)
  STATS:
    pkts: 1719
    bytes: 191286
```
- **Router entries are available in the file** `/cumulus/nl2docad/run/software-tables/18`. It includes all the IP prefixes known to the system.

**Example format for Entry with ECMP:**

```plaintext
Entry with ECMP:
- flow-entry: 0xaaaada723700
  flow-pattern:
    IPV6: LPM
    VRF: 0
    destination-ip: ::/0
  flow-actions:
    ECMP: 2
    STATS:
      pkts: 0
      bytes: 0

Entry without ECMP:
- flow-entry: 0xaaaada7e1400
  flow-pattern:
    IPV4: LPM
    VRF: 0
    destination-ip: 60.1.0.93/32
  flow-actions:
    SET FID: 200
    SMAC: 00:04:4b:a7:88:00
    DMAC: 00:03:00:08:00:12
    OUTPUT-PD-PORT: 19(TO_BR_INTF)
  STATS:
    pkts: 0
    bytes: 0
```

- **ECMP entries are available in the file** `/cumulus/nl2docad/run/software-tables/19`. It includes all the next hops in the system.

**Example format:**

```plaintext
- ECMP: 2
  ref-count: 2
  num-next-hops: 2
  entries:
```
To check counters for packets going to the kernel, run:

```
cat /cumulus/nl2docad/run/stats/punt
PUNT miss pkts:3154 bytes:312326
PUNT miss drop pkts:0 bytes:0
PUNT control pkts:31493 bytes:2853186
PUNT control drop pkts:0 bytes:0
ACL PUNT pkts:68 bytes:7364
ACL drop pkts:0 bytes:0
```

For a specific type of packet flow, programming can be referenced in block specific files. The typical flow is as follows:

For example, to check L2 EVPN ENCAP flows for remote MAC 8a:88:d0:b1:92:b1 on port pf0vf0_sf, the basic offload flow should look as follows: RxPort (pf0vf0_sf) -> BR (Overlay) -> RTR (Underlay) -> BR (Underlay) -> TxPort (one of the uplink p0_sf or p1_sf based on ECMP hash).

Step-by-step procedure:


2. Check for the RxPort (pf0vf0_sf):

```
Interface: pf0vf0_sf
    PD PORT: 6
    HW PORT: 16
    NETDEV PORT: 11
    Bridge-id: 61
    Untagged FID: 112
```

FID 112 is given to the receive port.
3. Check the bridge table file /cumulus/nl2docad/run/software-tables/17 with destination MAC 8a:88:d0:b1:92:b1 and FID 112:

```
flow-pattern:
  fid: 112
  dst mac: 8a:88:d0:b1:92:b1
flow-actions:
  VXLAN ENCAP:
    ENCAP dst ip: 6.0.0.26
    ENCAP vni id: 1000112
  SET VRF: 0
  OUTPUT-PD-PORT: 20(TO_RTR_INTF)
STATS:
  pkts: 100
  bytes: 10200
```

4. Check the router table file /cumulus/nl2docad/run/software-tables/18 with destination IP 6.0.0.26 and VRF 0:

```
flow-pattern:
  IPV4: LPM
  VRF: 0
  ip dst: 6.0.0.26/32
flow-actions:
  ECMP: 1
  OUTPUT PD PORT: 2(TO_BR_INTF)
STATS:
  pkts: 300
  bytes: 44400
```

5. Check the ECMP table file /cumulus/nl2docad/run/software-tables/19 with ECMP 1:

```
- ECMP: 1
  ref-count: 7
  num-next-hops: 2
  entries:
    - { index: 0, fid: 4100, src mac: 'b8:ce:f6:99:49:6a', dst mac: '00:02:00:00:00:2f' }
```
6. The ECMP hash calculation picks one of these paths for next-hop rewrite. Check bridge table file for them (fid=4100, dst mac: 00:02:00:00:00:2f or fid=4115, dst mac: 00:02:00:00:00:33):

```
- { index: 1, fid: 4115, src mac: 'b8:ce:f6:99:49:6b', dst mac: '00:02:00:00:00:33' }
```

This will show the packet going out on the uplink.

**NVUE Troubleshooting**

To check the status of the NVUE daemon, run:

```
supervisorctl status nvued
```

To restart the NVUE daemon, run:

```
supervisorctl restart nvued
```
NVIDIA DOCA Management Service Guide

This guide provides instructions on how to use the DOCA Management Service on top of NVIDIA® BlueField® Networking Platform or ConnectX® Network Adapters.

Note

DOCA DMS service is currently supported at Alpha level.

Introduction

DOCA Management Service (DMS) is a one-stop shop for the user to configure and operate NVIDIA BlueField and ConnectX devices. DMS governs all scripts/tools of NVIDIA with an easy and industry-standard API created by the OpenConfig community. The user can configure BlueField or ConnectX for any mode whether locally (ssh) or remotely (grpc). It makes it easy to migrate and bootstrap any customer for any NVIDIA network device.

DMS exposes configurable BlueField/ConnectX parameters over the external interface to support a management station in an automated configuration of the NVIDIA Network Adapters. The exposed interface presents a uniform approach for BF/CX device configuration and keeps hidden details about the internal tools used for the configuration of BlueField or ConnectX features.

The DMS is a Client-Server architecture. Using a daemon, the service handles the discovery of resources, and is ready to receive commands from clients, the user can use DMSc (DMS Client) which delivers as part of the DMS, or use/create any other client.
The Yang models describe a config tree which is easy to navigate and find any "config leaf" using XPath capabilities. Most gNMI/gNOI protocols are common with the OpenConfig community, utilizing gRPC protocol for transferring the command.

Note

The DOCA Yang model is experimental.

Note

The gNMI Subscribe mechanism for streaming telemetry is not currently supported yet.

Info

DMS can run either on the host machine where BlueField or ConnectX devices are installed or on BlueField Arm itself (when BlueField is operating in DPU mode).

Requirements

DMS requires DOCA to be installed on the target system, where DMS Service will be running:
• DMS for Host - requires DOCA for Host package to be installed on the host system (with doca-networking or doca-all profiles).

• DMS for DPU (BlueField Arm) - requires DOCA Image to be installed on BlueField Arm.

Please follow these instructions to install DOCA: NVIDIA DOCA Installation Guide for Linux.

**Note**

DMS supports only Linux-based environments today.

---

**Service Deployment**

DMS has 3 major components:

• **DMSD – Server** – DMS server inside the BlueField or on the host with an NVIDIA PCIe device

• **DMSC – Client** – DOCA provides OpenConfig client. Customers can choose to use this client, any other open-source client, or develop their own (gRPC-based) client.

• **Yang files** – Yang model files contain the data model used to configure the BlueField device, NVIDIA-specific extension to common OpenConfig YANG Models.

OpenConfig consists of 2 main protocols:

• **gNMI – gRPC Network Management Interface**, protocol to configure of network device.

• **gNOI – gRPC Network Operations Interface**, a protocol to perform operational commands on network device (i.e., provision, upgrade, reboot).

The following is an architectural diagram of DMS:
The following diagram presents the DMS mode of operation, as the DMS client can operate from anywhere:

1. Both DMS client and server components are deployed on the Host
2. Both DMS client and server components are deployed on DPU (BlueField Arm)
3. DMS server component is deployed on the Host, while DMS client is deployed remotely (connecting to DMS server over management network)
4. DMS server component is deployed on DPU (BlueField Arm), while DMS client is deployed remotely (connecting to DMS server over management network)

DMSD is a systemd service installed on the DPU by default with the BFB-Bundle and can be enable/disabled using `systemctl`. DMSD can be accessed using the command `dmscli` and
provided the dmsd user password (default is the root OS password). A systemd template is provided on host packages.

**Configuration**

To see the full list of flags, use the help flag (i.e., dmsd -help, dmsd -h).

**General Flags**

- **-bind_address <string>** – Bind to <address>:<port> or just :<port> (default is :9339). Can be localhost for local use case, or an IP address for remote use case.

- **-v <value>** – log level for V logs

- **-target_pci <string>** – The target PCIe address (i.e., 03:00). Auto-select if only one NVIDIA network device is present; otherwise, the PCIe address must be specified.

**Security Flags**

- **-auth string** – this flag has 3 options:
  - **Shadow**
    - Zero-touch, admin not required to create any dedicated additional user for DMS (re-use OS user)
    - Read the hashed password in real time on each client request
    - Use flags -username -shadow
    - Example: -username root -shadow /etc/shadow/
    - To disable: -noauth flag
  - **Credentials**
    - Admin must set a strong password
    - Use flags -username -password
• Example: -username root -password 123456

• To disable: -noauth flag

• Can leave password flag empty to invoke prompt for password at demon boot

• Certificate File

• The most secure option, based on (m)TLS

• Example: -ca /tmp/ca.crt -ca_key /tmp/ca.key

• To disable: -notls option

**Provisioning Flags**

• -target_pci <string> – The target PCIe address (i.e., 03:00). Auto-select if only one NVIDIA network device is present; otherwise, the PCIe address must be specified.

• -image_folder <string> – Specify image install folder. Can copy images directly to the folder to avoid transfer over the net. Default create folder: /tmp/dms.

• -chunk_size_ack <uint> – The chunk size of the image to respond with a transfer response in bytes (default: 12000000)

• -exec_timeout <uint> – The maximum execution timeout in seconds for a command if not responding (not printing to stdout); 0 (default) is unlimited

**Description**

**gNMI Command**

In DMSC, the gNMI part is powered by the GNMIC project.
Prompt mode with autocomplete options can be invoked using the command `prompt`. It can be accessed using the command `dmscli` and provided the `dmsd` user password (default is the root OS password).

**Get Supported Paths**

```
dmsc --file /opt/mellanox/doca/service/dms/yang path --types --descr
```

`/interfaces/interface[name=*]/config/enabled (type=boolean)`

  This leaf contains the configured, desired state of the interface.

  Systems that implement the IF-MIB use the value of this leaf in the 'running' datastore to set IF-MIB.ifAdminStatus to 'up' or 'down' after an ifEntry has been initialized, as described in RFC 2863.

  Changes in this leaf in the 'running' datastore are reflected in ifAdminStatus, but if ifAdminStatus is changed over SNMP, this leaf is not affected.

`/interfaces/interface[name=*]/config/mtu (type=uint16)`

  Set the max transmission unit size in octets for the physical interface. If this is not set, the mtu is set to the operational default -- e.g., 1514 bytes on an Ethernet interface.

`/interfaces/interface[name=*]/config/type (type=identityref)`

  The type of the interface.

  When an interface entry is created, a server MAY initialize the type leaf with a valid value, e.g., if it is possible to derive the type from the name of the interface.
If a client tries to set the type of an interface to a value that can never be used by the system, e.g., if the type is not supported or if the type does not match the name of the interface, the server MUST reject the request.

A NETCONF server MUST reply with an rpc-error with the error-tag 'invalid-value' in this case.

```
/interfaces/interface[name=\*]/ethernet/nvidia/config/inter-packet-gap (type=uint8)
    Inter packet gap configuration, in 4B unit
/interfaces/interface[name=\*]/ethernet/nvidia/config/rate-limit (type=uint16)
    The percentage of bandwidth, in permile units, to be used on the port.
/interfaces/interface[name=\*]/name (type=leafref)
    References the name of the interface
/interfaces/interface[name=\*]/nvidia/cc/config/priority[id=\*]/id (type=leafref)
    Enable CC algo slot execution.
/interfaces/interface[name=\*]/nvidia/cc/config/priority[id=\*]/np_enabled (type=boolean)
    Enable CC NP for a given priority on the interface
/interfaces/interface[name=\*]/nvidia/cc/config/priority[id=\*]/rp_enabled (type=boolean)
    Enable CC RP for a given priority on the interface
/interfaces/interface[name=\*]/nvidia/cc/slot[id=\*]/config/enabled (type=boolean)
    Enable a CC algo slot execution.
/interfaces/interface[name=\*]/nvidia/cc/slot[id=\*]/id (type=leafref)
    CC algo slot ID.
/interfaces/interface[name=\*]/nvidia/cc/slot[id=\*]/param[id=\*]/config/value (type=algo_param_value)
    Parameter value within the CC algo slot.
/interfaces/interface[name=\*]/nvidia/cc/slot[id=\*]/param[id=\*]/id (type=leafref)
    Parameter ID within the CC algo slot.
/interfaces/interface[name=\*]/nvidia/qos/config/pfc (type=boolean)
    Enables PFC
/interfaces/interface[name=\*]/nvidia/qos/config/priority[id=\*]/id (type=prio)
    Priority id.
/interfaces/interface[name=\*]/nvidia/qos/config/trust-mode (type=identityref)
    Trust mode for the interface QoS.
/interfaces/interface[name=\*]/nvidia/roce/config/adaptive-retransmission (type=boolean)
    Enable adaptive retransmission
/interfaces/interface[name=\*]/nvidia/roce/config/adaptive-routing-force (type=boolean)
    Force adaptive routing even if feature was not negotiated between a requestor and responder.
/interfaces/interface[name=\*]/nvidia/roce/config/rtt-resp-dscp (type=uint8)
    Defines the DSCP fixed value used if mode is set to FIXED.
/interfaces/interface[name=\*]/nvidia/roce/config/rtt-resp-dscp-mode (type=identityref)
    Defines the method for setting DSCP in RTT response packets.
/interfaces/interface[name=\*]/nvidia/roce/config/slow-restart (type=boolean)
    Enable slow restart when congestion
/interfaces/interface[name=\*]/nvidia/roce/config/slow-restart-idle (type=boolean)
    Enable slow restart when idle
```
Get Request

Get requests happen in real-time without cache. Get command require providing the
Yang Xpath as described in the following:

dmsc <flags> get --path /interfaces/interface[name=p0]/config/mtu
[
  {
    "source": "localhost:9339",
    "timestamp": 1712485149723248511,
    "time": "2024-04-07T10:19:09.723248511Z",
    "updates": [
      {
        "Path": "interfaces/interface[name=p0]/config/mtu",
        "values": {
          "interfaces/interface/config/mtu": "1500"
        }
      }
    ]
  }
]
Info

To insert params in the path, as an indication of the interface name (p0).

Set Request

Set requests happen immediately, invoking tools to configure the OS.

Set commands require providing Yang Xpath as described in the following:

```bash
dmsc <flags> set --update /interfaces/interface[name=p0]/config/mtu::int::9216
{
  "source": "localhost:9339",
  "time": "1970-01-01T00:00:00Z",
  "results": [
    {
      "operation": "UPDATE",
      "path": "interfaces/interface[name=p0]/config/mtu"
    }
  ]
}
```

Info

To insert params in the path, as an indication of the interface name (p0).

Note
The value provided must be separated by value type and char.

Note

Currently, only the --update flag is supported in set.

Note

Some leafs' updates take effect only after system reboot. Refer to gNOI system reboot for information.

It is also possible to invoke a command JSON list:

dmsc <flags> set --request-file req.json

req.json example:

```json
{
    "updates": [
        {
            "path": "/interfaces/interface[name=p0]/config/mtu",
            "value": 9216,
            "encoding": "uint"
        },
        {
            "path": "/interfaces/interface[name=p0]/config/enabled",
            "value": true,
            "encoding": "bool"
        }
    ]
}
```
gNOI Commands

In DMSc, the gNOI part is powered by GNOIC project, for full docs refer to GNOIC docs.

```
dmsc -a localhost --port 9339 --tls-cert client.crt --tls-key client.key <command>
```

Prompt mode with autocomplete options can be invoked using the command `prompt`.

All commands are blocking unless specified otherwise.

**OS**

The following subsections present actions for provisioning a new DOCA Image (BFB) or firmware on BlueField.

**Install**

This command transmits the file from the client to the server and authenticates the file's validity:

```
dmsc <flags> os install --version <free_text_version> --pkg <bfb|cfg|fw path>
dmsc <flags> os install --version 2_7_0 --pkg DOCA_2.7.0_Ubuntu.bfb
dmsc <flags> os install --version 2_7_0 --pkg config.cfg
dmsc <flags> os install --version 1_3_5_custom.bfb --pkg custom.bfb
```

The file is saved to the folder specified in the `-image_folder` flag (default `/tmp/dms`) if the file authenticates successfully. The file's extension is autodetected and is written automatically if `none` is provided in the `--version` field. Users may copy the file to the folder manually and invoke the command with file extension to authenticate the file. No file
transfer is initiated if the file already exists in the folder and the version specified with the extension.

**Activate**

Activate the command deploy the BFB bundle/firmware to the hardware:

```bash
dmsc <flags> os activate --version 2_7_0 # Invoke all files under 2_7_0 name
dmsc <flags> os activate --version "2_7_0.bfb;0_0_1.cfg;24_29_0046.fw"
```

The `--version` flag provides a version to search for in the folder specified by the `-image_folder` flag (default `/tmp/dms`). If no extension is provided, the command uses all files under the version name.

To activate separate files, use the `--version` flag separated by semi-colon.

**Note**

After running the command to activate firmware, firmware reset is automatically invoked.

**Verify**

Verify command retrieves the firmware and BFB bundle version:

```bash
dmsc <flags> os verify
```

The return value consists of both versions separated by semi-colon.
**System**

The following subsections provide actions for rebooting the BFB bundle/firmware on the BlueField.

**Reboot Status**

To verify BFB is rebooting:

```
dmsc <flags> system reboot-status
```

The value returned is `false` if the system is active. It is `true` if the system is rebooting. If the status cannot be retrieved, the status appears as a failure and the message field indicates what the issue is.

The flag `--reboot_status_check <string>` checks if firmware reboot is needed:

- If set to `fast` (default), a quick test occurs but not accurate (any config can trigger this flag)
- If set to `strict`, a more accurate test occurs but slower
- If set to `none`, then firmware check is skipped

**Note**

Currently, the BFB bundle can only be retrieved if it was installed via DMS.
To reboot the BlueField Arm and firmware:

```
dmsc <flags> system reboot --delay <uint>s --subcomponent <string> --method <string>
```

This command is non-blocking and returns immediately.

The flag `--delay` specifies the time interval to wait before invoking the reset.

The subcomponent and method are optional. By default, the reboot executes with the lowest reset level and type available.

ℹ️ **Note**

Currently, DMS supports `--subcomponent ARM` `--method <WARM|POWERDOWN>` flags.
NVIDIA OpenvSwitch Acceleration (OVS in DOCA)

Info

Note on naming conventions:

- OVS – Refers to the Open vSwitch distribution within DOCA framework

- OVS-DOCA – Describes the datapath offloading layer (DPIF) that utilizes the DOCA Flow library for offloading tasks. This layer is a component of OVS, along with additional DPIF implementations that facilitate offloading via DPDK or Kernel, known respectively as OVS-DPDK and OVS-Kernel.

Tip

NVIDIA advises utilizing the OVS-DOCA DPIF to maximize efficiency, performance, scalability, and feature support.

Warning
Introduction

Open vSwitch (OVS) is a software-based network technology that enhances virtual machine (VM) communication within internal and external networks. Typically deployed in the hypervisor, OVS employs a software-based approach for packet switching, which can strain CPU resources, impacting system performance and network bandwidth utilization. Addressing this, NVIDIA's Accelerated Switching and Packet Processing (ASAP\textsuperscript{2}) technology offloads OVS data-plane tasks to specialized hardware, like the embedded switch (eSwitch) within the NIC subsystem, while maintaining an unmodified OVS control-plane. This results in notably improved OVS performance without burdening the CPU.

NVIDIA's DOCA-OVS extends the traditional OVS-DPDK and OVS-Kernel data-path offload interfaces (DPIF), introducing OVS-DOCA as an additional DPIF implementation. DOCA-OVS, built upon NVIDIA's networking API, preserves the same interfaces as OVS-DPDK and OVS-Kernel while utilizing the DOCA Flow library with the additional OVS-DOCA DPIF. Unlike the use of the other DPIFs (DPDK, Kernel), OVS-DOCA DPIF exploits unique hardware offload mechanisms and application techniques, maximizing performance and features for NVIDIA NICs and DPUs. This mode is especially efficient due to its architecture and DOCA library integration, enhancing e-switch configuration and accelerating hardware offloads beyond what the other modes can achieve.

NVIDIA OVS installation contains all three OVS flavors. The following subsections describe the three flavors (default is OVS-Kernel) and how to configure each of them.

OVS and Virtualized Devices
When OVS is combined with NICs and DPUs (such as NVIDIA® ConnectX®-6 Lx/Dx and NVIDIA® BlueField®-2 and later), it utilizes the hardware data plane of ASAP$^2$. This data plane can establish connections to VMs using either SR-IOV virtual functions (VFs) or virtual host data path acceleration (vDPA) with virtio.

In both scenarios, an accelerator engine within the NIC accelerates forwarding and offloads the OVS rules. This integrated solution accelerates both the infrastructure (via VFs through SR-IOV or virtio) and the data plane. For DPUs (which include a NIC subsystem), an alternate virtualization technology implements full virtio emulation within the DPU, enabling the host server to communicate with the DPU as a software virtio device.

- When using ASAP$^2$ data plane over SR-IOV virtual functions (VFs), the VF is directly passed through to the VM, with the NVIDIA driver running within the VM.

- When using vDPA, the vDPA driver allows VMs to establish their connections through VirtIO. As a result, the data plane is established between the SR-IOV VF and the standard virtio driver within the VM, while the control plane is managed on the host by the vDPA application.

## OVS-Kernel Hardware Acceleration

OVS-Kernel is the default OVS flavor enabled on your NVIDIA device.

---

### Switchdev Configuration

1. Unbind the VFs:

   `echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind`
2. Change the eSwitch mode from legacy to switchdev on the PF device:

```bash
# devlink dev eswitch set pci/0000:3b:00.0 mode switchdev
```

This also creates the VF representor netdevices in the host OS.

### Note

VMs with attached VFs must be powered off to be able to unbind the VFs.

### Note

Before changing the mode, make sure that all VFs are unbound.

### Info

To return to SR-IOV legacy mode, run:

```bash
# devlink dev eswitch set pci/0000:3b:00.0 mode legacy
```

This also removes the VF representor netdevices.
On OSes or kernels that do not support devlink, moving to switchdev mode can be done using sysfs:

```bash
# echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

3. At this stage, VF representors have been created. To map a representor to its VF, make sure to obtain the representor's `switchid` and `portname` by running:

```bash
# ip -d link show eth4
41: enp0s8f0_1:  <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
```

```
DEFAULT group default qlen 1000
link/ether ba:e6:21:37:bc:37:bc:37:bd:34:bc:3d:bc:3f:bc:3f promiscuity 0 addrgenmode eui64 numtxqueues 10 numrxqueues 10 gso_max_size 65536 gso_max_segs 65535 portname pf0vf1 switchid f4ab580003a1420c
```

Where:

- `switchid` – used to map representor to device, both device PFs have the same `switchid`

- `portname` – used to map representor to PF and VF. Value returned is `pf<X>-vf<Y>`, where `X` is the PF number and `Y` is the number of VF.

4. Bind the VFs:

```bash
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

**Switchdev Performance Tuning**

Switchdev tuning improves its performance.

**Steering Mode**

OVS-kernel supports two steering modes for rule insertion into hardware:
- SMFS (software-managed flow steering) – default mode; rules are inserted directly to the hardware by the software (driver). This mode is optimized for rule insertion.

- DMFS (device-managed flow steering) – rule insertion is done using firmware commands. This mode is optimized for throughput with a small amount of rules in the system.

The steering mode can be configured via sysfs or devlink API in kernels that support it:

- For sysfs:

  ```
  echo <smfs|dmfs> > /sys/class/net/<pf-netdev>/compat/devlink/steering_mode
  ```

- For devlink:

  ```
  devlink dev param set pci/0000:00:08.0 name flow_steering_mode value "<smfs|dmfs>" cmode runtime
  ```

Notes:

- The mode should be set prior to moving to switchdev, by echoing to the sysfs or invoking the devlink command.

- Only when moving to switchdev will the driver use the mode configured.

- Mode cannot be changed after moving to switchdev.

- The steering mode is applicable for switchdev mode only (i.e., it does not affect legacy SR-IOV or other configurations).

### Troubleshooting SMFS

mlx5 debugfs supports presenting Software Steering resources. dr_domain including its tables, matchers and rules. The interface is read-only.
The steering information is dumped in the CSV form in the following format: `<object_type>,<object_ID>,<object_info>,...,<object_info>`.

This data can be read at the following path:
`/sys/kernel/debug/mlx5/<BDF>/steering/fdb/<domain_handle>`.

Example:

```bash
# cat /sys/kernel/debug/mlx5/0000:82:00.0/steering/fdb/dmn_000018644
3100,0x55caa4621c50,0xee802,4,65533
3101,0x55caa4621c50,0xe0100008
```

You can then use the steering dump parser to make the output more human-readable.

The parser can be found in [this GitHub repository](https://github.com).

### vPort Match Mode

OVS-kernel support two modes that define how the rules match on vport.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>Rules match on metadata instead of vport number (default mode).</td>
</tr>
<tr>
<td></td>
<td>This mode is needed to support SR-IOV live migration and dual-port RoCE.</td>
</tr>
</tbody>
</table>

**Note**

New steering rules cannot be inserted/deleted while the dump is being created.
### Mode | Description
--- | ---
Legacy | Rules match on vport number. In this mode, performance can be higher in comparison to Metadata. It can be used only if SR-IOV live migration or dual port RoCE are enabled/used.

Matching on Metadata can have a performance impact.

vPort match mode can be controlled via sysfs:

- Set legacy:

  ```
  echo legacy > /sys/class/net/<PF netdev>/compat/devlink/vport_match_mode
  ```

- Set metadata:

  ```
  echo metadata > /sys/class/net/<PF netdev>/compat/devlink/vport_match_mode
  ```

**Note**

This mode must be set prior to moving to switchdev.

---

**Flow Table Large Group Number**

Offloaded flows, including connection tracking (CT), are added to the virtual switch forwarding data base (FDB) flow tables. FDB tables have a set of flow groups, where each flow group saves the same traffic pattern flows. For example, for CT offloaded flow, TCP and UDP are different traffic patterns which end up in two different flow groups.
A flow group has a limited size to save flow entries. By default, the driver has 15 big FDB flow groups. Each of these big flow groups can save 4M/(15+1)=256k different 5-tuple flow entries at most. For scenarios with more than 15 traffic patterns, the driver provides a module parameter (num_of_groups) to allow customization and performance tuning.

The mode can be controlled via module param or devlink API for kernels that support it:

- **Module param:**

  ```bash
echo <num_of_groups> > /sys/module/mlx5_core/parameters/num_of_groups
  ``)

- **Devlink:**

  ```bash
devlink dev param set pci/0000:82:00.0 name fdb_large_groups cmode driverinit value 20
  ```

**Note**

The change takes effect immediately if no flows are inside the FDB table (no traffic running and all offloaded flows are aged out). And it can be dynamically changed without reloading the driver. If there are still offloaded flows when changing this parameter, it takes effect after all flows have aged out.

---

### Open vSwitch Configuration

OVS configuration is a simple OVS bridge configuration with switchdev.

1. Run the OVS service:
2. Create an OVS bridge (named `ovs-sriov` here):

```bash
ovs-vsctl add-br ovs-sriov
```

3. Enable hardware offload (disabled by default):

```bash
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

4. Restart the OVS service:

```bash
systemctl restart openvswitch
```

This step is required for hardware offload changes to take effect.

5. Add the PF and the VF representor netdevices as OVS ports:

```bash
ovs-vsctl add-port ovs-sriov enp4s0f0
ovs-vsctl add-port ovs-sriov enp4s0f0_0
ovs-vsctl add-port ovs-sriov enp4s0f0_1
```

Make sure to bring up the PF and representor netdevices:

```bash
ip link set dev enp4s0f0 up
ip link set dev enp4s0f0_0 up
ip link set dev enp4s0f0_1 up
```

The PF represents the uplink (wire):
6. Run traffic from the VFs and observe the rules added to the OVS data-path:

```
# ovs-dpctl show
system@ovs-system:
  lookups: hit:0 missed:192 lost:1
  flows: 2
  masks: hit:384 total:2 hit/pkt:2.00
  port 0: ovs-system (internal)
  port 1: ovs-sriov (internal)
  port 2: enp4s0f0
  port 3: enp4s0f0_0
  port 4: enp4s0f0_1
```

```
# ovs-dpctl dump-flows
  eth_type(0x0800),ipv4(frag=no), packets:33, bytes:3234, used:1.196s, actions:2

recirc_id(0),in_port(2),eth(src=e4:1d:2d:a5:f3:9d,dst=e4:11:22:33:44:50),
  eth_type(0x0800),ipv4(frag=no), packets:34, bytes:3332, used:1.196s, actions:3
```

In this example, the ping is initiated from VF0 (OVS port 3) to the outer node (OVS port 2), where the VF MAC is e4:11:22:33:44:50 and the outer node MAC is e4:1d:2d:a5:f3:9d. As previously shown, two OVS rules are added, one in each direction.

**Note**

Users can also verify offloaded packets by adding `type=offloaded` to the command. For example:

```
ovs-appctl dpctl/dump-flows type=offloaded
```
Flow Aging

The aging timeout of OVS is given in milliseconds and can be controlled by running:

```sh
ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

TC Policy

Specifies the policy used with hardware offloading:

- `none` – adds a TC rule to both the software and the hardware (default)
- `skip_sw` – adds a TC rule only to the hardware
- `skip_hw` – adds a TC rule only to the software

Example:

```sh
ovs-vsctl set Open_vSwitch . other_config:tc-policy=skip_sw
```

Note

TC policy should only be used for debugging purposes.

max-revalidator

Specifies the maximum time (in milliseconds) for the revalidator threads to wait for kernel statistics before executing flow revalidation.
n-handler-threads

Specifies the number of threads for software datapaths to use to handle new flows.

```
    ovs-vsctl set Open_vSwitch . other_config:n-handler-threads=4
```

The default value is the number of online CPU cores minus the number of revalidators.

n-revalidator-threads

Specifies the number of threads for software datapaths to use to revalidate flows in the datapath.

```
    ovs-vsctl set Open_vSwitch . other_config:n-revalidator-threads=4
```

vlan-limit

Limits the number of VLAN headers that can be matched to the specified number.

```
    ovs-vsctl set Open_vSwitch . other_config:vlan-limit=2
```

Basic TC Rules Configuration

Offloading rules can also be added directly, and not only through OVS, using the tc utility.
To create an offloading rule using TC:

1. Create an ingress qdisc (queueing discipline) for each interface that you wish to add rules into:

   ```
   tc qdisc add dev enp4s0f0 ingress
   tc qdisc add dev enp4s0f0_0 ingress
   tc qdisc add dev enp4s0f0_1 ingress
   ```

2. Add TC rules using flower classifier in the following format:

   ```
   tc filter add dev NETDEVICE ingress protocol PROTOCOL prio PRIORITY [chain CHAIN] flower [MATCH_LIST] [action ACTION_SPEC]
   ```

   **Note**
   
   A list of supported matches (specifications) and actions can be found in section "Classification Fields (Matches)".

3. Dump the existing tc rules using flower classifier in the following format:

   ```
   tc [-s] filter show dev NETDEVICE ingress
   ```

**SR-IOV VF LAG**

SR-IOV VF LAG allows the NIC’s physical functions (PFs) to get the rules that the OVS tries to offload to the bond net-device, and to offload them to the hardware e-switch.

The supported bond modes are as follows:

- Active-backup
• XOR

• LACP

SR-IOV VF LAG enables complete offload of the LAG functionality to the hardware. The bonding creates a single bonded PF port. Packets from the up-link can arrive from any of the physical ports and are forwarded to the bond device.

When hardware offload is used, packets from both ports can be forwarded to any of the VFs. Traffic from the VF can be forwarded to both ports according to the bonding state. This means that when in active-backup mode, only one PF is up, and traffic from any VF goes through this PF. When in XOR or LACP mode, if both PFs are up, traffic from any VF is split between these two PFs.

**SR-IOV VF LAG Configuration on ASAP2**

To enable SR-IOV VF LAG, both physical functions of the NIC must first be configured to SR-IOV switchdev mode, and only afterwards bond the up-link representors.

The following example shows the creation of a bond interface over two PFs:

1. Load the bonding device and subordinate the up-link representor (currently PF) net-device devices:

   ```
   modprobe bonding mode=802.3ad
   ifup bond0 (make sure ifcfg file is present with desired bond configuration)
   ip link set enp4s0f0 master bond0
   ip link set enp4s0f1 master bond0
   ```

2. Add the VF representor net-devices as OVS ports. If tunneling is not used, add the bond device as well.

   ```
   ovs-vsctl add-port ovs-sriov bond0
   ovs-vsctl add-port ovs-sriov enp4s0f0_0
   ovs-vsctl add-port ovs-sriov enp4s0f1_0
   ```

3. Bring up the PF and the representor netdevices:
Using TC with VF LAG

Both rules can be added either with or without shared block:

- With shared block (supported from kernel 4.16 and RHEL/CentOS 7.7 and above):

```
  tc qdisc add dev bond0 ingress_block 22 ingress
  tc qdisc add dev ens4p0 ingress_block 22 ingress
  tc qdisc add dev ens4p1 ingress_block 22 ingress
```

1. Add drop rule:

```
  # tc filter add block 22 protocol arp parent ffff: prio 3 \\
      flower \\
      dst_mac e4:11:22:11:4a:51 \\
      action drop
```

2. Add redirect rule from bond to representor:
3. Add redirect rule from representor to bond:

```bash
# tc filter add dev ens4f0_0 protocol arp parent ffff: prio 3 \
    flower \
    dst_mac ec:0d:9a:8a:28:42 \
    action mirred egress redirect dev bond0
```

- Without shared block (supported from kernel 4.15 and below):

  1. Add redirect rule from bond to representor:

```bash
# tc filter add dev bond0 protocol arp parent ffff: prio 1 \
    flower \
    dst_mac e4:11:22:11:4a:50 \
    action mirred egress redirect dev ens4f0_0
```

  2. Add redirect rule from representor to bond:

```bash
# tc filter add dev ens4f0_0 protocol arp parent ffff: prio 3 \
    flower \
    dst_mac ec:0d:9a:8a:28:42 \
    action mirred egress redirect dev bond0
```

---

**Classification Fields (Matches)**

OVS-Kernel supports multiple classification fields which packets can fully or partially match.
**Ethernet Layer 2**

- Destination MAC
- Source MAC
- Ethertype

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0), skb_mark(0/0), in_port(eth6), eth(src=00:02:10:40:10:0d, dst=68:54:ed:00:af:de), eth_type(0x800)
packets: 1981, bytes: 206024, used: 0.440s, dp:tc, actions: eth7
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
flower \
dst_mac e4:1d:2d:5d:25:35 \nsrc_mac e4:1d:2d:5d:25:34 \naction mirred egress redirect dev $NIC
```

**IPv4/IPv6**

- Source address
- Destination address
- Protocol
  - TCP/UDP/ICMP/ICMPv6
- TOS
- TTL (HLIMIT)
Supported on all kernels.

In OVS dump flows:

```
ipv4:
ipv4(src=0.0.0.0/0.0.0.0,dst=0.0.0.0/0.0.0.0,proto=17,tos=0/0,ttl=0/0,frag=no)
ipv6:
ipv6(src=::/::,dst=1:1:1:3:1040:1008,label=0/0,proto=58,tclass=0/0x3,hlimit=64),
```

Using TC rules:

```
IPv4:
tc filter add dev $rep parent ffff: protocol ip pref 1 \
   flower \
   dst_ip 1.1.1.1 \
   src_ip 1.1.1.2 \
   ip_proto TCP \
   ip_tos 0x3 \
   ip_ttl 63 \
   action mirred egress redirect dev $NIC

IPv6:
tc filter add dev $rep parent ffff: protocol ipv6 pref 1 \
   flower \
   dst_ip 1:1:1:3:1040:1009 \
   src_ip 1:1:1:3:1040:1008 \
   ip_proto TCP \
   ip_tos 0x3 \
   ip_ttl 63 \
   action mirred egress redirect dev $NIC
```

**TCP/UDP Source and Destination Ports and TCP Flags**

- TCP/UDP source and destinations ports
- TCP flags
Supported on kernel >4.13 and RHEL >7.5.

In OVS dump flows:

TCP: tcp(src=0/0,dst=32768/0x8000),
UDP: udp(src=0/0,dst=32768/0x8000),
TCP flags: tcp_flags(0/0)

Using TC rules:

tc filter add dev $rep parent ffff: protocol ip pref 1 \
flower \
ip_proto TCP \
dst_port 100 \
src_port 500 \
tcp_flags 0x4/0x7 \
action mirred egress redirect dev $NIC

**VLAN**

- ID
- Priority
- Inner vlan ID and Priority

Supported kernels: All (QinQ: kernel 4.19 and higher, and RHEL 7.7 and higher).

In OVS dump flows:

eth_type(0x8100),vlan(vid=2347,pcp=0),

Using TC rules:
Tunnel

- ID (Key)
- Source IP address
- Destination IP address
- Destination port
- TOS (supported from kernel 4.19 and above & RHEL 7.7 and above)
- TTL (support from kernel 4.19 and above & RHEL 7.7 and above)
- Tunnel options (Geneve)

Supported kernels:

- VXLAN: All
- GRE: Kernel >5.0, RHEL 7.7 and above
- Geneve: Kernel >5.0, RHEL 7.7 and above

In OVS dump flows:

```c
 tun(lid=0x5,srcc=121.9.1.1,srcd=131.10.1.1,ttd=0/0,tstp=4789,flags(+key))
```

Using TC rules:

```bash
# tc filter add dev $rep protocol 802.1Q parent ffff: pref 1
flower \\
  vlan_ethy 0x800 \\
  vlan_id 100 \\
  vlan_prio 0 \\
  action mirred egress redirect dev $NIC
QinQ:
# tc filter add dev vxlan100 protocol ip parent ffff: \\
  flower \\
    skip_sw \\
    dst_mac e4:11:22:11:4a:51 \\
    src_mac e4+:11:22:11:4a:50 \\
    enc_src_ip 20.1.11.1 \\
    enc_dst_ip 20.1.12.1 \\
    enc_key_id 100 \\
    enc_dst_port 4789 \\
    action tunnel_key unset \\
    action mirred egress redirect dev ens4f0_0
```

**Supported Actions**

**Forward**

Forward action allows for packet redirection:

- From VF to wire
- Wire to VF
- VF to VF

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0d,dst=68:54:ed:00:af:de),eth_type(0x packets:1981, bytes:206024, used:0.440s, dp:tc, actions:eth7
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1
  flower
  dst_mac e4:1d:2d:5d:25:35
  src_mac e4:1d:2d:5d:25:34
  action mirred egress redirect dev $NIC
```

**Drop**

Drop action allows to drop incoming packets.

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0),skb_mark(0/0),in_port(eth6),eth(src=00:02:10:40:10:0d,dst=68:54:ed:00:af:de),eth_type(0x packets:1981, bytes:206024, used:0.440s, dp:tc, actions:drop
```

Using TC rules:

```
tc filter add dev $rep parent ffff: protocol arp pref 1
  flower
  dst_mac e4:1d:2d:5d:25:35
  src_mac e4:1d:2d:5d:25:34
```
Statistics

By default, each flow collects the following statistics:

- Packets – number of packets which hit the flow
- Bytes – total number of bytes which hit the flow
- Last used – the amount of time passed since last packet hit the flow

Supported on all kernels.

In OVS dump flows:

```
skb_priority(0/0), skb_mark(0/0), in_port(eth6), eth(src=00:02:10:40:10:0d, dst=68:54:ed:00:af:de), eth_type(0x
packets:1981, bytes:206024, used:0.440s, dp:tc, actions:drop
```

Using TC rules:

```
# tc -s filter show dev $rep ingress

filter protocol ip pref 2 flower chain 0
filter protocol ip pref 2 flower chain 0 handle 0x2
eth_type ipv4
ip_proto tcp
src_ip 192.168.140.100
src_port 80
skip_sw
in_hw
action order 1: mirred (Egress Redirect to device p0v11_r) stolen
index 34 ref 1 bind 1 installed 144 sec used 0 sec
Action statistics:
Sent 388344 bytes 2942 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
Tunnels: Encapsulation/Decapsulation

OVS-kernel supports offload of tunnels using encapsulation and decapsulation actions.

- Encapsulation – pushing of tunnel header is supported on Tx
- Decapsulation – popping of tunnel header is supported on Rx

Supported Tunnels:

- VXLAN (IPv4/IPv6) – supported on all Kernels
- GRE (IPv4/IPv6) – supported on kernel 5.0 and above & RHEL 7.6 and above
- Geneve (IPv4/IPv6) – supported on kernel 5.0 and above & RHEL 7.6 and above

OVS configuration:

In case of offloading tunnel, the PF/bond should not be added as a port in the OVS datapath. It should rather be assigned with the IP address to be used for encapsulation.

The following example shows two hosts (PFs) with IPs 1.1.1.177 and 1.1.1.75, where the PF device on both hosts is enp4s0f0, and the VXLAN tunnel is set with VNID 98:

- On the first host:

```
# ip addr add 1.1.1.177/24 dev enp4s0f1
# ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0 type=vxlan options:local_ip=1.1.1.177 options:remote_ip=1.1.1.75 options:key=98
```

- On the second host:

```
# ip addr add 1.1.1.75/24 dev enp4s0f1
# ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0 type=vxlan options:local_ip=1.1.1.75 options:remote_ip=1.1.1.177 options:key=98
```
Info

For a GRE IPv4 tunnel, use `type=gre`. For a GRE IPv6 tunnel, use `type=ip6gre`. For a Geneve tunnel, use `type=geneve`.

Note

When encapsulating guest traffic, the VF's device MTU must be reduced to allow the host/hardware to add the encap headers without fragmenting the resulted packet. As such, the VF's MTU must be lowered by 50 bytes from the uplink MTU for IPv4 and 70 bytes for IPv6.

Tunnel offload using TC rules:

Encapsulation:

```
# tc filter add dev ens4f0_0 protocol 0x806 parent ffff: \
    flower \n    skip_sw \n    dst_mac e4:11:22:11:4a:51 \n    src_mac e4:11:22:11:4a:50 \n    action tunnel_key set \n    src_ip 20.1.12.1 \n    dst_ip 20.1.11.1 \n    id 100 \n    action mirred egress redirect dev vxlan100
```

Decapsulation:

```
# tc filter add dev vxlan100 protocol 0x806 parent ffff: \
    flower \n    skip_sw \n    dst_mac e4:11:22:11:4a:51 \n    src_mac e4:11:22:11:4a:50 \n    enc_src_ip 20.1.11.1 \
```
VLAN Push/Pop

OVS-kernel supports offload of VLAN header push/pop actions:

- Push – pushing of VLAN header is supported on Tx
- Pop – popping of tunnel header is supported on Rx

OVSwitch Configuration

Add a tag=$TAG section for the OVS command line that adds the representor ports. For example, VLAN ID 52 is being used here.

```
# ovs-vsctl add-port ovs-sriov enp4s0f0
# ovs-vsctl add-port ovs-sriov enp4s0f0_0 tag=52
# ovs-vsctl add-port ovs-sriov enp4s0f0_1 tag=52
```

The PF port should not have a VLAN attached. This will cause OVS to add VLAN push/pop actions when managing traffic for these VFs.

Dump Flow Example

```
recirc_id(0),in_port(3),eth(src=e4:11:22:33:44:50,dst=00:02:c9:eb:bb:b2),eth_type(0x0800),ipv4(frag=no), \
    packets:0, bytes:0, used:never, actions:push_vlan(vid=52,pcp=0),2

recirc_id(0),in_port(2),eth(src=00:02:c9:eb:bb:b2,dst=e4:11:22:33:44:50),eth_type(0x8100), \
    vlan(vid=52,pcp=0),encap(eth_type(0x0800),ipv4(frag=no)), packets:0, bytes:0, used:never, \
    actions:pop_vlan,3
```
**VLAN Offload Using TC Rules Example**

```bash
# tc filter add dev ens4f0_0 protocol ip parent ffff: \
   flower \
   skip_sw \
   dst_mac e4:11:22:11:4a:51 \
   src_mac e4:11:22:11:4a:50 
   action vlan push id 100 
   action mirred egress redirect dev ens4f0

# tc filter add dev ens4f0 protocol 802.1Q parent ffff: \
   flower \
   skip_sw \
   dst_mac e4:11:22:11:4a:51 \
   src_mac e4:11:22:11:4a:50 
   vlan_ethtype 0x800 
   vlan_id 100 
   vlan_prio 0 
   action vlan pop 
   action mirred egress redirect dev ens4f0_0

# tc filter add dev ens4f0_0 ingress protocol 802.1Q parent ffff: \
   flower \
   vlan_id 100 
   action vlan pop 
   action tunnel_key set 
   src_ip 4.4.4.1 
   dst_ip 4.4.4.2 
   dst_port 4789 
   id 42 
   action mirred egress redirect dev vxlan0

# tc filter add dev vxlan0 ingress protocol all parent ffff: \
```

**TC Configuration**

Example of VLAN Offloading with popping header on Tx and pushing on Rx using TC rules:

```bash
# tc filter add dev ens4f0_0 ingress protocol 802.1Q parent ffff: \
   flower \
   vlan_id 100 
   action vlan pop 
   action tunnel_key set 
   src_ip 4.4.4.1 
   dst_ip 4.4.4.2 
   dst_port 4789 
   id 42 
   action mirred egress redirect dev vxlan0

# tc filter add dev vxlan0 ingress protocol all parent ffff: \
```
Header Rewrite

This action allows for modifying packet fields.

Ethernet Layer 2

- Destination MAC
- Source MAC

Supported kernels:
- Kernel 4.14 and above
- RHEL 7.5 and above

In OVS dump flows:

```
flower \
    enc_dst_ip 4.4.4.1 \
    enc_src_ip 4.4.4.2 \
    enc_dst_port 4789 \
    enc_key_id 42 \
action tunnel_key unset \
action vlan push id 100 \
action mirred egress redirect dev ens4f0_0
```

Using TC rules:

```
skb_priority(0/0), skb_mark(0/0), in_port(eth6), eth(src=00:02:10:40:10:0d, dst=68:54:ed:00:af:de), eth_type(0x0) packets:1981, bytes:206024, used:0.440s, dptc, actions:
set(eth(src=68:54:ed:00:ab, dst=fa:16:3e:dd:69:09), eth7)
```
IPv4/IPv6

- Source address
- Destination address
- Protocol
- TOS
- TTL (HLIMIT)

Supported kernels:

- Kernel 4.14 and above
- RHEL 7.5 and above

In OVS dump flows:

IPv4:
```
set(eth(src=de:e8:ef:27:5e:45,dst=00:00:01:01:01:01)),
set(ipv4(src=10.10.0.111,dst=10.20.0.122,ttl=63))
```

IPv6:
```
set(ipv6(dst=2001:1:6::92eb:fcbe:f1c8,hlimit=63)),
```

Using TC rules:
```
```
IPv4:
tc filter add dev $rep parent ffff: protocol ip pref 1 
  flower 
  dst_ip 1.1.1.1 
  src_ip 1.1.1.2 
  ip_proto TCP 
  ip_tos 0x3 
  ip_ttl 63 
  pedit ex 
  munge ip src set 2.2.2.1 
  munge ip dst set 2.2.2.2 
  munge ip tos set 0 
  munge ip ttl dec 
  action mirred egress redirect dev $NIC

IPv6:
tc filter add dev $rep parent ffff: protocol ipv6 pref 1 
  flower 
  dst_ip 1::1040:1009 
  src_ip 1::1040:1008 
  ip_proto tcp 
  ip_tos 0x3 
  ip_ttl 63 
  pedit ex 
  munge ipv6 src set 2::2::3:1040:1009 
  munge ipv6 dst set 2::2::3:1040:1008 
  munge ipv6 hlimit dec 
  action mirred egress redirect dev $NIC

⚠️ Note
IPv4 and IPv6 header rewrite is only supported with match on UDP/TCP/ICMP protocols.

TCP/UDP Source and Destination Ports
- TCP/UDP source and destinations ports

Supported kernels:

- Kernel 4.16 and above
- RHEL 7.6 and above

In OVS dump flows:

```plaintext
TCP:
set(tcp(src=32768/0xffff,dst=32768/0xffff)),
UDP:
set(udp(src=32768/0xffff,dst=32768/0xffff)),
```

Using TC rules:

```plaintext
TCP:
tc filter add dev $rep parent ffff: protocol ip pref 1 \   flower \   dst_ip 1.1.1.1 \   src_ip 1.1.1.2 \   ip_proto tcp \   ip_tos 0x3 \   ip_ttl 63 \   pedit ex \   pedit ex munge ip tcp sport set 200 \   pedit ex munge ip tcp dport set 200 \   action mirred egress redirect dev $NIC
UDP:
tc filter add dev $rep parent ffff: protocol ip pref 1 \   flower \   dst_ip 1.1.1.1 \   src_ip 1.1.1.2 \   ip_proto udp \   ip_tos 0x3 
```
VLAN

- ID

Supported on all kernels.

In OVS dump flows:

```plaintext
Set(vlan(vid=2347,pcp=0/0)),
```

Using TC rules:

```plaintext
tc filter add dev $rep parent ffff: protocol 802.1Q pref 1 \  
  flower \  
  vlan_ethtype 0x800 \  
  vlan_id 100 \  
  vlan_prio 0 \  
  action vlan modify id 11 pipe  
  action mirred egress redirect dev $NIC
```

Connection Tracking

The TC connection tracking (CT) action performs CT lookup by sending the packet to netfilter conntrack module. Newly added connections may be associated, via the `ct commit` action, with a 32 bit mark, 128 bit label, and source/destination NAT values.

The following example allows ingress TCP traffic from the uplink representor to `vf1_rep`, while assuring that egress traffic from `vf1_rep` is only allowed on established connections.

```plaintext
ip_ttl 63 \  
pedit ex \  
pedit ex munge ip udp sport set 200  
pedit ex munge ip udp dport set 200  
action mirred egress redirect dev $NIC
```
In addition, mark and source IP NAT is applied.

In OVS dump flows:

```
ct(zone=2,nat)
ct_state(+est+trk)
actions:ct(commit,zone=2,mark=0x4/0xffffffff,nat(src=5.5.5.5))
```

Using TC rules:

```
# tc filter add dev $uplink_rep ingress chain 0 prio 1 proto ip \
   flower \
   ip_proto tcp \
   ct_state -trk \n   action ct zone 2 nat pipe
   action goto chain 2
# tc filter add dev $uplink_rep ingress chain 2 prio 1 proto ip \
   flower \
   ct_state +trk+new \n   action ct zone 2 commit mark 0xbb nat src addr 5.5.5.7 pipe \n   action mirred egress redirect dev $vf1_rep
# tc filter add dev $uplink_rep ingress chain 2 prio 1 proto ip \
   flower \
   ct_zone 2 \n   ct_mark 0xbb \
   ct_state +trk+est \n   action mirred egress redirect dev $vf1_rep

// Setup filters on $vf1_rep, allowing only established connections of zone 2 through, and reverse nat (dst nat in this case)

# tc filter add dev $vf1_rep ingress chain 0 prio 1 proto ip \
   flower \
   ip_proto tcp \
   ct_state -trk \n   action ct zone 2 nat pipe 
   action goto chain 1
# tc filter add dev $vf1_rep ingress chain 1 prio 1 proto ip \
   flower \
   ct_zone 2 \
```
CT Performance Tuning

- Max offloaded connections – specifies the limit on the number of offloaded connections. Example:

```
devlink dev param set pci/${pci_dev} name ct_max_offloaded_conns value $max cmode runtime
```

- Allow mixed NAT/non-NAT CT – allows offloading of the following scenario:

```
  cookie=0x0, duration=21.843s, table=0, n_packets=4838718, n_bytes=241958846, ct_state=-trk,ip,in_port=enp8s0f0 actions=ct(table=1,zone=2)
  cookie=0x0, duration=21.823s, table=1, n_packets=15363, n_bytes=773526, ct_state=+new+trk,ip,in_port=enp8s0f0
```

Example:

```
echo enable > /sys/class/net/<device>/compat/devlink/ct_action_on_nat_conns
```

Forward to Chain (TC Only)

TC interface supports adding flows on different chains. Only chain 0 is accessed by default. Access to the other chains requires using the `goto` action.

In this example, a flow is created on chain 1 without any match and redirect to wire.
The second flow is created on chain 0 and match on source MAC and action goto chain 1.

This example simulates simple MAC spoofing:

```bash
#tc filter add dev $rep parent ffff: protocol all chain 1 pref 1 \    flower \    action mirred egress redirect dev $NIC

#tc filter add dev $rep parent ffff: protocol all chain 1 pref 1 \    flower \    src_mac aa:bb:cc:aa:bb:cc \    action goto chain 1
```

### Port Mirroring: Flow-based VF Traffic Mirroring for ASAP²

Unlike para-virtual configurations, when the VM traffic is offloaded to hardware via SR-IOV VF, the host-side admin cannot snoop the traffic (e.g., for monitoring).

ASAP² uses the existing mirroring support in OVS and TC along with the enhancement to the offloading logic in the driver to allow mirroring the VF traffic to another VF.

The mirrored VF can be used to run traffic analyzer (e.g., tcpdump, wireshark, etc.) and observe the traffic of the VF being mirrored.

The following example shows the creation of port mirror on the following configuration:

```bash
# ovs-vsctl show
09d8a574-9c39-465c-9f16-47d81c12f88a
  Bridge br-vxlan
    Port "enp4s0f0_1"
      Interface "enp4s0f0_1"
    Port "vxlan0"
      Interface "vxlan0"
        type: vxlan
          options: {key="100", remote_ip="192.168.1.14"}
    Port "enp4s0f0_0"
      Interface "enp4s0f0_0"
```
To set `enp4s0f0_0` as the mirror port and mirror all the traffic:

```bash
# ovs-vsctl -- --id=@p get port enp4s0f0_0 \
   -- --id=@m create mirror name=m0 select-all=true output-port=@p \
   -- set bridge br-vxlan mirrors=@m
```

To set `enp4s0f0_0` as the mirror port, only mirror the traffic, and set `enp4s0f0_1` as the destination port:

```bash
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
   -- --id=@p2 get port enp4s0f0_1 \
   -- --id=@m create mirror name=m0 select-dst-port=@p2 output-port=@p1 \
   -- set bridge br-vxlan mirrors=@m
```

To set `enp4s0f0_0` as the mirror port, only mirror the traffic, and set `enp4s0f0_1` as the source port:

```bash
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
   -- --id=@p2 get port enp4s0f0_1 \
   -- --id=@m create mirror name=m0 select-src-port=@p2 output-port=@p1 \
   -- set bridge br-vxlan mirrors=@m
```

To set `enp4s0f0_0` as the mirror port and mirror all the traffic on `enp4s0f0_1`:

```bash
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
   -- --id=@p2 get port enp4s0f0_1
```
To clear the mirror port:

```
ovs-vsctl clear bridge br-vxlan mirrors
```

Mirroring using TC:

- Mirror to VF:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
   flower \
   dst_mac e4:1d:2d:5d:25:35 \
   src_mac e4:1d:2d:5d:25:34 \
   action mirred egress mirror dev $mirror_rep pipe \
   action mirred egress redirect dev $NIC
```

- Mirror to tunnel:

```
tc filter add dev $rep parent ffff: protocol arp pref 1 \
   flower \
   dst_mac e4:1d:2d:5d:25:35 \
   src_mac e4:1d:2d:5d:25:34 \
   action tunnel_key set \
   src_ip 1.1.1.1 \
   dst_ip 1.1.1.2 \
   dst_port 4789 \
   id 768 \
   pipe \
   action mirred egress mirror dev vxlan100 pipe \
   action mirred egress redirect dev $NIC
```

**Forward to Multiple Destinations**
Forwarding to up 32 destinations (representors and tunnels) is supported using TC:

- **Example 1 – forwarding to 32 VFs:**

```plaintext
tc filter add dev $NIC parent ffff: protocol arp pref 1 \  
  flower \  
  dst_mac e4:1d:2d:5d:25:35 \  
  src_mac e4:1d:2d:5d:25:34 \  
  action mirred egress mirror dev $rep0 pipe \  
  action mirred egress mirror dev $rep1 pipe \  
  ... \  
  action mirred egress mirror dev $rep30 pipe \  
  action mirred egress redirect dev $rep31
```

- **Example 2 – forwarding to 16 tunnels:**

```plaintext
tc filter add dev $rep parent ffff: protocol arp pref 1 \  
  flower \  
  dst_mac e4:1d:2d:5d:25:35 \  
  src_mac e4:1d:2d:5d:25:34 \  
  action tunnel_key set src_ip $ip_src dst_ip $ip_dst \  
  dst_port 4789 id 0 nocsum \  
  pipe action mirred egress mirror dev vxlan0 pipe \  
  action tunnel_key set src_ip $ip_src dst_ip $ip_dst \  
  dst_port 4789 id 1 nocsum \  
  pipe action mirred egress mirror dev vxlan0 pipe \  
  ... \  
  action tunnel_key set src_ip $ip_src dst_ip $ip_dst \  
  dst_port 4789 id 15 nocsum \  
  pipe action mirred egress redirect dev vxlan0
```

**Note**

TC supports up to 32 actions.
**Note**

If header rewrite is used, then all destinations should have the same header rewrite.

**Note**

If VLAN push/pop is used, then all destinations should have the same VLAN ID and actions.

### sFlow

sFlow allows for monitoring traffic sent between two VMs on the same host using an sFlow collector.

The following example assumes the environment is configured as described later.

```bash
# ovs-vsctl show
09d8a574-9c39-465c-9f16-47d81c12f88a
  Bridge br-vxlan
    Port "enp4s0f0_1"
      Interface "enp4s0f0_1"
    Port "vxlan0"
      Interface "vxlan0"
        type: vxlan
          options: {key="100", remote_ip="192.168.1.14"}
    Port "enp4s0f0_0"
      Interface "enp4s0f0_0"
    Port "enp4s0f0_2"
      Interface "enp4s0f0_2"
  Port br-vxlan
    Interface br-vxlan
      type: internal
```
To sample all traffic over the OVS bridge:

```bash
# ovs-vsctl --id=@sflow create sflow agent="$SFLOW_AGENT" \
    target="$SFLOW_TARGET:$SFLOW_PORT" \
    header=$SFLOW_HEADER \
    sampling=$SFLOW_SAMPLING polling=10 \
    -- set bridge br-vxlan sflow=@sflow
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFLOW_AGENT</td>
<td>Indicates that the sFlow agent should send traffic from SFLOW_AGENT's IP address</td>
</tr>
<tr>
<td>SFLOW_TARGET</td>
<td>Remote IP address of the sFlow collector</td>
</tr>
<tr>
<td>SFLOW_HEADER</td>
<td>Size of packet header to sample (in bytes)</td>
</tr>
<tr>
<td>SFLOW_SAMPLING</td>
<td>Sample rate</td>
</tr>
</tbody>
</table>

To clear the sFlow configuration:

```bash
# ovs-vsctl clear bridge br-vxlan sflow
```

To list the sFlow configuration:

```bash
# ovs-vsctl list sflow
```

sFlow using TC:

```bash
Sample to VF
tc filter add dev $rep parent ffff: protocol arp pref 1 \
    flower \
    dst_mac e4:1d:2d:5d:25:35 
```
Rate Limit

OVS-kernel supports offload of VF rate limit using OVS configuration and TC.

The following example sets the rate limit to the VF related to representor eth0 to 10Mb/s:

- **OVS:**

```bash
ovs-vsctl set interface eth0 ingress_policing_rate=10000
```

- **TC:**

```bash
tc_filter add dev eth0 root prio 1 protocol ip matchall skip_sw action police rate 10mbit burst 20k
```

Kernel Requirements

This kernel config should be enabled to support switchdev offload.

- **CONFIG_NET_ACT_CSUM** – needed for action csum
- **CONFIG_NET_ACT_PEDIT** – needed for header rewrite
- **CONFIG_NET_ACT_MIRRED** – needed for basic forward

Note

A userspace application is needed to process the sampled packet from the kernel. An example is available on Github.
- `CONFIG_NET_ACT_CT` – needed for CT (supported from kernel 5.6)
- `CONFIG_NET_ACT_VLAN` – needed for action vlan push/pop
- `CONFIG_NET_ACT_GACT`
- `CONFIG_NET_CLS_FLOWER`
- `CONFIG_NET_CLS_ACT`
- `CONFIG_NET_SWITCHDEV`
- `CONFIG_NET_TC_SKB_EXT` – needed for CT (supported from kernel 5.6)
- `CONFIG_NET_ACT_CT` – needed for CT (supported from kernel 5.6)
- `CONFIG_NFT_FLOW_OFFLOAD`
- `CONFIG_NET_ACT_TUNNEL_KEY`
- `CONFIG_NF_FLOW_TABLE` – needed for CT (supported from kernel 5.6)
- `CONFIG_SKB_EXTENSIONS` – needed for CT (supported from kernel 5.6)
- `CONFIG_NET_CLS_MATCHALL`
- `CONFIG_NET_ACT_POLICE`
- `CONFIG_MLX5_ESWITCH`

**VF Metering**

OVS-kernel supports offloading of VF metering (TX and RX) using sysfs. Metering of number of packets per second (PPS) and bytes per second (BPS) is supported.

The following example sets Rx meter on VF 0 with value 10Mb/s BPS:

```
echo 10000000 > /sys/class/net/enp4s0f0/device/sriov/0/meters/rx/bps/rate
echo 65536 > /sys/class/net/enp4s0f0/device/sriov/0/meters/rx/bps/burst
```
The following example sets Tx meter on VF 0 with value 1000 PPS:

```bash
echo 1000 > /sys/class/net/enp4s0f0/device/sriov/0/meters/tx/pps/rate
echo 100 > /sys/class/net/enp4s0f0/device/sriov/0/meters/tx/pps/burst
```

**Note**

Both `rate` and `burst` must not be zero and `burst` may need to be adjusted according to the requirements.

The following counters can be used to query the number dropped packet/bytes:

- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/rx/pps/packets_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/rx/pps/bytes_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/rx/bps/packets_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/rx/bps/bytes_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/tx/pps/packets_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/tx/pps/bytes_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/tx/bps/packets_dropped`
- `cat /sys/class/net/enp8s0f0/device/sriov/0/meters/tx/bps/bytes_dropped`

**Representor Metering**

**Info**

Metering for uplink and VF representors traffic is supported.

Traffic going to a representor device can be a result of a miss in the embedded switch (eSwitch) FDB tables. This means that a packet which arrives from that representor into
the eSwitch has not matched against the existing rules in the hardware FDB tables and
must be forwarded to software to be handled there and is, therefore, forwarded to the
originating representor device driver.

The meter allows to configure the max rate [packets per second] and max burst [packets]
for traffic going to the representor driver. Any traffic exceeding values provided by the
user are dropped in hardware. There are statistics that show the number of dropped
packets.

The configuration of representor metering is done via miss_rl_cfg.

- Full path of the miss_rl_cfg parameter: /sys/class/net//rep_config/miss_rl_cfg
- Usage: echo "<rate> <burst>" > /sys/class/net//rep_config/miss_rl_cfg.
  - rate is the max rate of packets allowed for this representor (in packets/sec
    units)
  - burst is the max burst size allowed for this representor (in packets units)
  - Both values must be specified. Both of their default values is 0, signifying
    unlimited rate and burst.

To view the amount of packets and bytes dropped due to traffic exceeding the user-
provided rate and burst, two read-only sysfs for statistics are available:

- /sys/class/net//rep_config/miss_rl_dropped_bytes – counts how many FDB-miss bytes are
dropped due to reaching the miss limits
- /sys/class/net//rep_config/miss_rl_dropped_packets – counts how many FDB-miss packets are
dropped due to reaching the miss limits

**OVS Metering**

There are two types of meters, kpps (kilobits per second) and pktps (packets per second).
OVS-Kernel supports offloading both of them.

The following example is to offload a kpps meter.

1. Create OVS meter with a target rate:
2. Delete the default rule:

```bash
ovs-ofctl del-flows ovs-sriov
```

3. Configure OpenFlow rules:

```bash
ovs-ofctl -O OpenFlow13 add-meter ovs-sriov meter=1,kbps,band=type=drop,rate=204800

ovs-ofctl del-flows ovs-sriov

ovs-ofctl -O OpenFlow13 add-flow ovs-sriov 'ip,dl_dst=e4:11:22:33:44:50,actions= meter:1,output:enp4s0f0_0'


ovs-ofctl -O OpenFlow13 add-flow ovs-sriov 'arp,actions=normal'
```

Here, the VF bandwidth on the receiving side is limited by the rate configured in step 1.

4. Run iperf server and be ready to receive UDP traffic. On the outer node, run iperf client to send UDP traffic to this VF. After traffic starts, check the offloaded meter rule:

```bash
ovs-appctl dpctl/dump-flows --names type=offloaded

recirc_id(0),in_port(enp4s0f0),eth(dst=e4:11:22:33:44:50),eth_type(0x0800),ipv4(frag=no), packets:11626587, bytes:17625889188, used:0.470s, actions:meter(0),enp4s0f0_0
```

To verify metering, iperf client should set the target bandwidth with a number which is larger than the meter rate configured. Then it should apparent that packets are received with the limited rate on the server side and the extra packets are dropped by hardware.

**Multiport eSwitch Mode**
The multiport eswitch mode allows adding rules on a VF representor with an action forwarding the packet to the physical port of the physical function. This can be used to implement failover or forward packets based on external information such as the cost of the route.

1. To configure multiport eswitch mode, the nvconfig parameter LAG_RESOURCE_ALLOCATION must be set.

2. After the driver loads, configure multiport eSwitch for each PF where `enp8s0f0` and `enp8s0f1` represent the netdevices for the PFs:

```
echo multiport_esw > /sys/class/net/enp8s0f0/compat/devlink/lag_port_select_mode
echo multiport_esw > /sys/class/net/enp8s0f1/compat/devlink/lag_port_select_mode
```

The mode becomes operational after entering switchdev mode on both PFs.

Rule example:

```
tc filter add dev enp8s0f0_0 prot ip root flower dst_ip 7.7.7.7 action mirred egress redirect dev enp8s0f1
```

### OVS-DPDK Hardware Acceleration

### OVS-DPDK Hardware Offloads Configuration

To configure OVS-DPDK HW offloads:

1. Unbind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
```
2. Change the e-switch mode from legacy to switchdev on the PF device (make sure all VFs are unbound). This also creates the VF representor netdevices in the host OS.

```
    echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

To revert to SR-IOV legacy mode:

```
    echo legacy > /sys/class/net/enp4s0f0/compat/devlink/mode
```

**Note**

VMs with attached VFs must be powered off to be able to unbind the VFs.

```
    echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
    echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

**Note**

This command removes the VF representor netdevices.

3. Bind the VFs:

```
    echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
    echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

4. Run the OVS service:

```
    systemctl start openvswitch
```
5. Enable hardware offload (disabled by default):

```
  ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
  ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

6. Configure the DPDK whitelist:

```
  ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-extra="-a 0000:01:00.0,representor=[0],dv_flow_en=1,dv_esw_en=1,dv_xmeta_en=1"
```

   Where representor=[0-N].

7. Restart the OVS service:

```
  systemctl restart openvswitch
```

   **Info**
   
   This step is required for the hardware offload changes to take effect.

8. Create OVS-DPDK bridge:

```
  ovs-vsctl --no-wait add-br br0-ovs -- set bridge br0-ovs datapath_type=netdev
```

9. Add PF to OVS:
Offloading VXLAN Encapsulation/Decapsulation Actions

vSwitch in userspace requires an additional bridge. The purpose of this bridge is to allow use of the kernel network stack for routing and ARP resolution.

The datapath must look up the routing table and ARP table to prepare the tunnel header and transmit data to the output port.

Configuring VXLAN Encap/Decap Offloads

1. Create a br-phy bridge:

   ```
   ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk options:dpdk-devargs=0000:88:00.0
   ```

2. Add representor to OVS:

   ```
   ovs-vsctl add-port br0-ovs representor -- set Interface representor type=dpdk options:dpdk-devargs=0000:88:00.0,representor=[0]
   ```

   Where representor=[0-N].

**Note**

The configuration is done with:

- PF on 0000:03:00.0 PCIe and MAC 98:03:9b:cc:21:e8
- Local IP 56.56.67.1 – br-phy interface is configured to this IP
- Remote IP 56.56.68.1

To configure OVS-DPDK VXLAN:

1. Create a br-phy bridge:
2. Attach PF interface to br-phy bridge:

```bash
ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk options:dpdk-devargs=0000:03:00.0
```

3. Configure IP to the bridge:

```bash
ip addr add 56.56.67.1/24 dev br-phy
```

4. Create a br-ovs bridge:

```bash
ovs-vsctl add-br br-ovs -- set Bridge br-ovs datapath_type=netdev -- br-set-external-id br-ovs bridge-id br-ovs -- set bridge br-ovs fail-mode=standalone
```

5. Attach representor to br-ovs:

```bash
ovs-vsctl add-port br-ovs pf0vf0 -- set Interface pf0vf0 type=dpdk options:dpdk-devargs=0000:03:00.0,representor=[0]
```

6. Add a port for the VXLAN tunnel:

```bash
ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0 type=vxlan options:local_ip=56.56.67.1 options:remote_ip=56.56.68.1 options:key=45 options:dst_port=4789
```
CT Offload

CT enables stateful packet processing by keeping a record of currently open connections. OVS flows using CT can be accelerated using advanced NICs by offloading established connections.

To view offloaded connections, run:

```
  ovs-appctl dpctl/offload-stats-show
```

SR-IOV VF LAG

To configure OVS-DPDK SR-IOV VF LAG:

1. Enable SR-IOV in the NIC firmware:

   ```
   // It is recommended to query the parameters first to determine if change is needed, to save unnecessary reboot
   mst start
   mlxconfig -d <mst device> -y set PF_NUM_OF_VF_VALID=0 SRIOV_EN=1 NUM_OF_VFS=8
   ```

   If configuration changes were made, unless the NIC is BlueField DPU Mode, perform a warm reboot of the Server OS. Otherwise, please perform BlueField System-Level Reset.

2. Allocate the desired number of VFs per port:

   ```
   echo $n > /sys/class/net/<net name>/device/sriov_numvfs
   ```

3. Unbind all VFs:

   ```
   echo <VF PCI> >/sys/bus/pci/drivers/mlx5_core/unbind
   ```
4. Change both devices' mode to switchdev:

   `devlink dev eswitch set pci/<PCI> mode switchdev`

5. Create Linux bonding using kernel modules:

   `modprobe bonding mode=<desired mode>`

   **Info**

   Other bonding parameters can be added here. The supported bond modes are: Active-backup, XOR and LACP.

6. Bring all PFs and VFs down:

   `ip link set <PF/VF> down`

7. Attach both PFs to the bond:

   `ip link set <PF> master bond0`

8. To use VF-LAG with OVS-DPDK, add the bond master (PF) to the bridge:

   `ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk options:dpdk-devargs=0000:03:00.0 options:dpdk-lsc-interrupt=true`

9. Add representor $N of PF0 or PF1 to a bridge:
In user space, there are two main approaches for communicating with a guest (VM), either through SR-IOV or virtio.

PHY ports (SR-IOV) allow working with port representor, which is attached to the OVS and a matching VF is given with pass-through to the guest. HW rules can process packets from

**VirtIO Acceleration Through VF Relay: Software and Hardware vDPA**

1. **Note**

   Hardware vDPA is enabled by default. If your hardware does not support vDPA, the driver will fall back to Software vDPA.

   To check which vDPA mode is activated on your driver, run: `ovs-ofctl -O OpenFlow14 dump-ports br0-ovs` and look for `hw-mode` flag.

2. **Note**

   This feature has not been accepted to the OVS-DPDK upstream yet, making its API subject to change.

```plaintext
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N type=dpdk options:dpdk-devargs=<PF0 PCI>,representor=pf0vf$N

Or:

ovs-vsctl add-port br-phy rep$N -- set Interface rep$N type=dpdk options:dpdk-devargs=<PF0 PCI>,representor=pf1vf$N
```
up-link and direct them to the VF without going through SW (OVS). Therefore, using SR-IOV achieves the best performance.

However, SR-IOV architecture requires the guest to use a driver specific to the underlying HW. Specific HW driver has two main drawbacks:

- Breaks virtualization in some sense (guest is aware of the HW). It can also limit the type of images supported.

- Gives less natural support for live migration.

Using a virtio port solves both problems, however, it reduces performance and causes loss of some functionalities, such as, for some HW offloads, working directly with virtio. The netdev type dpdkvdpdpa solves this conflict as it is similar to the regular DPDK netdev yet introduces several additional functionalities.

dpdkvdpdpa translates between the PHY port to the virtio port. It takes packets from the Rx queue and sends them to the suitable Tx queue, and allows transfer of packets from the virtio guest (VM) to a VF and vice-versa, benefitting from both SR-IOV and virtio.

To add a vDPA port:

```
  ovs-vsctl add-port br0 vdpa0 -- set Interface vdpa0 type=dpdkvdpdpa \
  options:vdpa-socket-path=<sock path> \
  options:vdpa-accelerator-devargs=<vf pci id> \
  options:dpdk-devargs=<pf pci id>,representor=[id] \
  options: vdpa-max-queues =<num queues> \
  options: vdpa-sw=<true/false>
```

**Note**

vdpa-max-queues is an optional field. When the user wants to configure 32 vDPA ports, the maximum queues number is limited to 8.

**vDPA Configuration in OVS-DPDK Mode**
Prior to configuring vDPA in OVS-DPDK mode, perform the following:

1. Generate the VF:

   ```
   echo 0 > /sys/class/net/enp175s0f0/device/sriov_numvfs
   echo 4 > /sys/class/net/enp175s0f0/device/sriov_numvfs
   ```

2. Unbind each VF:

   ```
   echo <pci> > /sys/bus/pci/drivers/mlx5_core/unbind
   ```

3. Switch to switchdev mode:

   ```
   echo switchdev >> /sys/class/net/enp175s0f0/compat/devlink/mode
   ```

4. Bind each VF:

   ```
   echo <pci> > /sys/bus/pci/drivers/mlx5_core/bind
   ```

5. Initialize OVS:

   ```
   ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
   ovs-vsctl --no-wait set Open_vSwitch . other_config:hw-offload=true
   ```

To configure vDPA in OVS-DPDK mode:

1. OVS configuration:

   ```
   ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-extra="-a 0000:01:00.0,representor=
   ```
To configure vDPA in OVS-DPDK mode on BlueField DPUs, set the bridge with the software or hardware vDPA port:

- To create the OVS-DPDK bridge on the Arm side:

  ```
  ovs-vsctl add-br br0-ovs -- set bridge br0-ovs datapath_type=netdev
  ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk options:dpdk-devargs=0000:01:00.0
  ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0 type=dpdkvdpa options:vdpa-socket-path=/var/run/virtio-forwarder/sock0 options:vdpa-accelerator-devargs=0000:01:00.2 options:dpdk-devargs=0000:01:00.0,representor=[0] options: vdpa-max-queues=8
  ```

- To create the OVS-DPDK bridge on the host side:

  ```
  ovs-vsctl add-br br1-ovs -- set bridge br1-ovs datapath_type=netdev protocols=OpenFlow14
  ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0 type=dpdkvdpa options:vdpa-socket-path=/var/run/virtio-forwarder/sock0 options:vdpa-accelerator-devargs=0000:af:00.2
  ```

To create the OVS-DPDK bridge:

```bash
ovs-vsctl add-br br0-ovs -- set bridge br0-ovs datapath_type=netdev
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk options:dpdk-devargs=0000:01:00.0
```

3. Create vDPA port as part of the OVS-DPDK bridge:

```bash
ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0 type=dpdkvdpa options:vdpa-socket-path=/var/run/virtio-forwarder/sock0 options:vdpa-accelerator-devargs=0000:01:00.2 options:dpdk-devargs=0000:01:00.0,representor=[0] options: vdpa-max-queues=8
```
**Software vDPA Configuration in OVS-Kernel Mode**

Software vDPA can also be used in configurations where hardware offload is done through TC and not DPDK.

1. OVS configuration:

   ```bash
   ovs-vsctl set Open_vSwitch . other_config:dpdk-extra="-a 0000:01:00.0,representor=0,dv_flow_en=1,dv_esw_en=0,idv_xmeta_en=0,isolated_mode=1"
   /usr/share/openvswitch/scripts/ovs-ctl restart
   ```

2. Create OVS-DPDK bridge:

   ```bash
   ovs-vsctl add-br br0-ovs -- set bridge br0-ovs datapath_type=netdev
   ```

3. Create vDPA port as part of the OVS-DPDK bridge:

   ```bash
   ovs-vsctl add-port br0-ovs vdpa0 -- set interface vdpa0 type=dpkvdpa options:vdpa-socket-path=/var/run/virtio-forwarder/sock0 options:vdpa-accelerator-devargs=0000:01:00.2 options:dpdk-devargs=0000:01:00.0,representor=[0] options:vdpa-max-queues=8
   ```

4. Create Kernel bridge:

   ```bash
   ovs-vsctl add-br br-kernel
   ```

5. Add representors to Kernel bridge:
Large MTU/Jumbo Frame Configuration

To configure MTU/jumbo frames:

1. Verify that the Kernel version on the VM is 4.14 or above:
   ```
   cat /etc/redhat-release
   ```

2. Set the MTU on both physical interfaces in the host:
   ```
   ifconfig ens4f0 mtu 9216
   ```

3. Send a large size packet and verify that it is sent and received correctly:
   ```
   tcpdump -i ens4f0 -nev icmp &
   ping 11.100.126.1 -s 9188 -M do -c 1
   ```

4. Enable `host_mtu` in XML and add the following values:
   ```
   host_mtu=9216,csum=on,guest_csum=on,host_tso4=on,host_tso6=on
   ```

   Example:
   ```
   <qemu:commandline>
   <qemu:arg value='-chardev'/>
   ```
5. Add the `mtu_request=9216` option to the OVS ports inside the container and restart the OVS:

```
  ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk options:dpdk-devargs=0000:c4:00.0 mtu_request=9216
```

Or:

```
  ovs-vsctl add-port br0-ovs vdpa0 -- set Interface vdpa0 type=dpdkvdpa options:vdpa-socket-path=/tmp/sock0 options:vdpa-accelerator-devargs=0000:c4:00.2 options:dpdk-devargs=0000:c4:00.0,representor=[0] mtu_request=9216
  /usr/share/openvswitch/scripts/ovs-ctl restart
```

6. Start the VM and configure the MTU on the VM:

```
  ifconfig eth0 11.100.124.2/16 up
  ifconfig eth0 mtu 9216
  ping 11.100.126.1 -s 9188 -M do -c1
```

**E2E Cache**

❗️ Note
OVS offload rules are based on a multi-table architecture. E2E cache enables merging the multi-table flow matches and actions into one joint flow.

This improves CT performance by using a single-table when an exact match is detected.

To set the E2E cache size (default is 4k):

```bash
ovs-vsctl set open_vswitch . other_config:e2e-size=<size>
systemctl restart openvswitch
```

To enable E2E cache (disabled by default):

```bash
ovs-vsctl set open_vswitch . other_config:e2e-enable=true
systemctl restart openvswitch
```

To run E2E cache statistics:

```bash
ovs-appctl dpctl/dump-e2e-stats
```

To run E2E cache flows:

```bash
ovs-appctl dpctl/dump-e2e-flows
```

**Geneve Encapsulation/Decapsulation**

Geneve tunneling offload support includes matching on extension header.

To configure OVS-DPDK Geneve encap/decap:
1. Create a br-phy bridge:

```
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach PF interface to br-phy bridge:

```
ovs-vsctl add-port br-phy pf -- set Interface pf type=dpdk options:dpdk-devargs=<PF PCI>
```

3. Configure IP to the bridge:

```
ifconfig br-phy <$local_ip_1> up
```

4. Create a br-int bridge:

```
ovs-vsctl --may-exist add-br br-int -- set Bridge br-int datapath_type=netdev -- br-set-external-id br-int bridge-id br-int -- set bridge br-int fail-mode=standalone
```

5. Attach representor to br-int:

```
ovs-vsctl add-port br-int rep$x -- set Interface rep$x type=dpdk options:dpdk-devargs=<PF PCI>,representor=[$x]
```

6. Add a port for the Geneve tunnel:

```
ovs-vsctl add-port br-int geneve0 -- set interface geneve0 type=geneve options:key=<VNI> options:remote_ip=<$remote_ip_1> options:local_ip=<$local_ip_1>
```

**Parallel Offloads**
OVS-DPDK supports parallel insertion and deletion of offloads (flow and CT). While multiple threads are supported (only one is used by default).

To configure multiple threads:

```sh
ovs-vsctl set Open_vSwitch . other_config:offload-threads=3
systemctl restart openvswitch
```

**Note**

Refer to the [OVS user manual](#) for more information.

---

**sFlow**

sFlow allows monitoring traffic sent between two VMs on the same host using an sFlow collector.

To sample all traffic over the OVS bridge, run the following:

```sh
# ovs-vsctl -- --id=@sflow create sflow agent="$SFLOW_AGENT" 
  target="$SFLOW_TARGET:$SFLOW_HEADER" 
  header=$SFLOW_HEADER 
  sampling=$SFLOW_SAMPLING polling=10 
  -- set bridge sflow=@sflow
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFLOW_AGENT</td>
<td>Indicates that the sFlow agent should send traffic from SFLOW_AGENT's IP address</td>
</tr>
<tr>
<td>SFLOW_TARGET</td>
<td>Remote IP address of the sFlow collector</td>
</tr>
<tr>
<td>SFLOW_PORT</td>
<td>Remote IP destination port of the sFlow collector</td>
</tr>
<tr>
<td>SFLOW_HEADER</td>
<td>Size of packet header to sample (in bytes)</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFLOW_SAMPLING</td>
<td>Sample rate</td>
</tr>
</tbody>
</table>

To clear the sFlow configuration, run:

```bash
# ovs-vsctl clear bridge br-vxlan mirrors
```

#### Note

Currently sFlow for OVS-DPDK is supported without CT.

---

### CT CT NAT

To enable ct-ct-nat offloads in OVS-DPDK (disabled by default), run:

```bash
ovs-vsctl set open_vswitch . other_config:ct-action-on-nat-conns=true
```

If disabled, ct-ct-nat configurations are not fully offloaded, improving connection offloading rate for other cases (ct and ct-nat).

If enabled, ct-ct-nat configurations are fully offloaded but ct and ct-nat offloading would be slower to create.

### OpenFlow Meters (OpenFlow13+)

OpenFlow meters in OVS are implemented according to RFC 2697 (Single Rate Three Color Marker—srTCM).
The srTCM meters an IP packet stream and marks its packets either green, yellow, or red. The color is decided on a Committed Information Rate (CIR) and two associated burst sizes, Committed Burst Size (CBS), and Excess Burst Size (EBS).

A packet is marked green if it does not exceed the CBS, yellow if it exceeds the CBS but not the EBS, and red otherwise.

The volume of green packets should never be smaller than the CIR.

To configure a meter in OVS:

1. Create a meter over a certain bridge, run:

   ```bash
   ovs-ofctl -O openflow13 add-meter $bridge
   meter=$id,$pktps/$kbps,band=type=drop,rate=$rate,[burst,burst_size=$burst_size]
   ```

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bridge</td>
<td>Name of the bridge on which the meter should be applied.</td>
</tr>
<tr>
<td>id</td>
<td>Unique meter ID (32 bits) to be used as an identifier for the meter.</td>
</tr>
<tr>
<td>pktps/kbps</td>
<td>Indication if the meter should work according to packets or kilobits per second.</td>
</tr>
<tr>
<td>rate</td>
<td>Rate of pktps/kbps of allowed data transmission.</td>
</tr>
<tr>
<td>burst</td>
<td>If set, enables burst support for meter bands through the burst_size parameter.</td>
</tr>
<tr>
<td>burst_size</td>
<td>If burst is specified for the meter entry, configures the maximum burst allowed for the band in kilobits/packets, depending on whether kbps or pktps has been specified. If unspecified, the switch is free to select some reasonable value depending on its configuration. Currently, if burst is not specified, the burst_size parameter is set the same as rate.</td>
</tr>
</tbody>
</table>
2. Add the meter to a certain OpenFlow rule. For example:

```bash
ovs-ofctl -O openflow13 add-flow $bridge "table=0,actions=meter:$id,normal"
```

3. View the meter statistics:

```bash
ovs-ofctl -O openflow13 meter-stats $bridge meter=$id
```

4. For more information, refer to official OVS documentation.

**OVS-DOCA Hardware Acceleration**

OVS-DOCA is designed on top of NVIDIA's networking API to preserve the same OpenFlow, CLI, and data interfaces (e.g., vdpa, VF passthrough), as well as datapath offloading APIs, also known as OVS-DPDK and OVS-Kernel. While all OVS flavors make use of flow offloads for hardware acceleration, due to its architecture and use of DOCA libraries, the OVS-DOCA mode provides the most efficient performance and feature set among them, making the most out of NVIDIA NICs and DPUs.

The following subsections provide the necessary steps to launch/deploy OVS DOCA.

**Configuring OVS-DOCA**

To configure OVS DOCA HW offloads:

1. Unbind the VFs:
2. Change the e-switch mode from `legacy` to `switchdev` on the PF device (make sure all VFs are unbound):

```
echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

**Note**

VMs with attached VFs must be powered off to be able to unbind the VFs.

To revert to SR-IOV `legacy` mode:

```
echo legacy > /sys/class/net/enp4s0f0/compat/devlink/mode
```

3. Bind the VFs:

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/bind
```

**Note**

This command also creates the VF representor netdevices in the host OS.
4. Configure huge pages:

```bash
mkdir -p /hugepages
mount -t hugetlbfs hugetlbfs /hugepages
echo 4096 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

5. Run the Open vSwitch service:

```bash
systemctl start openvswitch
```

6. Enable DOCA mode and hardware offload (disabled by default):

```bash
ovs-vsctl --no-wait set Open_vSwitch . other_config:doca-init=true
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

7. Restart the Open vSwitch service.

```bash
systemctl restart openvswitch
```

8. Create OVS-DOCA bridge:

```bash
ovs-vsctl --no-wait add-br br0-ovs -- set bridge br0-ovs datapath_type=netdev
```

Info

This step is required for HW offload changes to take effect.
9. Add PF to OVS:

```bash
ovs-vsctl add-port br0-ovs enp4s0f0 -- set Interface enp4s0f0 type=dpdk
```

10. Add representor to OVS:

```bash
ovs-vsctl add-port br0-ovs enp4s0f0_0 -- set Interface enp4s0f0_0 type=dpdk
```

**Info**

The legacy option to add DPDK ports without using a related netdev by providing dpdk-devargs still exists:

1. Add a PF port:

```bash
ovs-vsctl add-port br0-ovs pf -- set Interface pf type=dpdk
options:dpdk-devargs=0000:88:00.0
```

2. Add a VF representor port:

```bash
ovs-vsctl add-port br0-ovs representor -- set Interface representor type=dpdk
options:dpdk-devargs=0000:88:00.0,representor=[0]
```

3. Add a SF representor port:

```bash
ovs-vsctl add-port br0-ovs representor -- set Interface representor type=dpdk
options:dpdk-devargs=0000:88:00.0,representor=sf[0]
```
4. Add a BlueField host PF representor port:

```
ovs-vsctl add-port br0-ovs hpf -- set Interface hpf type=dpdk
options:dpdk-devargs=0000:88:00.0,representor=[65535]
```

11. Optional configuration:

1. To set port MTU, run:

```
ovs-vsctl set interface enp4s0f0 mtu_request=9000
```

**Note**

OVS restart is required for changes to take effect.

2. To set VF/SF MAC, run:

```
ovs-vsctl add-port br0-ovs enp4s0f0 -- set Interface enp4s0f0 type=dpdk
options:dpdk-vf-mac=00:11:22:33:44:55
```

**Note**

Unbinding and rebinding the VFs/SFs is required for the change to take effect.

**Notable Differences Between OVS-DPDK and OVS-DOCA**
OVS-DOCA shares most of its structure with OVS-DPDK. To benefit from the DOCA offload design, some of the behavior of userland datapath and ports are however modified.

**Eswitch Dependency**

Configured in switchdev mode, the physical port and all supported functions share a single general domain to execute the offloaded flows, the eswitch.

All ports on the same eswitch are dependent on its physical function. If this main physical function is deactivated (e.g., removed from OVS or its link set down), dependent ports are disabled as well.

**Pre-allocated Offload Tables**

To offer the highest insertion speed, DOCA offloads pre-allocate offload structures (entries and containers).

When starting the vSwitch daemon, offloads are thus configured with sensible defaults. If different numbers of offloads are required, configuration entries specific to OVS-DOCA are available and are described in the next section.

**Unsupported CT-CT-NAT**

The special ct-ct-nat mode that can be configured in OVS-kernel and OVS-DPDK is not supported by OVS-DOCA.

**OVS-DOCA Specific vSwitch Configuration**

The following configuration is particularly useful or specific to OVS-DOCA mode.
The following table provides other_config configurations which are global to the vSwitch (non-exhaustive list, check manpage for more):

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>
| other_config:doca-init   | • Optional string, either true or false  
                          • Set this value to true to enable DOCA Flow HW offload  
                          • The default value is false. Changing this value requires restarting the daemon.  
                          • This is only relevant for userspace datapath |
| other_config:hw-offload-ct-size | • Optional string, containing an integer, at least 0  
                              • Only for the DOCA offload provider on netdev datapath  
                              • Configure the usable amount of connection tracking (CT) offload entries  
                              • The default value is 250000. Changing this value requires restarting the daemon.  
                              • Setting a value of 0 disables CT offload  
                              • Changing this configuration affects the OVS memory usage as CT tables are allocated on OVS start  
                              • Maximum number of supported connections is 2M |

⚠️ **Warning**  
Setting this parameter to more than 2M might result in failures.
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>
| offload-ct-ipv6-enabled               | • Set this value to true to enable IPv6 CT offload  
• The default value is false. Changing this value requires restarting the daemon.  
• Changing this configuration affects the OVS memory usage as CT tables are allocated on OVS start |
| other_config:doca-congestion-threshold | • Optional string, containing an integer, in range 30 to 90  
• The occupancy rate of DOCA offload structures that triggers a resize, as a percentage  
• Default to 80, but only relevant if other_config:doca-init is true. Changing this value requires restarting the daemon. |
| other_config:ctl-pipe-size            | • Optional string, containing an integer  
• The initial size of DOCA control pipes  
• Default to 0, which is DOCA's internal default value |
| other_config:ctl-pipe-infra-size      | • Optional string, containing an integer  
• The initial size of infrastructure DOCA control pipes: root, post-hash, post-ct, post-meter, split, miss.  
• Default to 0, which fallbacks to other_config:ctl-pipe-size |
| other_config:pmd-quiet-idle           | • Optional string, either true or false  
• Allow the PMD threads to go into quiescent mode when idling. If no packets are received or waiting to be processed and sent, enter a continuous quiescent period. End this period as soon as a packet is received.  
• This option is disabled by default |
| other_config:pmd-maxsleep             | • Optional string, containing an integer, in range 0 to 10,000  
• Specifies the maximum sleep time in microseconds per iteration for a PMD thread which has received zero or a small amount of packets from the Rx queues it is polling.  
• The actual sleep time requested is based on the load of the Rx queues that the PMD polls and may be less than the maximum value  
• The default value is 0 microseconds, which means that the PMD does not sleep regardless of the load from the Rx queues that it polls |
To avoid requesting very small sleeps (e.g., less than 10 µs) the value is rounded up to the nearest 10 µs
• The maximum value is 10000 microseconds.

Optional string, containing an integer
• Specifies the maximum number of memzones that can be created in DPDK
• The default is empty, keeping DPDK’s default. Changing this value requires restarting the daemon.

With PMD multi-threading support, OVS creates one PMD thread for each NUMA node by default if there is at least one DPDK interface added to OVS from that NUMA node. However, in cases where there are multiple ports/rxqs producing traffic, performance can be improved by creating multiple PMD threads running on separate cores. These PMD threads can share the workload by each being responsible for different ports/rxqs. Assignment of ports/rxqs to PMD threads is done automatically.
A set bit in the mask means a PMD thread is created and pinned to the corresponding CPU core. For example, to run PMD threads on cores 1 and 2, run:

```
$ ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x6
```

### netdev-dpdk

The following table provides netdev-dpdk configurations which only userland (DOCA or DPDK) netdevs support (non-exhaustive list, check manpage for more):

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options:iface-name</td>
<td>• Specifies the interface name of the port</td>
</tr>
</tbody>
</table>
### Offloading VXLAN Encapsulation/Decapsulation Actions

vSwitch in userspace rather than kernel-based Open vSwitch requires an additional bridge. The purpose of this bridge is to allow use of the kernel network stack for routing and ARP resolution.

The datapath must look up the routing table and ARP table to prepare the tunnel header and transmit data to the output port.

VXLAN encapsulation/decapsulation offload configuration is done with:

- PF on 0000:03:00.0 PCIe and MAC 98:03:9b:cc:21:e8
- Local IP 56.56.67.1 – the br-phy interface is configured to this IP
- Remote IP 56.56.68.1

To configure OVS DOCA VXLAN:

1. Create a br-phy bridge:

   ```
   ovs-vsctl add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone other_config:hwaddr=98:03:9b:cc:21:e8
   ```

2. Attach PF interface to br-phy bridge:

   ```
   ovs-vsctl add-port br-phy enp4s0f0 -- set Interface enp4s0f0 type=dpdk
   ```
3. Configure IP to the bridge:

```
ip addr add 56.56.67.1/24 dev br-phy
```

4. Create a br-ovs bridge:

```
ovs-vsctl add-br br-ovs -- set Bridge br-ovs datapath_type=netdev -- br-set-external-id br-ovs
bridge-id br-ovs -- set bridge br-ovs fail-mode=standalone
```

5. Attach representor to br-ovs:

```
ovs-vsctl add-port br-ovs enp4s0f0_0 -- set Interface enp4s0f0_0 type=dpdk
```

6. Add a port for the VXLAN tunnel:

```
ovs-vsctl add-port ovs-sriov vxlan0 -- set Interface vxlan0 type=vxlan options:local_ip=56.56.67.1
options:remote_ip=56.56.68.1 options:key=45 options:dst_port=4789

ovs-vsctl add-port br-int vxlan0 -- set interface vxlan0 type=vxlan options:key=30
options:remote_ip=10.0.30.1 options:exts=gbp
```

**VXLAN GBP Extension**

The VXLAN group-based policy (GBP) model outlines an application-focused policy framework that specifies connectivity requirements for applications, independent of the network's physical layout.

Setting GBP extension for a VXLAN port allows for matching on and setting a GBP ID per flow. To enable GBP extension when the port vxlan0 is first added:

```
ovs-vsctl add-port br-int vxlan0 -- set interface vxlan0 type=vxlan options:key=30
options:remote_ip=10.0.30.1 options:exts=gbp
```

It is also possible to enable GBP extension for an existing VXLAN port:
This approach has a limitation that it does not take effect until after the OVS `vswitchd` service is restarted. In cases where there are multiple VXLAN ports, they must all share the same GBP extension configuration in their port options. A mixed configuration with some VXLAN ports having the GBP extension enabled and others disabled is not supported.

When GBP extension is enabled, the following OpenFlow rules which match on a GBP ID 32 or set a GBP ID 64 in the actions, can be offloaded:

```
ovs-vsctl set interface vxlan1 options:exts=gbp
```

```
ovs-ofctl add-flow br-int table=0,priority=100,in_port=vxlan0,tun_gbp_id=32 actions=output:pf0vf0
ovs-ofctl add-flow br-int table=0,priority=100,in_port=pf0vf0 actions=load:64->NXM_NX_TUN_GBP_ID[],output:vxlan0
```

**Offloading Connection Tracking**

Connection tracking enables stateful packet processing by keeping a record of currently open connections.

OVS flows utilizing connection tracking can be accelerated using advanced NICs by offloading established connections.

To view offload statistics, run:

```
ovs-appctl dpctl/offload-stats-show
```

**SR-IOV VF LAG**

To configure OVS-DOCA SR-IOV VF LAG:

1. Enable SR-IOV on the NICs:
2. Allocate the desired number of VFs per port:

```
echo $n > /sys/class/net/<net name>/device/sriov_numvfs
```

3. Unbind all VFs:

```
echo <VF PCI> >/sys/bus/pci/drivers/mlx5_core/unbind
```

4. Change both NICs' mode to SwitchDev:

```
devlink dev eswitch set pci/<PCI> mode switchdev
```

5. Create Linux bonding using kernel modules:

```
modprobe bonding mode=<desired mode>
```
6. Bring all PFs and VFs down:

   ```
   ip link set <PF/VF> down
   ```

7. Attach both PFs to the bond:

   ```
   ip link set <PF> master bond0
   ```

8. Bring PFs and bond link up:

   ```
   ip link set <PF0> up
   ip link set <PF1> up
   ip link set bond0 up
   ```

9. Add the bond interface to the bridge as type=dpdk:

   ```
   ovs-vsctl add-port br-phy bond0 -- set Interface bond0 type=dpdk options:dpdk-lsc-interrupt=true
   ```

---

**Info**

The legacy option to work with VF-LAG in OVS-DPDK is to add the bond master (PF) interface to the bridge:
10. Add representor of PF0 or PF1 to a bridge:

```bash
ovs-vsctl add-port br-phy p0 -- set Interface p0 type=dpdk options:dpdk-devargs=<PF0-PCI>,dv_flow_en=2,dv_xmeta_en=4 options:dpdk-lsc-interrupt=true
```

Or:

```bash
ovs-vsctl add-port br-phy enp4s0f0_0 -- set Interface enp4s0f0_0 type=dpdk
```

Or:

```bash
ovs-vsctl add-port br-phy enp4s0f1_0 -- set Interface enp4s0f1_0 type=dpdk
```

ℹ️ **Info**

The legacy option to add DPDK ports:

```bash
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N type=dpdk options:dpdk-devargs=<PF0-PCI>,representor=pf0vf$N,dv_flow_en=2,dv_xmeta_en=4
```

Or:

```bash
ovs-vsctl add-port br-phy rep$N -- set Interface rep$N type=dpdk options:dpdk-devargs=<PF0-PCI>,representor=pf1vf$N,dv_flow_en=2,dv_xmeta_en=4
```

**Multiport eSwitch Mode**
Multiport eswitch mode allows adding rules on a VF representor with an action, forwarding the packet to the physical port of the physical function. This can be used to implement failover or to forward packets based on external information such as the cost of the route.

1. To configure multiport eswitch mode, the nvconig parameter `LAG_RESOURCE_ALLOCATION=1` must be set in the BlueField Arm OS, according to the following instructions:

   ```
   mst start
   mlxconfig -d /dev/mst/mt*conf0 -y s LAG_RESOURCE_ALLOCATION=1
   ```

2. Perform a BlueField system reboot for the mlxconfig settings to take effect.

3. After the driver loads, and before moving to switchdev mode, configure multiport eswitch for each PF where p0 and p1 represent the netdevices for the PFs:

   ```
   devlink dev param set pci/0000:03:00.0 name esw_multiport value 1 cmode runtime
   devlink dev param set pci/0000:03:00.1 name esw_multiport value 1 cmode runtime
   ```

   **Info**

   The mode becomes operational after entering switchdev mode on both PFs.

4. This mode can be activated by default in BlueField by adding the following line into `/etc/mellanox/mlnx-bf.conf`:

   ```
   ENABLE_ESWITCH_MULTIPORT="yes"
   ```
While in this mode, the second port is not an eswitch manager, and should be add to OVS using this command:

```bash
ovs-vsctl add-port br-phy enp4s0f1 -- set interface enp4s0f1 type=dpdk
```

VFs for the second port can be added using this command:

```bash
ovs-vsctl add-port br-phy enp4s0f1_0 -- set interface enp4s0f1_0 type=dpdk
```

### Info

The legacy option to add DPDK ports:

```bash
ovs-vsctl add-port br-phy p1 -- set interface p1 type=dpdk options:dpdk-devargs="0000:08:00.0,dev_xmeta_en=4,dev_flow_en=2,representor=pf1"
```

VFs for the second port can be added using this command:

```bash
ovs-vsctl add-port br-phy p1vf0 -- set interface p1 type=dpdk options:dpdk-devargs="0000:08:00.0,dev_xmeta_en=4,dev_flow_en=2,representor=pf1vf0"
```

### Offloading Geneve Encapsulation/Decapsulation

Geneve tunneling offload support includes matching on extension header.

### Note

OVS-DOCA Geneve option limitations:
To configure OVS-DOCA Geneve encapsulation/decapsulation:

1. Create a `br-phy` bridge:

```bash
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to `br-phy` bridge:

```bash
ovs-vsctl add-port br-phy enp4s0f0 -- set Interface enp4s0f0 type=dpdk
```

3. Configure an IP to the bridge:

```bash
ifconfig br-phy <local_ip_1> up
```

4. Create a `br-int` bridge:

- Only 1 Geneve option is supported
- Max option len is 7
- To change the Geneve option currently being matched and encapsulated, users must remove all ports or restart OVS and configure the new option
- Matching on Geneve options can work with `FLEX_PARSER` profile 0 (the default profile). Working with `FLEX_PARSER` profile 8 is also supported as well. To configure it, run:

```bash
mst start
mlxconfig -d <mst device> s FLEX_PARSER_PROFILE_ENABLE=8
```

00000191-5727-d31e-adf1-57a77ca70003
5. Attach a representor to br-int:

```bash
ovs-vsctl add-port br-int rep$x -- set Interface rep$x type=dpdk options:dpdk-devargs=<PF PCI>\,representor=[$x]\,dv_flow_en=2\,dv_xmeta_en=4
```

6. Add a port for the Geneve tunnel:

```bash
ovs-vsctl add-port br-int geneve0 -- set interface geneve0 type=geneve options:key=<VNI> \ options:remote_ip=<$remote_ip_1> \ options:local_ip=<$local_ip_1>
```

**GRE Tunnel Offloads**

To configure OVS-DOCA GRE encapsulation/decapsulation:

1. Create a br-phy bridge:

```bash
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to br-phy bridge:

```bash
ovs-vsctl add-port br-phy enp4s0f0 -- set Interface enp4s0f0 type=dpdk
```

3. Configure an IP to the bridge:

```bash
ifconfig br-phy <$local_ip_1> up
```
4. Create a `br-int` bridge:

```bash
ovs-vsctl --may-exist add-br br-int -- set Bridge br-int datapath_type=netdev -- br-set-external-id br-int bridge-id br-int -- set bridge br-int fail-mode=standalone
```

5. Attach a representor to `br-int`:

```bash
ovs-vsctl add-port br-int enp4s0f0_0 -- set Interface enp4s0f0_0 type=dpdk
```

Add a port for the Geneve tunnel:

```bash
ovs-vsctl add-port br-int gre0 -- set interface gre0 type=gre options:key=<VNI> options:remote_ip=<$remote_ip_1> options:local_ip=<$local_ip_1>
```

### Slow Path Rate Limiting/SW-Meter

Slow path rate limiting allows controlling the rate of traffic that bypasses hardware offload rules and is subsequently processed by software.

To configure slow path rate limiting:

1. Create a `br-phy` bridge:

```bash
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to `br-phy` bridge:

```bash
ovs-vsctl add-port br-phy pf0 -- set Interface pf0 type=dpdk
```

3. Rate limit `pf0vf0` to 10Kpps with 6K burst size:
4. Restart OVS:

```bash
systemctl restart openvswitch-switch.service
```

A dry-run option is also supported to allow testing different software meter configurations in a production environment. This allows gathering statistics without impacting the actual traffic flow. These statistics can then be analyzed to determine appropriate rate limiting thresholds. When the dry-run option is enabled, traffic is not dropped or rate-limited, allowing normal operations to continue without disruption. However, the system simulates the rate limiting process and increment counters as though packets are being dropped.

To enable slow path rate limiting dry-run:

1. Create a `br-phy` bridge:

```bash
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to `br-phy` bridge:

```bash
ovs-vsctl add-port br-phy pf0 -- set Interface pf0 type=dpdk
```

3. Rate limit `pf0vf0` to 10Kpps with 6K burst size:

```bash
ovs-vsctl set interface pf0 options:sw-meter=pps:10k:6k
```

4. Set the `sw-meter-dry-run` option:
5. Restart OVS:

```
systemctl restart openvswitch-switch.service
```

### Hairpin

Hairpin allows forwarding packets from wire to wire.

To configure hairpin:

1. Create a `br-phy` bridge:

```
ovs-vsctl --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
```

2. Attach a PF interface to `br-phy` bridge:

```
ovs-vsctl add-port br-phy pf0 -- set Interface pf0 type=dpdk
```

3. Add hairpin OpenFlow rule:

```
ovs-ofctl add-flow br-phy"in_port=pf0,ip,actions=in_port"
```

### OpenFlow Meters

OVS-DOCA supports OpenFlow meter action as covered in this document in section "OpenFlow Meters". In addition, OVS-DOCA supports chaining multiple meter actions together in a single datapath rule.
The following is an example configuration of such OpenFlow rules:

```bash
ovs-ofctl add-flow br-phy -O OpenFlow13 "table=0,priority=1,in_port=pf0vf0_r,ip actions=meter=1,resubmit(1)"
ovs-ofctl add-flow br-phy -O OpenFlow13 "table=1,priority=1,in_port=pf0vf0_r,ip actions=meter=2,normal"
```

Meter actions are applied sequentially, first using meter ID 1 and then using meter ID 2.

Use case examples for such a configuration:

- Rate limiting the same logical flow with different meter types—bytes per second and packets per second

- Metering a group of flows. As meter IDs can be used by multiple flows, it is possible to re-use meter ID 2 from this example with other logical flows; thus, making sure that their cumulative bandwidth is limited by the meter.

### DP-HASH Offloads

OVS supports group configuration. The "select" type executes one bucket in the group, balancing across the buckets according to their weights. To select a bucket, for each live bucket, OVS hashes flow data with the bucket ID and multiplies that by the bucket weight to obtain a "score". The bucket with the highest score is selected.

For example:

- `ovs-ofctl add-group br-int 'group_id=1,type=select,bucket=<port1>'`

- `ovs-ofctl add-flow br-int in_port=<port0>,actions=group=1`

Limitations:

1. Info

For more details, refer to the `ovs-ofctl man`. 
• Offloads are supported on IP traffic only (IPv4 or IPv6)

**sFlow**

The sFlow standard outlines a method for capturing traffic data in switched or routed networks. It employs sampling technology to gather statistics from the device, making it suitable for high-speed networks.

With a predetermined sampling rate, one out of every N packets is captured. While this sampling method does not yield completely accurate results, it does offer acceptable accuracy.

To activate sampling for 0.2% of all traffic traversing an OVS bridge named `br-int`, run:

```
  ovs-vsctl -- --id=@sflow create sflow agent=lo target=127.0.0.1:6343 header=96 sampling=512 -- set bridge br-int sflow=@sflow
```

With this sFlow configuration on the bridge, captured packets are mirrored to an sFlow collector application that listens on the default sFlow port, 6343, on localhost.

ℹ️ **Info**

sFlow collector applications fall outside the scope of this guide.

It is possible to set the sampling rate to 1 while configuring sFlow on a bridge, which effectively mirrors all traffic to the sFlow collector.

**OVS-DOCA Known Limitations**

• Only one insertion thread is supported (`n-offload-threads=1`)

• Only 250K connection are offloadable by default (can be configured)
Only 8 CT zones are supported by CT offload

When using two PFs with 127 VFs each and adding their representors to OVS bridge, the user must configure `dpdk-memzones`:

```
  ovs-vscti set o . other_config:dpdk-max-memzones=6500
  restart ovs
```

In an OVS topology that includes both physical and internal bridges, sFlow offloads are only supported on the internal bridge when employing a VXLAN tunnel. Utilizing sFlow on the physical bridge leads to only partial offload of flows in this scenario.

**OVS-DOCA Debugging**

Additional debugging information can be enabled in the vSwitch log file using the `dbg` log level:

```
(  
  topics='netdev|ofproto|ofp|odp|doca'
  IFS='$\n'; for topic in $(ovs-appctl vlog/list | grep -E "$topics" | cut -d' ' -f1)
  do
    printf "$topic:file:dbg "
    done
) | xargs ovs-appctl vlog/set
```

The listed topics are relevant to DOCA offload operations.

Coverage counters specific to the DOCA offload provider have been added. The following command should be used to check them:

```
  ovs-vsctl set o . other_config:dpdk-max-memzones=6500
  restart ovs
```
The following table provides the meaning behind these DOCA-specific counters:

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doca_async_queue_full</td>
<td>The asynchronous offload insertion queue was full while the daemon attempted to insert a new offload. The queue will have been flushed and insertion attempted again. This is not a fatal error but is the sign of a slowed down hardware.</td>
</tr>
<tr>
<td>doca_async_queue_blocked</td>
<td>The asynchronous offload insertion queue has remained full even after several attempts to flush its currently enqueued requests. While not a fatal error, it should never happen during normal offload operations and should be considered a bug.</td>
</tr>
<tr>
<td>doca_async_add_failed</td>
<td>An asynchronous insertion failed specifically due to its asynchronous nature. This is not expected to happen and should be considered a bug.</td>
</tr>
<tr>
<td>doca_pipe_resize</td>
<td>The number of time a DOCA pipe has been resized. This is normal and expected as DOCA pipes receives more entries.</td>
</tr>
<tr>
<td>doca_pipe_resize_over_10_ms</td>
<td>A DOCA pipe resize took longer than 10ms to complete. It can happen infrequently. If a sudden drop in insertion rate is measured, this counter could help identify the root cause.</td>
</tr>
</tbody>
</table>

**OVS-DOCA Build**

To build OVS-DOCA from provided sources and pre-installed DOCA with the same version packages, run:

```
$ ./boot.sh
$ ./configure --prefix=/usr --localstatedir=/var --sysconfdir=/etc --with-dpdk=static --with-doca=static
$ make -j 10
$ make install
```

A helper build script is bundled with OVS-DOCA sources that can be used as follows:
Scaling Megaflows

Megaflows aggregate multiple microflows into a single flow entry, reduce the load on the flow table, and improve packet processing efficiency. Scaling megaflows in OVS is crucial for optimizing network performance and ensuring efficient handling of high traffic volumes. By default, OVS-DOCA can handle up to 200k megaflows.

To effectively manage and scale megaflows, several key configurations in the other_config section of OVS can be adjusted:

- The `flow-limit` parameter sets the maximum number of flows that can be stored in the flow table, helping to control memory usage and prevent overflow.

- The `max-revalidator` parameter defines the longest duration (in milliseconds) that revalidator threads will wait before initiating flow revalidation. It is crucial to understand that this represents the upper limit, and the actual timeout employed by OVS is the lesser of the `max-idle` and `max-revalidator` values. Modifying this parameter is generally not recommended without a thorough understanding of its effects. For systems with less powerful CPUs, setting a higher `max-revalidator` value is suggested to compensate for reduced computational capacity and ensure revalidation completes.

Fine-tuning these settings can improve the scalability and performance of an OVS deployment, allowing it to manage a greater number of megaflows efficiently.

- To set `flow-limit` (default is 200k):

  ```bash
  $ ovs-vsctl set o . other_config:flow-limit=<desired_value>
  ```

- To set `max-revalidator` (default is 250ms).

  ```bash
  $ ovs-vsctl set o . other_config:max-revalidator=<desired_value>
  ```

```
OVS Metrics

OVS exposes Prometheus metrics through its control socket (experimental feature). These metrics can be accessed using the command:

```
$ ovs-appctl metrics/show
```

A terminal dashboard is also installed with OVS, `ovs-metrics`. This script is dependent on the OVS Python API (package `python3-openvswitch`). Its default mode currently watches over a set of offload-related metrics.

OVS Inside BlueField

Verifying Host Connection on Linux

When the DPU is connected to another DPU on another machine, manually assign IP addresses with the same subnet to both ends of the connection.

1. Assuming the link is connected to `p3p1` on the other host, run:

```
$ ifconfig p3p1 192.168.200.1/24 up
```

2. On the host which the DPU is connected to, run:

```
$ ifconfig p4p2 192.168.200.2/24 up
```

3. Have one ping the other. This is an example of the DPU pinging the host:

```
$ ping 192.168.200.1
```
Verifying Connection from Host to BlueField

There are two SFs configured on the BlueField device, `enp3s0f0s0` and `enp3s0f1s0`, and their representors are part of the built-in bridge. These interfaces will get IP addresses from the DHCP server if it is present. Otherwise it is possible to configure IP address from the host. It is possible to access BlueField via the SF netdev interfaces.

For example:

1. Verify the default OVS configuration. Run:

   ```
   # ovs-vsctl show
   5668f9a6-6b93-49cf-a72a-14fd64b4c82b
   Bridge ovsbr1
   Port pf0hpf
     Interface pf0hpf
   Port ovsbr1
     Interface ovsbr1
       type: internal
   Port p0
     Interface p0
   Port en3f0pf0sf0
     Interface en3f0pf0sf0
   Bridge ovsbr2
   Port en3f1pf1sf0
     Interface en3f1pf1sf0
   Port ovsbr2
     Interface ovsbr2
       type: internal
   Port pf1hpf
     Interface pf1hpf
   Port p1
     Interface p1
   ovs_version: "2.14.1"
   ```

2. Verify whether the SF netdev received an IP address from the DHCP server. If not, assign a static IP. Run:

   ```
   # ifconfig enp3s0f0s0
   ```
3. Verify the connection of the configured IP address. Run:

```
# ping 192.168.200.25 -c 5
64 bytes from 192.168.200.25: icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 192.168.200.25: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 192.168.200.25: icmp_seq=3 ttl=64 time=0.232 ms
64 bytes from 192.168.200.25: icmp_seq=4 ttl=64 time=0.174 ms
64 bytes from 192.168.200.25: icmp_seq=5 ttl=64 time=0.168 ms

--- 192.168.200.25 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 91ms
rtt min/avg/max/mdev = 0.168/0.195/0.232/0.031 ms
```

**Verifying Host Connection on Windows**

Set IP address on the Windows side for the RShim or Physical network adapter, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> New-NetIPAddress -InterfaceAlias "Ethernet 16" -IPAddress "192.168.100.1" -PrefixLength 22
```

To get the interface name, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> Get-NetAdapter
```
Output should give us the interface name that matches the description (e.g. NVIDIA BlueField Management Network Adapter).

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Status</th>
<th>MAC Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet 2</td>
<td>NVIDIA ConnectX-4 Lx Ethernet Adapter</td>
<td>6 Not Present</td>
<td>24-8A-07-0D-E8-1D</td>
</tr>
<tr>
<td>Ethernet 6</td>
<td>NVIDIA ConnectX-4 Lx Ethernet Adapter</td>
<td>23 Not Present</td>
<td>24-8A-07-0D-E8-1C</td>
</tr>
<tr>
<td>Ethernet 16</td>
<td>NVIDIA BlueField Management Netw...</td>
<td>15 Up</td>
<td>CA-FE-01-CA-</td>
</tr>
</tbody>
</table>

Once IP address is set, Have one ping the other.

C:\Windows\system32>ping 192.168.100.2

Pinging 192.168.100.2 with 32 bytes of data:
Reply from 192.168.100.2: bytes=32 time=148ms TTL=64
Reply from 192.168.100.2: bytes=32 time=152ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
NVIDIA DOCA Telemetry Service Guide

This guide provides instructions on how to use the DOCA Telemetry Service (DTS) container on top of NVIDIA® BlueField® DPU.

Introduction

DOCA Telemetry Service (DTS) collects data from built-in providers and from external telemetry applications. The following providers are available:

- Data providers:
  - sysfs
  - ethtool
  - tc (traffic control)

- Aggregation providers:
  - fluent_aggr
  - prometheus_aggr

**Note**

Sysfs provider is enabled by default.

DTS stores collected data into binary files under the /opt/mellanox/doca/services/telemetry/data directory. Data write is disabled by default due to BlueField storage restrictions.
DTS can export the data via Prometheus Endpoint (pull) or Fluent Bit (push).

DTS allows exporting NetFlow packets when data is collected from the DOCA Telemetry Exporter NetFlow API client application. NetFlow exporter is enabled from `dts_config.ini` by setting NetFlow collector IP/address and port.

**Service Deployment**

**Available Images**

**Built-in DOCA Service Image**

DOCA Telemetry Service is enabled by default on the DPU and is shipped as part of the BlueField image. That is, every BlueField image contains a fixed service version so as to provide out-of-the-box support for programs based on the DOCA Telemetry Exporter library.

**DOCA Service on NGC**

In addition to the built-in image shipped with the BlueField boot image, DTS is also available on NGC, NVIDIA's container catalog. This is useful in case a new version of the service has been released and the user wants to upgrade from the built-in image. For service-specific configuration steps and deployment instructions, refer to the service's container page.

![Diagram showing data flow and services](image)
DPU Deployment

As mentioned above, DTS starts automatically on BlueField boot. This is done according to the .yaml file located at /etc/kubelet.d/doca_telemetry_standalone.yaml. Removing the .yaml file from this path stops the automatic DTS boot.

DTS files can be found under the directory /opt/mellanox/doca/services/telemetry/.

- Container folder mounts:
  - config
  - data
  - ipcsockets

- Backup files:
  - doca_telemetry_service_{version}_arm64.tar.gz – DTS image
  - doca_telemetry_standalone.yaml – copy of the default boot .yaml file

Host Deployment

DTS supports x86_64 hosts. The providers and exporters all run from a single docker container.

1. Initialize and configure host DTS with the desired DTS version:
2. Run with:

```bash
docker run -d --net=host --uts=host --ipc=host
--privileged
--ulimit stack=67108864 --ulimit memlock=-1
--device=/dev/mst/
--device=/dev/infiniband/
--gpus all
-v "/opt/mellanox/doca/services/telemetry/config:/config"
-v "/opt/mellanox/doca/services/telemetry/ipc_sockets:/tmp/ipc_sockets"
-v "/opt/mellanox/doca/services/telemetry/data:/data"
-v "/usr/lib/mft:/usr/lib/mft"
-v "/sys/kernel/debug:/sys/kernel/debug"
--rm --name doca-telemetry -it $DTS_IMAGE /usr/bin/telemetry-run.sh
```

**Note**

The following mounts are required by specific services only:
Deployment with Grafana Monitoring

Refer to section "Deploying with Grafana Monitoring".

Configuration

The configuration of DTS is placed under `/opt/mellanox/doca/services/telemetry/config` by DTS during initialization. The user can interact with the `dts_config.ini` file and `fluent_bit_configs` folder. `dts_config.ini` contains the main configuration for the service and must be used to enable/disable providers, exporters, data writing. More details are provided in the corresponding sections. For every update in this file, DST must be restarted. Interaction with `fluent_bit_configs` folder is described in section Fluent Bit.

Init Scripts

The InitContainers section of the `.yaml` file has 2 scripts for config initialization:

- `/usr/bin/telemetry-init.sh` – generates the default configuration files if, and only if, the `/opt/mellanox/doca/services/telemetry/config` folder is empty.
Enabling Fluent Bit Forwarding

To enable Fluent Bit forward, add the destination host and port to the command line found in the initContainers section of the .yaml file:

```
    command: ["/bin/bash", "-c", "/usr/bin/telemetry-init.sh && /usr/bin/enable-fluent-forward.sh -i=127.0.0.1 -p=24224"]
```

⚠️ Note

The host and port shown above are just an example. See section Fluent Bit to learn about manual configuration.

Generating Configuration

The configuration folder /opt/mellanox/doca/services/telemetry/config starts empty by default. Once the service starts, the initial scripts run as a part of the initial container and create configuration as described in section Enabling Fluent Bit Forwarding.

Resetting Configuration
Resetting the configuration can be done by deleting the content found in the configuration folder and restarting the service to generate the default configuration.

**Enabling Providers**

Providers are enabled from the `dts_config.ini` configuration file. Uncomment the `enable-provider=$provider-name` line to allow data collection for this provider. For example, uncommenting the following line enables the `ethtool` provider:

```
#enable-provider=ethtool
```

**Note**

More information about telemetry providers can be found under the Providers section.

**Remote Collection**

Certain providers or components are unable to execute properly within the container due to various container limitations. Therefore, they would have to perform remote collection or execution.

The following steps enable remote collection:

1. Activate DOCA privileged executor (DPE), as DPE is how remote collection is achieved:

   ```
   systemctl start dpe
   ```

2. Add `grpc` before `provider-name` (i.e., `enable-provider=grpc.$provider-name`). For example, the following line configures remote collection of the `hcaperf` provider:
3. If there are any configuration lines that are provider-specific, then add the `grpc` prefix as well. Building upon the previous example:

```
enable-provider=grpc.hcaperf
```

```
grpc.hcaperf.mlx5_0=sample
grpc.hcaperf.mlx5_1=sample
```

---

### Enabling Data Write

Uncomment the following line in `dts_config.ini`:

```
#output=/data
```

---

💡 **Note**

Changes in `dts_config.ini` force the main DTS process to restart in 60 seconds to apply the new settings.

---

### Enabling IPC with Non-container Program

For information on enabling IPC between DTS and an application that runs outside of a container, refer to section "Using IPC with Non-container Application" in the DOCA Telemetry Exporter.
Description

Providers

DTS supports on-board data collection from sysf, ethtool, and tc providers. Fluent and Prometheus aggregator providers can collect the data from other applications.

Other providers are available based on different conditions (e.g., specific container mounts or host only such as amber, ppcc_eth, etc). Such providers are described with their dependencies in their corresponding sections.

Sysfs Counters List

The sysfs provider has several components: ib_port, hw_port, mr_cache, eth, hwmon and bf_ptm. By default, all the components (except bf_ptm) are enabled when the provider is enabled:

```
#disable-provider=sysfs
```

The components can be disabled separately. For instance, to disable eth:

```
enable-provider=sysfs
disable-provider=sysfs.eth
```

Note

ib_port and ib_hwv are state counters which are collected per port. These counters are only collected for ports whose state is active.

- ib_port counters:

```
{hca_name}:{port_num}:ib_port_state
{hca_name}:{port_num}:VL15_dropped
```
• **ib_hw counters:**

  
  ```
  {hca_name}:{port_num}:hw_state
  {hca_name}:{port_num}:hw_duplicate_request
  {hca_name}:{port_num}:hw_implied_nak_seq_err
  {hca_name}:{port_num}:hw_lifespan
  {hca_name}:{port_num}:hw_local_ack_timeout_err
  {hca_name}:{port_num}:hw_out_of_buffer
  {hca_name}:{port_num}:hw_out_of_sequence
  {hca_name}:{port_num}:hw_packet_seq_err
  {hca_name}:{port_num}:hw_req_cqe_error
  {hca_name}:{port_num}:hw_req_cqe_flush_error
  {hca_name}:{port_num}:hw_req_remote_access_errors
  {hca_name}:{port_num}:hw_req_remote_invalid_request
  {hca_name}:{port_num}:hw_resp_cqe_error
  {hca_name}:{port_num}:hw_resp_cqe_flush_error
  {hca_name}:{port_num}:hw_resp_local_length_error
  {hca_name}:{port_num}:hw_resp_remote_access_errors
  {hca_name}:{port_num}:hw_rnr_nak_retry_err
  {hca_name}:{port_num}:hw_rx_atomic_requests
  {hca_name}:{port_num}:hw_rx_dct_connect
  ```

```
• ib_mr_cache counters:

{hca_name}:{port_num}:hw_rx_icrc_encapsulated
{hca_name}:{port_num}:hw_rx_read_requests
{hca_name}:{port_num}:hw_rx_write_requests

• Note

Where \( n \) ranges from 0 to 24.

• eth counters:

{hca_name}:{device_name}:eth_collisions
{hca_name}:{device_name}:eth_multicast
{hca_name}:{device_name}:eth_rx_bytes
{hca_name}:{device_name}:eth_rx_compressed
{hca_name}:{device_name}:eth_rx_crc_errors
{hca_name}:{device_name}:eth_rx_dropped
{hca_name}:{device_name}:eth_rx_errors
{hca_name}:{device_name}:eth_rx_fifo_errors
{hca_name}:{device_name}:eth_rx_frame_errors
{hca_name}:{device_name}:eth_rx_length_errors
{hca_name}:{device_name}:eth_rx_missed_errors
{hca_name}:{device_name}:eth_rx_nohandler
{hca_name}:{device_name}:eth_rx_over_errors
{hca_name}:{device_name}:eth_rx_packets
{hca_name}:{device_name}:eth_tx_aborted_errors
{hca_name}:{device_name}:eth_tx_bytes
{hca_name}:{device_name}:eth_tx_carrier_errors
{hca_name}:{device_name}:eth_tx_compressed
BlueField-2 hwmon counters:

(hca_name):{device_name}:eth_tx_dropped
(hca_name):{device_name}:eth_tx_errors
(hca_name):{device_name}:eth_tx_fifo_errors
(hca_name):{device_name}:eth_tx_heartbeat_errors
(hca_name):{device_name}:eth_tx_packets
(hca_name):{device_name}:eth_tx_window_errors

(hwmon_name):{l3cache}:CYCLES
(hwmon_name):{l3cache}:HITS_BANK0
(hwmon_name):{l3cache}:HITS_BANK1
(hwmon_name):{l3cache}:MISSES_BANK0
(hwmon_name):{l3cache}:MISSES_BANK1
(hwmon_name):{pcie}:IN_C_BYTE_CNT
(hwmon_name):{pcie}:IN_C_PKT_CNT
(hwmon_name):{pcie}:IN_NP_BYTE_CNT
(hwmon_name):{pcie}:IN_NP_PKT_CNT
(hwmon_name):{pcie}:IN_P_BYTE_CNT
(hwmon_name):{pcie}:IN_P_PKT_CNT
(hwmon_name):{pcie}:OUT_C_BYTE_CNT
(hwmon_name):{pcie}:OUT_C_PKT_CNT
(hwmon_name):{pcie}:OUT_NP_BYTE_CNT
(hwmon_name):{pcie}:OUT_NP_PKT_CNT
(hwmon_name):{pcie}:OUT_P_PKT_CNT
(hwmon_name):{tile}:MEMORY_READS
(hwmon_name):{tile}:MEMORY_WRITES
(hwmon_name):{tile}:MSS_NO_CREDIT
(hwmon_name):{tile}:VICTIM_WRITE
(hwmon_name):{tilenet}:CDN_DIAG_C_OUT_OF_CRED
(hwmon_name):{tilenet}:CDN_REQ
(hwmon_name):{tilenet}:DDN_REQ
(hwmon_name):{tilenet}:NDN_REQ
(hwmon_name):{trio}:TDMA_DATA_BEAT
(hwmon_name):{trio}:TDMA_PBUF_MAC_AF
(hwmon_name):{trio}:TDMA_RT_AF
(hwmon_name):{trio}:TPIO_DATA_BEAT
(hwmon_name):{triogen}:TX_DAT_AF
(hwmon_name):{triogen}:TX_DAT_AF
• BlueField-3 hwmon counters:

```plaintext
{hwmon_name}:{llt}:GDC_BANK0_RD_REQ
{hwmon_name}:{llt}:GDC_BANK1_RD_REQ
{hwmon_name}:{llt}:GDC_BANK0_WR_REQ
{hwmon_name}:{llt}:GDC_BANK1_WR_REQ
{hwmon_name}:{llt_miss}:GDC_MISS_MACHINE_RD_REQ
{hwmon_name}:{llt_miss}:GDC_MISS_MACHINE_WR_REQ
{hwmon_name}:{mss}:SKYLIB_DDN_TX_FLITS
{hwmon_name}:{mss}:SKYLIB_DDN_RX_FLITS
```

• BlueField-3 bf_ptm counters:

```plaintext
bf:ptm:active_power_profile
bf:ptm:atx_power_available
bf:ptm:core_temp
bf:ptm:ddr_temp
bf:ptm:error_state
bf:ptm:power_envelope
bf:ptm:power_throttling_event_count
bf:ptm:power_throttling_state
bf:ptm:thermal_throttling_event_count
bf:ptm:thermal_throttling_state
bf:ptm:throttling_state
bf:ptm:total_power
bf:ptm:vr0_power
bf:ptm:vr1_power
```

**Port Counters**

The following parameters are located in `/sys/class/infiniband/mlx5_0/ports/1/counters`.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>InfiniBand Spec Name</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>port_rcv_data</td>
<td>The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLs from the port.</td>
<td>PortRcvData</td>
<td>Informati</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>InfiniBand Spec Name</td>
<td>Group</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>port_rcv_packets</td>
<td>Total number of packets (this may include packets containing Errors. This is 64 bit counter.</td>
<td>PortRcvPkts</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_multicast_rcv_packets</td>
<td>Total number of multicast packets, including multicast packets containing errors.</td>
<td>PortMultiCastRcvPkts</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_unicast_rcv_packets</td>
<td>Total number of unicast packets, including unicast packets containing errors.</td>
<td>PortUnicastRcvPkts</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_xmit_data</td>
<td>The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLs from the port.</td>
<td>PortXmitDData</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_xmit_packets</td>
<td>Total number of packets transmitted on all VLs from this port. This may include packets with errors. This is 64 bit counter.</td>
<td>PortXmitPkts</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_xmit_packets_64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>port_rcv_switch_relay_errors</td>
<td>Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay.</td>
<td>PortRcvSwitchRelayErrors</td>
<td>Error</td>
</tr>
<tr>
<td>port_rcv_errors</td>
<td>Total number of packets containing an error that were received on the port.</td>
<td>PortRcvErrors</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_rcv_constraint_errors</td>
<td>Total number of packets received on the switch physical port that are discarded.</td>
<td>PortRcvConstraintErrors</td>
<td>Error</td>
</tr>
<tr>
<td>local_link_integrity_errors</td>
<td>The number of times that the count of local physical errors exceeded the threshold specified by LocalPhyErrors.</td>
<td>LocalLinkIntegrityErrors</td>
<td>Error</td>
</tr>
<tr>
<td>port_xmit_wait</td>
<td>The number of ticks during which the port had data to transmit but no data was sent during the entire tick (either because of insufficient credits or because of lack of arbitration).</td>
<td>PortXmitWait</td>
<td>Informativ</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>InfiniBand Spec Name</td>
<td>Group</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>port_multicast_xmit_packets</td>
<td>Total number of multicast packets transmitted on all VLs from the port. This may include multicast packets with errors.</td>
<td>PortMultiCastXmitPktss</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_unicast_xmit_packets</td>
<td>Total number of unicast packets transmitted on all VLs from the port. This may include unicast packets with errors.</td>
<td>PortUnicastXmitPktts</td>
<td>Informativ</td>
</tr>
<tr>
<td>port_xmit_discards</td>
<td>Total number of outbound packets discarded by the port because the port is down or congested.</td>
<td>PortXmitDiscards</td>
<td>Error</td>
</tr>
<tr>
<td>port_xmit_constraint_errors</td>
<td>Total number of packets not transmitted from the switch physical port.</td>
<td>PortXmitConstraintErrors</td>
<td>Error</td>
</tr>
<tr>
<td>port_rcv_remote_physical_errors</td>
<td>Total number of packets marked with the EBP delimiter received on the port.</td>
<td>PortRcvRemotePhysicalErrors</td>
<td>Error</td>
</tr>
<tr>
<td>symbol_error</td>
<td>Total number of minor link errors detected on one or more physical lanes.</td>
<td>SymbolErrorCounter</td>
<td>Error</td>
</tr>
<tr>
<td>VL15_dropped</td>
<td>Number of incoming VL15 packets dropped due to resource limitations (e.g., lack of buffers) of the port.</td>
<td>VL15Dropped</td>
<td>Error</td>
</tr>
<tr>
<td>link_error_recovery</td>
<td>Total number of times the Port Training state machine has successfully completed the link error recovery process.</td>
<td>LinkErrorRecoveryCounter</td>
<td>Error</td>
</tr>
<tr>
<td>link_downed</td>
<td>Total number of times the Port Training state machine has failed the link error recovery process and downed the link.</td>
<td>LinkDownedCounter</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Hardware Counters**

The hardware counters, found under `/sys/class/infiniband/mlx5_0/ports/1/hw_counters/`, are counted per function and exposed on the function. Some counters are not counted per function. These counters are commented with a relevant comment.
<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplicate_request</td>
<td>Number of received packets. A duplicate request is a request that had been previously executed.</td>
<td>Error</td>
</tr>
<tr>
<td>implied_nak_seq_err</td>
<td>Number of time the requested decided an ACK with a PSN larger than the expected PSN for an RDMA read or response.</td>
<td>Error</td>
</tr>
<tr>
<td>lifespan</td>
<td>The maximum period in ms which defines the aging of the counter reads. Two consecutive reads within this period might return the same values.</td>
<td>Informative</td>
</tr>
<tr>
<td>local_ack_timeout_err</td>
<td>The number of times QP's ack timer expired for RC, XRC, DCT QPs at the sender side. The QP retry limit was not exceed, therefore it is still recoverable error.</td>
<td>Error</td>
</tr>
<tr>
<td>np_cnp_sent</td>
<td>The number of CNP packets sent by the Notification Point when it noticed congestion experienced in the RoCEv2 IP header (ECN bits).</td>
<td>Informative</td>
</tr>
<tr>
<td>np_ecn_marked_roce_packets</td>
<td>The number of RoCEv2 packets received by the notification point which were marked for experiencing the congestion (ECN bits where '11' on the ingress RoCE traffic).</td>
<td>Informative</td>
</tr>
<tr>
<td>out_of_buffer</td>
<td>The number of drops occurred due to lack of WQE for the associated QPs.</td>
<td>Error</td>
</tr>
<tr>
<td>out_of_sequence</td>
<td>The number of out of sequence packets received.</td>
<td>Error</td>
</tr>
<tr>
<td>packet_seq_err</td>
<td>The number of received NAK sequence error packets. The QP retry limit was not exceeded.</td>
<td>Error</td>
</tr>
<tr>
<td>req_cqe_error</td>
<td>The number of times requester detected CQEs completed with errors.</td>
<td>Error</td>
</tr>
<tr>
<td>req_cqe_flush_error</td>
<td>The number of times requester detected CQEs completed with flushed errors.</td>
<td>Error</td>
</tr>
<tr>
<td>req_remote_access_errors</td>
<td>The number of times requester detected remote access errors.</td>
<td>Error</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>req_remote_invalid_request</td>
<td>The number of times requester detected remote invalid request errors.</td>
<td>Error</td>
</tr>
<tr>
<td>resp_cqe_error</td>
<td>The number of times responder detected CQEs completed with errors.</td>
<td>Error</td>
</tr>
<tr>
<td>resp_cqe_flush_error</td>
<td>The number of times responder detected CQEs completed with flushed errors.</td>
<td>Error</td>
</tr>
<tr>
<td>resp_local_length_error</td>
<td>The number of times responder detected local length errors.</td>
<td>Error</td>
</tr>
<tr>
<td>resp_remote_access_errors</td>
<td>The number of times responder detected remote access errors.</td>
<td>Error</td>
</tr>
<tr>
<td>rnr_nak_retry_error</td>
<td>The number of received RNR NAK packets. The QP retry limit was not exceeded.</td>
<td>Error</td>
</tr>
<tr>
<td>rp_cnp_handled</td>
<td>The number of CNP packets handled by the Reaction Point HCA to throttle the transmission rate.</td>
<td>Informative</td>
</tr>
<tr>
<td>rp_cnp_ignored</td>
<td>The number of CNP packets received and ignored by the Reaction Point HCA. This counter should not raise if RoCE Congestion Control was enabled in the network. If this counter raise, verify that ECN was enabled on the adapter. See HowTo Configure DCQCN (RoCE CC) values for ConnectX-4 (Linux).</td>
<td>Error</td>
</tr>
<tr>
<td>rx_atomic_requests</td>
<td>The number of received ATOMIC request for the associated QPs.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_dct_connect</td>
<td>The number of received connection request for the associated DCTs.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_read_requests</td>
<td>The number of received READ requests for the associated QPs.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Group</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>rx_write_request</td>
<td>The number of received WRITE requests for the associated QPs.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_icrc_encapsulated</td>
<td>The number of RoCE packets with ICRC errors.</td>
<td>Error</td>
</tr>
<tr>
<td>roce_adp_retrans</td>
<td>Counts the number of adaptive retransmissions for RoCE traffic</td>
<td>Informative</td>
</tr>
<tr>
<td>roce_adp_retrans_to</td>
<td>Counts the number of times RoCE traffic reached timeout due to adaptive retransmission</td>
<td>Informative</td>
</tr>
<tr>
<td>roce_slow_restart</td>
<td>Counts the number of times RoCE slow restart was used</td>
<td>Informative</td>
</tr>
<tr>
<td>roce_slow_restart_cnps</td>
<td>Counts the number of times RoCE slow restart generated CNP packets</td>
<td>Informative</td>
</tr>
<tr>
<td>roce_slow_restart_trans</td>
<td>Counts the number of times RoCE slow restart changed state to slow restart</td>
<td>Informative</td>
</tr>
<tr>
<td>roce_adp_retrans_to</td>
<td>Counts the number of adaptive retransmissions for RoCE traffic</td>
<td>Informative</td>
</tr>
<tr>
<td>roce_slow_restart</td>
<td>Counts the number of times RoCE traffic reached timeout due to adaptive retransmission</td>
<td>Informative</td>
</tr>
</tbody>
</table>

**Debug Status Counters**

The following parameters are located in `/sys/class/net/<interface>/debug`. 
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>lro_timeout</td>
<td>Sets the LRO timer period value in usecs which will be used as LRO session expiration time. For example:</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td># cat /sys/class/net/eth2/debug/lro_timeout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Actual timeout: 32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported timeout: 8 16 32 1024</td>
<td></td>
</tr>
<tr>
<td>link_down_reason</td>
<td>Link down reason will allow the user to query the reason which is preventing the link from going up. For example:</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>$ cat /sys/class/net/ethXX/debug/link_down_reason</td>
<td></td>
</tr>
<tr>
<td></td>
<td>monitor_opcode: 0x0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>status_message: The port is Active.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refer to the adapter PRM for all possible options (PDDR register).</td>
<td></td>
</tr>
</tbody>
</table>

**Power Thermal Counters**

The `bf_p tm` component collects BlueField-3 power thermal counters using remote collection. It is disabled by default and can be enabled as follows:

1. Load kernel module `mlxbf-ptm`:
   ```
   modprobe -v mlxbf-ptm
   ```

2. Enable component using remote collection:
   ```
   enable-provider=grpc.sysfs.bf_p tm
   ```
Ethtool Counters

Ethtool counters is the generated list of counters which corresponds to Ethtool utility. Counters are generated on a per-device basis.

There are several counter groups, depending on where the counter is counted:

- **Ring** – software ring counters
- **Software port** – an aggregation of software ring counters
- **vPort counters** – traffic counters and drops due to steering or no buffers. May indicate BlueField issues. These counters include Ethernet traffic counters (including raw Ethernet) and RDMA/RoCE traffic counters.
- **Physical port counters** – the physical port connecting BlueField to the network. May indicate device issues or link or network issues. This measuring point holds information on standardized counters like IEEE 802.3, RFC2863, RFC 2819, RFC 3635 and additional counters like flow control, FEC, and more. Physical port counters are not exposed to virtual machines.
- **Priority port counters** – a set of the physical port counters, per priority per port

Each group of counters may have different counter types:

- **Traffic informative counters** – counters which counts traffic. These counters can be used for load estimation of for general debug.
- **Traffic acceleration counters** – counters which counts traffic accelerated by NVIDIA drivers or by hardware. The counters are an additional layer to the informative counter set and the same traffic is counted in both informative and acceleration counters. Acceleration counters are marked with [A].

**Note**

DPE server should be active before changing the `dts_config.ini` file. See section "Remote Collection" for details.
• Error counters – increment of these counters might indicate a problem

The following acceleration mechanisms have dedicated counters:

• TCP segmentation offload (TSO) – increasing outbound throughput and reducing CPU utilization by allowing the kernel to buffer multiple packets in a single large buffer. The BlueField splits the buffer into packet and transmits it.

• Large receive offload (LRO) – increasing inbound throughput and reducing CPU utilization by aggregation of multiple incoming packets of a single stream to a single buffer

• CHECKSUM – calculation of TCP checksum (by the BlueField). The following checksum offloads are available (refer to skbuff.h for detailed explanation)
  - CHECKSUM_UNNECESSARY
  - CHECKSUM_NONE – no checksum acceleration was used
  - CHECKSUM_COMPLETE – device provided checksum on the entire packet
  - CHECKSUM_PARTIAL – device provided checksum

• CQE compress – compression of completion queue events (CQE) used for sparing bandwidth on PCIe and hence achieve better performance.

**Ring/Software Port Counters**

The following counters are available per ring or software port.

These counters provide information on the amount of traffic accelerated by the BlueField. The counters tally the accelerated traffic in addition to the standard counters which tally that (i.e. accelerated traffic is counted twice).

The counter names in the table below refers to both ring and port counters. the notation for ring counters includes the [i] index without the braces. the notation for port counters does not include the [i]. a counter name rx[i]_packets will be printed as rx0_packets for ring 0 and rx_packets for the software port
<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx[i].packets</td>
<td>The number of packets received on ring i.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i].bytes</td>
<td>The number of bytes received on ring i.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx[i].packets</td>
<td>The number of packets transmitted on ring i.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx[i].bytes</td>
<td>The number of bytes transmitted on ring i.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx[i].ts_o.packets</td>
<td>The number of TSO packets transmitted on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i].ts_o.bytes</td>
<td>The number of TSO bytes transmitted on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i].ts_o.inner.packets</td>
<td>The number of TSO packets which are indicated to be carry internal encapsulation transmitted on ring i [A]</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i].ts_o.inner.bytes</td>
<td>The number of TSO bytes which are indicated to be carry internal encapsulation transmitted on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i].lro.packets</td>
<td>The number of LRO packets received on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>kets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rx[i].lro_bytes</td>
<td>The number of LRO bytes received on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i].checksum_unnecessary</td>
<td>Packets received with a CHECKSUM_UNNECESSARY on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i].checksum_none</td>
<td>Packets received with CHECKSUM_NONE on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i].checksum_complete</td>
<td>Packets received with a CHECKSUM_COMPLETE on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i].checksum_unnecessary_inner</td>
<td>Packets received with inner encapsulation with a CHECKSUM_UNNECESSARY on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i].checksum_partial</td>
<td>Packets transmitted with a CHECKSUM_PARTIAL on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i].checksum_partial_inner</td>
<td>Packets transmitted with inner encapsulation with a CHECKSUM_PARTIAL on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i].checksum_none</td>
<td>Packets transmitted with no hardware checksum acceleration on ring i.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>tx[i]_stopped tx_queue_stopped 1</td>
<td>Events where SQ was full on ring i. If this counter is increased, check the amount of buffers allocated for transmission.</td>
<td>Error</td>
</tr>
<tr>
<td>tx[i]_wake tx_queue_wake 1</td>
<td>Events where SQ was full and has become not full on ring i.</td>
<td>Error</td>
</tr>
<tr>
<td>tx[i]_dropped tx_queue_dropped 1</td>
<td>Packets transmitted that were dropped due to DMA mapping failure on ring i. If this counter is increased, check the amount of buffers allocated for transmission.</td>
<td>Error</td>
</tr>
<tr>
<td>rx[i]_wqe_err</td>
<td>The number of wrong opcodes received on ring i.</td>
<td>Error</td>
</tr>
<tr>
<td>tx[i]_nop</td>
<td>The number of no WQEs (empty WQEs) inserted to the SQ (related to ring i) due to the reach of the end of the cyclic buffer. When reaching near to the end of cyclic buffer the driver may add those empty WQEs to avoid handling a state the a WQE start in the end of the queue and ends in the beginning of the queue. This is a normal condition.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_mpwq_e_frag</td>
<td>The number of WQEs that failed to allocate compound page and hence fragmented MPWQE's (multipacket WQEs) were used on ring i. If this counter raise, it may suggest that there is no enough memory for large pages, the driver allocated fragmented pages. This is not abnormal condition.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>rx[i]_mpwqe_filler_cqes</td>
<td>The number of filler CQEs events that were issued on ring i.</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td><em>Info</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The counter name before kernel 4.19 was rx[i]_mpwqe_filler.</td>
<td></td>
</tr>
<tr>
<td>rx[i]_cqecompress_blks</td>
<td>The number of receive blocks with CQE compression on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i]_cqecompress_pkts</td>
<td>The number of receive packets with CQE compression on ring i [A].</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i]_cache_reuse</td>
<td>The number of events of successful reuse of a page from a driver's internal page cache</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i]_cache_full</td>
<td>The number of events of full internal page cache where driver can't put a page back to the cache for recycling (page will be freed)</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i]_cache_empty</td>
<td>The number of events where cache was empty - no page to give. driver shall allocate new page</td>
<td>Acceleration</td>
</tr>
<tr>
<td>rx[i]_cache_busy</td>
<td>The number of events where cache head was busy and cannot be recycled. driver allocated new page</td>
<td>Acceleration</td>
</tr>
<tr>
<td><strong>Counter</strong></td>
<td><strong>Description</strong></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>rx[i]_xmit_more</td>
<td>The number of packets sent with xmit_more indication set on the skbuff (no doorbell)</td>
<td>Acceleration</td>
</tr>
<tr>
<td>tx[i]_cques</td>
<td>The number of completions received on the CQ of TX ring.</td>
<td>Informative</td>
</tr>
<tr>
<td>ch[i]_poll</td>
<td>The number of invocations of NAPI poll of channel.</td>
<td>Informative</td>
</tr>
<tr>
<td>ch[i]_arm</td>
<td>The number of times the NAPI poll function completed and armed the completion queues on channel</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td><strong>Info</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.19.</td>
<td></td>
</tr>
<tr>
<td>ch[i]_aff_change</td>
<td>The number of times the NAPI poll function explicitly stopped execution on a CPU due to a change in affinity, on channel.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_congst_umr</td>
<td>The number of times an outstanding UMR request is delayed due to congestion, on ring.</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td><strong>Info</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.19.</td>
<td></td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ch[i]_e vents</td>
<td>The number of hard interrupt events on the completion queues of channel.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_mpwq_e_filler_stride</td>
<td>The number of strides consumed by filler CQEs on ring.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_xdp_tx_xmit</td>
<td>The number of packets forwarded back to the port due to XDP program XDP_TX action (bouncing). these packets are not counted by other software counters. These packets are counted by physical port and vPort counters. You may open more rx queues and spread traffic rx over all queues and/or increase rx ring size.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_xdp_tx_full</td>
<td>The number of packets that should have been forwarded back to the port due to XDP_TX action but were dropped due to full tx queue. these packets are not counted by other software counters. These packets are counted by physical port and vPort counters. You may open more rx queues and spread traffic rx over all queues and/or increase rx ring size.</td>
<td>Error</td>
</tr>
<tr>
<td>rx[i]_xdp_tx_err</td>
<td>The number of times an XDP_TX error such as frame too long and frame too short occurred on XDP_TX ring of RX ring.</td>
<td>Error</td>
</tr>
<tr>
<td>rx[i]_xdp_tx_cqes</td>
<td>The number of completions received on the CQ of the XDP-TX ring.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_xdp_dp_drop</td>
<td>The number of packets dropped due to XDP program XDP_DROP action. these packets are not counted by other software counters. These packets are counted by physical port and vPort counters.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_xdp_redirect</td>
<td>The number of times an XDP redirect action has been triggered on ring.</td>
<td>Acceleration</td>
</tr>
</tbody>
</table>
## Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx[i]_xdp_xmit</td>
<td>The number of packets redirected to the interface (due to XDP redirect). These packets are not counted by other software counters. These packets are counted by physical port and vPort counters.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx[i]_xdp_full</td>
<td>The number of packets redirected to the interface (due to XDP redirect) but were dropped due to the Tx queue being full. These packets are not counted by other software counters. Users may enlarge Tx queues.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx[i]_xdp_err</td>
<td>The number of packets redirected to the interface (due to XDP redirect) but were dropped due to an error (e.g., frame too long and frame too short).</td>
<td>Error</td>
</tr>
<tr>
<td>tx[i]_xdp_cqes</td>
<td>The number of completions received for packets redirected to the interface (due to XDP redirect) on the CQ.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx[i]_cache_waive</td>
<td>The number of cache evacuation. This can occur due to page move to another NUMA node or page was pfmemalloc-ed and should be freed as soon as possible.</td>
<td>Acceleration</td>
</tr>
</tbody>
</table>

1. The corresponding ring and global counters do not share the same name (i.e., do not follow the common naming scheme).  

### vPort Counters

Counters on the eswitch port that is connected to the vNIC.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_vport_unicast_packets</td>
<td>Unicast packets received, steered to a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_vport_unicast_bytes</td>
<td>Unicast bytes received, steered to a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>tx_vport_unicast _packets</td>
<td>Unicast packets transmitted, steered from a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>tx_vport_unicast _bytes</td>
<td>Unicast bytes transmitted, steered from a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>rx_vport_multicast_packets</td>
<td>Multicast packets received, steered to a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>rx_vport_multicast_bytes</td>
<td>Multicast bytes received, steered to a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>tx_vport_multicast_packets</td>
<td>Multicast packets transmitted, steered from a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>tx_vport_multicast_bytes</td>
<td>Multicast bytes transmitted, steered from a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>rx_vport_broadcast_packets</td>
<td>Broadcast packets received, steered to a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>rx_vport_broadcast_bytes</td>
<td>Broadcast bytes received, steered to a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>tx_vport_broadcast_packets</td>
<td>Broadcast packets transmitted, steered from a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>tx_vport_broadcast_bytes</td>
<td>Broadcast packets transmitted, steered from a port including raw Ethernet QP/DPDK traffic, excluding RDMA traffic</td>
<td>Informativ e</td>
</tr>
<tr>
<td>rx_vport_rdma_unicast_packets</td>
<td>RDMA unicast packets received, steered to a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Acceleration</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>rx_vport_rdma_unicast_bytes</td>
<td>RDMA unicast bytes received, steered to a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>tx_vport_rdma_unicast_packets</td>
<td>RDMA unicast packets transmitted, steered from a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>tx_vport_rdma_unicast_bytes</td>
<td>RDMA unicast bytes transmitted, steered from a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>rx_vport_rdma_multicast_packets</td>
<td>RDMA multicast packets received, steered to a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>rx_vport_rdma_multicast_bytes</td>
<td>RDMA multicast bytes received, steered to a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>tx_vport_rdma_multicast_packets</td>
<td>RDMA multicast packets transmitted, steered from a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>tx_vport_rdma_multicast_bytes</td>
<td>RDMA multicast bytes transmitted, steered from a port (counters counts RoCE/UD/RC traffic) [A]</td>
<td>Accele</td>
</tr>
<tr>
<td>rx_steer_missed_packets</td>
<td>Number of packets received by the NIC but discarded due to not matching any flow in the NIC flow table.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_packets</td>
<td>Representor only: packets received, that were handled by the hypervisor.</td>
<td>Infor</td>
</tr>
<tr>
<td>rx_bytes</td>
<td>Representor only: bytes received, that were handled by the hypervisor.</td>
<td>Infor</td>
</tr>
</tbody>
</table>

![Detail](https://via.placeholder.com/150)
### tx_packets

Representor only: packets transmitted which have been handled by the hypervisor.

- **Info**
  - Supported from kernel 4.18.

### tx_bytes

Representor only: bytes transmitted which have been handled by the hypervisor.

- **Info**
  - Supported from kernel 4.18.

**Physical Port Counters**

The physical port counters are the counters on the external port connecting adapter to the network. This measuring point holds information on standardized counters like IEEE 802.3, RFC2863, RFC 2819, RFC 3635 and additional counters like flow control, FEC and more.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_packets_</td>
<td>The number of packets received on the physical port. This counter doesn't include packets that were discarded due to FCS, frame size and similar</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>phy</td>
<td>errors.</td>
<td></td>
</tr>
<tr>
<td>tx_packets_phy</td>
<td>The number of packets transmitted on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_bytes_phy</td>
<td>The number of bytes received on the physical port, including Ethernet header and FCS.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_bytes_phy</td>
<td>The number of bytes transmitted on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_multicast_phy</td>
<td>The number of multicast packets received on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_multicast_phy</td>
<td>The number of multicast packets transmitted on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_broadcast_phy</td>
<td>The number of broadcast packets received on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_broadcast</td>
<td>The number of broadcast packets transmitted on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>t_phy</td>
<td></td>
<td>informative</td>
</tr>
<tr>
<td>rx_crc_error_s_phy</td>
<td>The number of dropped received packets due to frame check sequence (FCS) error on the physical port. If this counter is increased in high rate, check the link quality using <code>rx_symbol_error_phy</code> and <code>rx_corrected_bits_phy</code> counters below.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_in_range_len_errors_phy</td>
<td></td>
<td>Error</td>
</tr>
<tr>
<td>rx_out_of_range_len_phy</td>
<td>The number of received packets dropped due to length greater than allowed on a physical port. If this counter is increasing, it implies that the peer connected to the adapter has a larger MTU configured. Using same MTU configuration shall resolve this issue.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_oversize_pkts_phy</td>
<td>The number of dropped received packets due to length which exceed MTU size on a physical port. If this counter is increasing, it implies that the peer connected to the adapter has a larger MTU configured. Using same MTU configuration shall resolve this issue.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_symbol_err_phy</td>
<td>The number of received packets dropped due to physical coding errors (symbol errors) on a physical port.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_mac_control_phy</td>
<td>The number of MAC control packets received on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_mac_control_phy</td>
<td>The number of MAC control packets transmitted on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>rx_pause_control_phy</td>
<td>The number of link layer pause packets received on a physical port. If this counter is increasing, it implies that the network is congested and cannot absorb the traffic coming from to the adapter.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_pause_control_phy</td>
<td>The number of link layer pause packets transmitted on a physical port. If this counter is increasing, it implies that the NIC is congested and cannot absorb the traffic coming from the network.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_unsupported_opcode_phy</td>
<td>The number of MAC control packets received with unsupported opcode on a physical port.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_discards_phy</td>
<td>The number of received packets dropped due to lack of buffers on a physical port. If this counter is increasing, it implies that the adapter is congested and cannot absorb the traffic coming from the network.</td>
<td>Error</td>
</tr>
<tr>
<td>tx_discards_phy</td>
<td>The number of packets which were discarded on transmission, even no errors were detected. The drop might occur due to link in down state, head of line drop, pause from the network, etc.</td>
<td>Error</td>
</tr>
<tr>
<td>tx_errors_phy</td>
<td>The number of transmitted packets dropped due to a length which exceed MTU size on a physical port.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_undersize_pkts_phy</td>
<td>The number of received packets dropped due to length which is shorter than 64 bytes on a physical port. If this counter is increasing, it implies that the peer connected to the adapter has a non-standard MTU configured or malformed packet had arrived.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_fragment_phy</td>
<td>The number of received packets dropped due to a length which is shorter than 64 bytes and has FCS error on a physical port. If this counter is increasing, it implies that the peer connected to the adapter has a non-standard MTU configured.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_jabbers_phy</td>
<td>The number of received packets dropped due to a length which is longer than 64 bytes and had FCS error on a physical port.</td>
<td>Error</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>rx_64_bytes_phy</td>
<td>The number of packets received on the physical port with size of 64 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_65_to_127_bytes_phy</td>
<td>The number of packets received on the physical port with size of 65 to 127 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_128_to_255_bytes_phy</td>
<td>The number of packets received on the physical port with size of 128 to 255 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_256_to_511_bytes_phy</td>
<td>The number of packets received on the physical port with size of 256 to 512 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_512_to_1023_bytes_phy</td>
<td>The number of packets received on the physical port with size of 512 to 1023 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_1024_to_1518_bytes_phy</td>
<td>The number of packets received on the physical port with size of 1024 to 1518 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_1519_to_2047_bytes_phy</td>
<td>The number of packets received on the physical port with size of 1519 to 2047 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>rx_2048_to_4095_bytes_phy</td>
<td>The number of packets received on the physical port with size of 2048 to 4095 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_4096_to_8191_bytes_phy</td>
<td>The number of packets received on the physical port with size of 4096 to 8191 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_8192_to_10239_bytes_phy</td>
<td>The number of packets received on the physical port with size of 8192 to 10239 bytes.</td>
<td>Informative</td>
</tr>
<tr>
<td>link_down_events_phy</td>
<td>The number of times where the link operative state changed to down. In case this counter is increasing it may imply on port flapping. You may need to replace the cable/transceiver.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_out_of_buffer</td>
<td>Number of times receive queue had no software buffers allocated for the adapter's incoming traffic.</td>
<td>Error</td>
</tr>
<tr>
<td>module_bus_stuck</td>
<td>The number of times that module's i\textsuperscript{2}C bus (data or clock) short-wire was detected. You may need to replace the cable/transceiver.</td>
<td>Error</td>
</tr>
<tr>
<td>module_hightemp</td>
<td>The number of times that the module temperature was too high. If this issue persists, you may need to check the ambient temperature or replace the cable/transceiver module.</td>
<td>Error</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>module_bad_shorted</td>
<td>The number of times that the module cables were shorted. You may need to replace the cable/transceiver module.</td>
<td>Error</td>
</tr>
<tr>
<td>module_unplugged</td>
<td>The number of times that module was ejected.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_buffer_passed_threshold</td>
<td>The number of events where the port receive buffer was over 85% full.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_pause_storm_warning_events</td>
<td>The number of times the device was sending pauses for a long period of time.</td>
<td>Informative</td>
</tr>
</tbody>
</table>

- **Info**: Supported from kernel 4.10.
- **Error**: Supported from kernel 4.10.
- **Informative**: Supported from kernel 4.14.
<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx_pause_storm_error_events</td>
<td>The number of times the device was sending pauses for a long period of time, reaching time out and disabling transmission of pause frames. On the period where pause frames were disabled, drop could have been occurred.</td>
<td>Error</td>
</tr>
<tr>
<td>rx[i]_buff_allc_err / rx_buff_alloc_err</td>
<td>Failed to allocate a buffer to received packet (or SKB) on port (or per ring)</td>
<td>Error</td>
</tr>
<tr>
<td>rx_bits_phy</td>
<td>This counter provides information on the total amount of traffic that could have been received and can be used as a guideline to measure the ratio of errored traffic in rx_pcs_symbol_err_phy and rx_corrected_bits_phy.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_pcs_symbol_err_phy</td>
<td>This counter counts the number of symbol errors that wasn't corrected by FEC correction algorithm or that FEC algorithm was not active on this interface. If this counter is increasing, it implies that the link between the NIC and the network is suffering from high BER, and that traffic is lost. You may need to replace the cable/transceiver. The error rate is the number of rx_pcs_symbol_err_phy divided by the number of rx_phy_bits on a specific time frame.</td>
<td>Error</td>
</tr>
<tr>
<td>rx_corrected_bits_phy</td>
<td>The number of corrected bits on this port according to active FEC (RS/FC). If this counter is increasing, it implies that the link between the NIC and the network is suffering from high BER. The corrected bit rate is the number of</td>
<td>Error</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>rx_corrected_bits_phy</td>
<td>divided by the number of rx_phy_bits on a specific time frame.</td>
<td></td>
</tr>
<tr>
<td>phy_raw_errors_lane[l]</td>
<td>This counter counts the number of physical raw errors per lane [l] index. The counter counts errors before FEC corrections. If this counter is increasing, it implies that the link between the NIC and the network is suffering from high BER, and that traffic might be lost. You may need to replace the cable/transceiver. Please check in accordance with rx_corrected_bits_phy .</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Priority Port Counters**

The following counters are physical port counters that being counted per L2 priority (0-7).

<table>
<thead>
<tr>
<th>Info</th>
<th>Supported from kernel 4.20.</th>
</tr>
</thead>
</table>

_{p} in the counter name represents the priority.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_prio[p]_bytes</td>
<td>The number of bytes received with priority p on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>rx_prio[p]_packets</td>
<td>The number of packets received with priority p on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_prio[p]_bytes</td>
<td>The number of bytes transmitted on priority p on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_prio[p]_packets</td>
<td>The number of packets transmitted on priority p on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_prio[p]_pause</td>
<td>The number of pause packets received with priority p on a physical port. If this counter is increasing, it implies that the network is congested and cannot absorb the traffic coming from the adapter. <strong>Note:</strong> This counter is available only if PFC was enabled on priority p. Refer to HowTo Configure PFC on ConnectX-4.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_prio[p]_pause_duration</td>
<td>The duration of pause received (in microSec) on priority p on the physical port. The counter represents the time the port did not send any traffic on this priority. If this counter is increasing, it implies that the network is congested and cannot absorb the traffic coming from the adapter. <strong>Note:</strong> This counter is available only if PFC was enabled on priority p. Refer to HowTo Configure PFC on ConnectX-4.</td>
<td>Informative</td>
</tr>
<tr>
<td>rx_prio[p]_pause_transition</td>
<td>The number of times a transition from Xoff to Xon on priority p on the physical port has occurred. <strong>Note:</strong> This counter is available only if PFC was enabled on priority p. Refer to HowTo Configure PFC on ConnectX-4.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_prio[p]_pause</td>
<td>The number of pause packets transmitted on priority p on a physical port. If this counter is increasing, it implies that the adapter is congested and cannot absorb the traffic coming from the network. <strong>Note:</strong> This counter is available only if PFC was enabled on priority p. Refer to HowTo Configure PFC on ConnectX-4.</td>
<td>Informative</td>
</tr>
<tr>
<td>tx_prio[p]_pause</td>
<td>The duration of pause transmitter (in microSec) on priority p on the physical port.</td>
<td>Informative</td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>use_duration</td>
<td><strong>Note:</strong> This counter is available only if PFC was enabled on priority p. Refer to HowTo Configure PFC on ConnectX-4.</td>
<td>active</td>
</tr>
<tr>
<td>rx_prio[p].buf_discard</td>
<td>The number of packets discarded by device due to lack of per host receive buffers.</td>
<td>Info</td>
</tr>
<tr>
<td>rx_prio[p].cong_discard</td>
<td>The number of packets discarded by device due to per host congestion.</td>
<td>Info</td>
</tr>
<tr>
<td>rx_prio[p].marked</td>
<td>The number of packets ecn marked by device due to per host congestion.</td>
<td>Info</td>
</tr>
<tr>
<td>rx_prio[p].discard</td>
<td>The number of packets discarded by device due to lack of receive buffers.</td>
<td>Info</td>
</tr>
</tbody>
</table>

- **Info:** Supported from kernel 5.3.
- **Info:** Supported from kernel 5.6.
## Device Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rx_pci_signal_integrity</code></td>
<td>Counts physical layer PCIe signal integrity errors, the number of transitions to recovery due to Framing errors and CRC (dlp and tlp). If this counter is raising, try moving the adapter card to a different slot to rule out a bad PCIe slot. Validate that you are running with the latest firmware available and latest server BIOS version.</td>
<td>Error</td>
</tr>
<tr>
<td><code>tx_pci_signal_integrity</code></td>
<td>Counts physical layer PCIe signal integrity errors, the number of transition to recovery initiated by the other side (moving to recovery due to getting TS/EIEOS). If this counter is raising, try moving the adapter card to a different slot to rule out a bad PCI slot. Validate that you are running with the latest firmware available and latest server BIOS version.</td>
<td>Error</td>
</tr>
<tr>
<td><code>outbound_pci_buffer_overflow</code></td>
<td>The number of packets dropped due to pci buffer overflow. If this counter is raising in high rate, it might indicate that the receive traffic rate for a host is larger than the PCIe bus and therefore a congestion occurs.</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td><strong>Info</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.14.</td>
<td></td>
</tr>
<tr>
<td><code>outbound_pci_stalled_rd</code></td>
<td>The percentage (in the range 0...100) of time within the last second that the NIC had outbound non-posted reads requests but could not perform the operation due to insufficient posted credits.</td>
<td>Informative</td>
</tr>
<tr>
<td></td>
<td><strong>Info</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.14.</td>
<td></td>
</tr>
<tr>
<td>Counter</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>outbound_pci_stalled_wr</td>
<td>The percentage (in the range 0...100) of time within the last second that the NIC had outbound posted writes requests but could not perform the operation due to insufficient posted credits.</td>
<td>Info</td>
</tr>
<tr>
<td></td>
<td><strong>i</strong> Info</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.14.</td>
<td></td>
</tr>
<tr>
<td>outbound_pci_stalled_rd_ev</td>
<td>The number of seconds where <code>outbound_pci_stalled_rd</code> was above 30%.</td>
<td>Info</td>
</tr>
<tr>
<td></td>
<td><strong>i</strong> Info</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.14.</td>
<td></td>
</tr>
<tr>
<td>outbound_pci_stalled_wr_ev</td>
<td>The number of seconds where <code>outbound_pci_stalled_wr</code> was above 30%.</td>
<td>Info</td>
</tr>
<tr>
<td></td>
<td><strong>i</strong> Info</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supported from kernel 4.14.</td>
<td></td>
</tr>
<tr>
<td>dev_out_of_buffer</td>
<td>The number of times the device owned queue had not enough buffers allocated.</td>
<td>Error</td>
</tr>
</tbody>
</table>

**Full List of Counters**

```bash
# ethtool -S eth5

NIC statistics:
rx_packets: 10
```
rx_bytes: 3420
rx_packets: 18
rx_bytes: 1296
tx_packets: 0
tx_bytes: 0
tx_tso_packets: 0
tx_tso_bytes: 0
tx_tso_inner_packets: 0
tx_tso_inner_bytes: 0
tx_added_vlan_packets: 0
tx_nop: 0
rx_lro_packets: 0
rx_lro_bytes: 0
rx_ecn_mark: 0
rx_removed_vlan_packets: 0
rx_csum_unnecessary: 0
rx_csum_none: 0
rx_csum_complete: 10
rx_csum_unnecessary_inner: 0
rx_xdp_drop: 0
rx_xdp_redirect: 0
rx_xdp_tx_xmit: 0
rx_xdp_tx_full: 0
rx_xdp_tx_err: 0
rx_xdp_tx_cqe: 0
tx_csum_none: 18
tx_csum_partial: 0
tx_csum_partial_inner: 0
tx_queue_stopped: 0
tx_queue_dropped: 0
tx_xmit_more: 0
tx_recover: 0
tx_cqes: 18
tx_queue_wake: 0
tx_udp_seg_rem: 0
tx_cqe_err: 0
tx_xdp_xmit: 0
tx_xdp_full: 0
tx_xdp_err: 0
tx_xdp_cqes: 0
rx_wqe_err: 0
rx_mpqwe_filler_cqes: 0
rx_mpqwe_filler_strides: 0
rx_buff_alloc_err: 0
rx_cqe_compress_blks: 0
rx_cqe_compress_pks: 0
rx_page_reuse: 0
rx_cache_reuse: 0
rx_cache_full: 0
rx_cache_empty: 2688
rx_cache_busy: 0
rx_cache_waive: 0
rx_congst_umr: 0
rx_arfs_err: 0
ch_events: 75
ch_poll: 75
ch_arm: 75
ch_aff_change: 0
ch_eq_rearm: 0
rx_out_of_buffer: 0
rx_if_down_packets: 15
rx_steer_missed_packets: 0
rx_vport_unicast_packets: 0
rx_vport_unicast_bytes: 0
tx_vport_unicast_packets: 0
tx_vport_unicast_bytes: 0
rx_vport_multicast_packets: 2
rx_vport_multicast_bytes: 172
tx_vport_multicast_packets: 12
tx_vport_multicast_bytes: 936
rx_vport_broadcast_packets: 37
rx_vport_broadcast_bytes: 9270
tx_vport_broadcast_packets: 6
tx_vport_broadcast_bytes: 360
rx_vport_rdma_unicast_packets: 0
rx_vport_rdma_unicast_bytes: 0
tx_vport_rdma_unicast_packets: 0
tx_vport_rdma_unicast_bytes: 0
rx_vport_rdma_multicast_packets: 0
rx_vport_rdma_multicast_bytes: 0
tx_vport_rdma_multicast_packets: 0
tx_vport_rdma_multicast_bytes: 0
tx_packets_phy: 0
rx_packets_phy: 0
rx_crc_errors_phy: 0
tx_bytes_phy: 0
rx_bytes Phy: 0
tx_multicast Phy: 0
tx_broadcast Phy: 0
rx_multicast Phy: 0
rx_broadcast_phy: 0
rx_in_range_len_errors_phy: 0
rx_out_of_range_len_phy: 0
rx_oversize_pkts_phy: 0
rx_symbol_err_phy: 0
tx_mac_control_phy: 0
rx_mac_control_phy: 0
rx_unsupported_op_phy: 0
rx_pause_ctrl_phy: 0
tx_pause_ctrl_phy: 0
rx_discards_phy: 0
tx_discards_phy: 0
tx_errors_phy: 0
rx_undersize_pkts_phy: 0
rx_fragments_phy: 0
rx_jabbers_phy: 0
rx_64_bytes_phy: 0
rx_65_to_127_bytes_phy: 0
rx_128_to_255_bytes_phy: 0
rx_256_to_511_bytes_phy: 0
rx_512_to_1023_bytes_phy: 0
rx_1024_to_1518_bytes_phy: 0
rx_1519_to_2047_bytes_phy: 0
rx_2048_to_4095_bytes_phy: 0
rx_4096_to_8191_bytes_phy: 0
rx_8192_to_10239_bytes_phy: 0
link_down_events_phy: 0
rx_prio0_bytes: 0
rx_prio0_packets: 0
tx_prio0_bytes: 0
tx_prio0_packets: 0
rx_prio1_bytes: 0
rx_prio1_packets: 0
tx_prio1_bytes: 0
tx_prio1_packets: 0
rx_prio2_bytes: 0
rx_prio2_packets: 0
tx_prio2_bytes: 0
tx_prio2_packets: 0
rx_prio3_bytes: 0
rx_prio3_packets: 0
tx_prio3_bytes: 0
tx_prio3_packets: 0
rx_prio4_bytes: 0

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>rx_prio4_packets</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio4_bytes</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio4_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx_prio5_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx_prio5_packets</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio5_bytes</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio5_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx_prio6_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx_prio6_packets</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio6_bytes</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio6_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx_prio7_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx_prio7_packets</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio7_bytes</td>
<td>0</td>
</tr>
<tr>
<td>tx_prio7_packets</td>
<td>0</td>
</tr>
<tr>
<td>module_unplug</td>
<td>0</td>
</tr>
<tr>
<td>module_bus_stuck</td>
<td>0</td>
</tr>
<tr>
<td>module_high_temp</td>
<td>0</td>
</tr>
<tr>
<td>module_bad_shorted</td>
<td>0</td>
</tr>
<tr>
<td>ch0_events</td>
<td>9</td>
</tr>
<tr>
<td>ch0_poll</td>
<td>9</td>
</tr>
<tr>
<td>ch0_arm</td>
<td>9</td>
</tr>
<tr>
<td>ch0_aff_change</td>
<td>0</td>
</tr>
<tr>
<td>ch0_eq_rearm</td>
<td>0</td>
</tr>
<tr>
<td>ch1_events</td>
<td>23</td>
</tr>
<tr>
<td>ch1_poll</td>
<td>23</td>
</tr>
<tr>
<td>ch1_arm</td>
<td>23</td>
</tr>
<tr>
<td>ch1_aff_change</td>
<td>0</td>
</tr>
<tr>
<td>ch1_eq_rearm</td>
<td>0</td>
</tr>
<tr>
<td>ch2_events</td>
<td>8</td>
</tr>
<tr>
<td>ch2_poll</td>
<td>8</td>
</tr>
<tr>
<td>ch2_arm</td>
<td>8</td>
</tr>
<tr>
<td>ch2_aff_change</td>
<td>0</td>
</tr>
<tr>
<td>ch2_eq_rearm</td>
<td>0</td>
</tr>
<tr>
<td>ch3_events</td>
<td>19</td>
</tr>
<tr>
<td>ch3_poll</td>
<td>19</td>
</tr>
<tr>
<td>ch3_arm</td>
<td>19</td>
</tr>
<tr>
<td>ch3_aff_change</td>
<td>0</td>
</tr>
<tr>
<td>ch3_eq_rearm</td>
<td>0</td>
</tr>
<tr>
<td>ch4_events</td>
<td>8</td>
</tr>
<tr>
<td>ch4_poll</td>
<td>8</td>
</tr>
<tr>
<td>ch4_arm</td>
<td>8</td>
</tr>
<tr>
<td>ch4_aff_change</td>
<td>0</td>
</tr>
<tr>
<td>ch4_eq_rearm</td>
<td>0</td>
</tr>
<tr>
<td>Metric</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>ch5_events</td>
<td>8</td>
</tr>
<tr>
<td>ch5_poll</td>
<td>8</td>
</tr>
<tr>
<td>ch5_arm</td>
<td>8</td>
</tr>
<tr>
<td>ch5_aff_change</td>
<td>0</td>
</tr>
<tr>
<td>ch5_eq_rearm</td>
<td>0</td>
</tr>
<tr>
<td>rx0_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx0_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx0_csum_complete</td>
<td>0</td>
</tr>
<tr>
<td>rx0_csum_unnecessary</td>
<td>0</td>
</tr>
<tr>
<td>rx0_csum_unnecessary_inner</td>
<td>0</td>
</tr>
<tr>
<td>rx0_csum_none</td>
<td>0</td>
</tr>
<tr>
<td>rx0_xdp_drop</td>
<td>0</td>
</tr>
<tr>
<td>rx0_xdp_redirect</td>
<td>0</td>
</tr>
<tr>
<td>rx0_lro_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx0_lro_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx0_ecn_mark</td>
<td>0</td>
</tr>
<tr>
<td>rx0_removed_vlan_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx0_wqe_err</td>
<td>0</td>
</tr>
<tr>
<td>rx0_mpwqe_filler_cques</td>
<td>0</td>
</tr>
<tr>
<td>rx0_mpwqe_filler_strides</td>
<td>0</td>
</tr>
<tr>
<td>rx0_buff_alloc_err</td>
<td>0</td>
</tr>
<tr>
<td>rx0_cqe_compress_blks</td>
<td>0</td>
</tr>
<tr>
<td>rx0_cqe_compress_pkts</td>
<td>0</td>
</tr>
<tr>
<td>rx0_pageReuse</td>
<td>0</td>
</tr>
<tr>
<td>rx0_cacheReuse</td>
<td>0</td>
</tr>
<tr>
<td>rx0_cacheFull</td>
<td>0</td>
</tr>
<tr>
<td>rx0_cacheEmpty</td>
<td>448</td>
</tr>
<tr>
<td>rx0_cacheBusy</td>
<td>0</td>
</tr>
<tr>
<td>rx0_cacheWaive</td>
<td>0</td>
</tr>
<tr>
<td>rx0_congst_umr</td>
<td>0</td>
</tr>
<tr>
<td>rx0_arfs_err</td>
<td>0</td>
</tr>
<tr>
<td>rx0_xdp_tx_xmit</td>
<td>0</td>
</tr>
<tr>
<td>rx0_xdp_tx_full</td>
<td>0</td>
</tr>
<tr>
<td>rx0_xdp_tx_err</td>
<td>0</td>
</tr>
<tr>
<td>rx0_xdp_tx_cqes</td>
<td>0</td>
</tr>
<tr>
<td>rx1_packets</td>
<td>10</td>
</tr>
<tr>
<td>rx1_bytes</td>
<td>3420</td>
</tr>
<tr>
<td>rx1_csum_complete</td>
<td>10</td>
</tr>
<tr>
<td>rx1_csum_unnecessary</td>
<td>0</td>
</tr>
<tr>
<td>rx1_csum_unnecessary_inner</td>
<td>0</td>
</tr>
<tr>
<td>rx1_csum_none</td>
<td>0</td>
</tr>
<tr>
<td>rx1_xdp_drop</td>
<td>0</td>
</tr>
<tr>
<td>rx1_xdp_redirect</td>
<td>0</td>
</tr>
<tr>
<td>rx1_lro_packets</td>
<td>0</td>
</tr>
<tr>
<td>Metric</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>rx1_lro_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx1_ecn_mark</td>
<td>0</td>
</tr>
<tr>
<td>rx1_removed_vlan_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx1_wqe_err</td>
<td>0</td>
</tr>
<tr>
<td>rx1_mpwqe_filler_cqes</td>
<td>0</td>
</tr>
<tr>
<td>rx1_mpwqe_filler_strides</td>
<td>0</td>
</tr>
<tr>
<td>rx1_buff_alloc_err</td>
<td>0</td>
</tr>
<tr>
<td>rx1_cqe_compress_blks</td>
<td>0</td>
</tr>
<tr>
<td>rx1_cqe_compress_pkts</td>
<td>0</td>
</tr>
<tr>
<td>rx1_page_reuse</td>
<td>0</td>
</tr>
<tr>
<td>rx1_cache_reuse</td>
<td>0</td>
</tr>
<tr>
<td>rx1_cache_full</td>
<td>0</td>
</tr>
<tr>
<td>rx1_cache_empty</td>
<td>448</td>
</tr>
<tr>
<td>rx1_cache_busy</td>
<td>0</td>
</tr>
<tr>
<td>rx1_cache_waive</td>
<td>0</td>
</tr>
<tr>
<td>rx1_congst_umr</td>
<td>0</td>
</tr>
<tr>
<td>rx1_arfs_err</td>
<td>0</td>
</tr>
<tr>
<td>rx1_xdp_tx_xmit</td>
<td>0</td>
</tr>
<tr>
<td>rx1_xdp_tx_full</td>
<td>0</td>
</tr>
<tr>
<td>rx1_xdp_tx_err</td>
<td>0</td>
</tr>
<tr>
<td>rx1_xdp_tx_cqes</td>
<td>0</td>
</tr>
<tr>
<td>rx2_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx2_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx2_csum_complete</td>
<td>0</td>
</tr>
<tr>
<td>rx2_csum_unnecessary</td>
<td>0</td>
</tr>
<tr>
<td>rx2_csum_unnecessary_inner</td>
<td>0</td>
</tr>
<tr>
<td>rx2_csum_none</td>
<td>0</td>
</tr>
<tr>
<td>rx2_xdp_drop</td>
<td>0</td>
</tr>
<tr>
<td>rx2_xdp_redirect</td>
<td>0</td>
</tr>
<tr>
<td>rx2_lro_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx2_lro_bytes</td>
<td>0</td>
</tr>
<tr>
<td>rx2_ecn_mark</td>
<td>0</td>
</tr>
<tr>
<td>rx2_removed_vlan_packets</td>
<td>0</td>
</tr>
<tr>
<td>rx2_wqe_err</td>
<td>0</td>
</tr>
<tr>
<td>rx2_mpwqe_filler_cqes</td>
<td>0</td>
</tr>
<tr>
<td>rx2_mpwqe_filler_strides</td>
<td>0</td>
</tr>
<tr>
<td>rx2_buff_alloc_err</td>
<td>0</td>
</tr>
<tr>
<td>rx2_cqe_compress_blks</td>
<td>0</td>
</tr>
<tr>
<td>rx2_cqe_compress_pkts</td>
<td>0</td>
</tr>
<tr>
<td>rx2_page_reuse</td>
<td>0</td>
</tr>
<tr>
<td>rx2_cache_reuse</td>
<td>0</td>
</tr>
<tr>
<td>rx2_cache_full</td>
<td>0</td>
</tr>
<tr>
<td>rx2_cache_empty</td>
<td>448</td>
</tr>
<tr>
<td>rx2_cache_busy</td>
<td>0</td>
</tr>
</tbody>
</table>
rx2_cache_waive: 0
rx2_congst Frm: 0
rx2_arfs_err: 0
rx2_xdp_tx_xmit: 0
rx2_xdp_tx_full: 0
rx2_xdp_tx_err: 0
rx2_xdp_tx_cqes: 0
...

...
### Traffic Control Info

The following TC objects are supported and reported regarding the ingress filters:

- Filters
  - `flower`
- Actions
  - `mirred`
  - `tunnel_key`

The info is provided as one of the following events:

- Basic filter event
- Flower/IPv4 filter event
• Flower/IPv6 filter event
• Basic action event
• Mirred action event
• Tunnel_key/IPv4 action event
• Tunnel_key/IPv6 action event

General notes:
• Actions always belong to a filter, so action events share the filter event's ID via the event_id data member
• Basic filter event only contains textual kind (so users can see which real life objects' support they are lacking)
• Basic action event only contains textual kind and some basic common statistics if available

**Amber Provider**

Amber data for both InfiniBand and Ethernet MST devices in amBER format.

---

**Info**

MST device names can be found under /dev/mst/.

**Note**

/dev/mst should be accessible within DTS container.
The following config files are available:

amber_devices=DEV1,DEV2,DEV3  # Default:all, or set comma separated list of devices under /dev/mst
amber_update_interval_sec=30   # Sample rate for collection amber counters

**PPCC_ETH Provider**

Programmable congestion control counters are based on an algorithm defined by an end-user, although default algorithms are also available.

Counters are collected per MST device and algorithm parameters.

⚠️ **Info**

MST device names can be found under /dev/mst/.

⚠️ **Note**

/dev/mst should be accessible within the DTS container.

The counter list depends on the installed MFT version.

⚠️ **Note**

/usr/lib64/mft or /usr/lib/mft should be mounted to the DTS container to get the counter list according to the installed MFT version. If not mounted, the internal DTS version of the counters is used.
A comma-separated list of device names is required to enable this provider:

```
ppcc_eth_devices=mt41692_pciconf0,mt41692_pciconf0.1
```

The following algorithm parameters are available:

```
ppcc_algo_slot=1
ppcc_algo_param_index=0
local_port=1
pnat=0
lp_msb=0
```

**Info**

For more details, consult the official PPCC documentation.

**Note**

Some of the *algo_slots* are not implemented:

- If there are no counters to collect, the device is ignored
- If there are no devices to collect, the provider is disabled

**Fluent Aggregator**

`fluent_aggr` listens on a port for Fluent Bit Forward protocol input connections. Received data can be streamed via a Fluent Bit exporter.
The default port is 42442. This can be changed by updating the following option:

```
fluent-aggr-port=42442
```

**Prometheus Aggregator**

the `prometheus_aggr` polls data from a list of Prometheus endpoints.

Each endpoint is listed in the following format:

```
prometheus_aggr_endpoint.{N}={host_name},{host_port_url},{poll_interval_msec}
```

Where N starts from 0.

Aggregated data can be exported via a Prometheus Aggr Exporter endpoint.

**Network Interfaces**

`ifconfig` collects network interface data. To enable, set:

```
enable-provider=ifconfig
```

If the Prometheus endpoint is enabled, add the following configuration to cache every collected network interface and arrange the index according to their names:

```
prometheus-fset-indexes=name
```

Metrics are collected for each network interface as follows:
HCA Performance

`hcaperf` collects HCA performance data. Since it requires access to an RDMA device, it must use remote collection on the DPU. On the host, the user runs the container in privileged mode and RDMA device mount.

The counter list is device dependent.

**hcaperf DPU Configuration**

To enable `hcaperf` in remote collection mode, set:

```bash
enable-provider=grpc.hcaperf
```
hcaperf Host Configuration

To enable hcaperf in regular mode, set:

```
enable-provider=hcaperf

# specify HCAs to sample
hcaperf.mlx5_0=sample
hcaperf.mlx5_1=sample
```

**Note**

DPE server should be active before changing the `dts_config.ini` file. See section "Remote Collection" for details.

NVIDIA System Management Interface

The `nvidia-smi` provider collects GPU and GPU process information provided by the NVIDIA system management interface.

This provider is supported only on x86_64 hosts with installed GPUs. All GPU cards supported by `nvidia-smi` are supported by this provider.

The counter list is GPU dependent. Additionally, per-process information is collected for the first 20 (by default) `nvidia_smi_max_processes` processes.
Counters can be either collected as string data "as is" in nvidia-smi or converted to numbers when nvsmi_with_numeric_fields is set.

To enable nvidia-smi provider and change parameters, set:

```
enable-provider=nvidia-smi

# Optional parameters:
#nvidia_smi_max_processes=20
#nvsmi_with_numeric_fields=1
```

**NVIDIA Data Center GPU Manager**

The `dcgm` provider collects GPU information provided by the NVIDIA data center GPU manager (DCGM) API.

This provider is supported only on x86_64 hosts with installed GPUs, and requires running the `nv-hostengine` service (refer to [DCGM documentation](#) for details).

DCGM counters are split into several groups by context:

- **GPU** – basic GPU information (always)
- **COMMON** – common fields that can be collected from all devices
- **PROF** – profiling fields
- **ECC** – ECC errors
- **NVLINK / NVSWITCH / VGPU** – fields depending on the device type

To enable DCGM provider and counter groups, set:

```
enable-provider=dcgm

dcgm_events_enable_common_fields=1
#dcgm_events_enable_prof_fields=0
#dcgm_events_enable_ecc_fields=0
```
BlueField Performance

The `bfperf` provider collects calculated performance counters of BlueField Arm cores. It requires the executable `bfperf_pmc`, which is integrated in the DOCA BFB bundle of BlueField-3, as well as an active DPE.

To enable BlueField performance provider, set:

```plaintext
enable-provider=bfperf
```

**Note**

When running, the `bfperf` provider is expected to recurrently reset the counters of the `sysfs.hwmon` component. Consider disabling it if `bfperf` is enabled.

Ngauche

Ngauche is comprised of two providers which gather diagnostic data counters from network interface cards (NICs). These providers support the same counters (as defined in a YAML file), but they differ in usage and collection frequency:

- **Low frequency provider** is defined in `dts_config.ini` and is controlled by DTS collection loop

- **High frequency provider** is defined in `dts_high_freq_config.ini` and operates in a distinct flow for a limited duration
The `fwctl` and `mlx5_fwctl` drivers (supported on NVIDIA networking devices from BlueField-3 and ConnectX-7 and onward) are required for firmware interaction, and are part of MLNX_OFED driver. To load them, run:

```
modprobe -a fwctl mlx5_fwctl
```

Both providers get the counter set from a YAML file.

**Ngauge Low Frequency**

To enable the Ngauge low frequency provider, set:

```
enable-provider=ngauge_low_freq
```

To verify that the YAML file name matches the connected NIC's type:

```
gauge-yml-file=/config/ngauge_configs/all-single-port.yml
```

To configure the Ngauge timestamp collection type, set the following:

```
gauge-timestamp-collection-type=<method>
```

Where `<method>` can be one of the following:

- `no_counters` – Do not collect timestamp counters. Default.
- `start_and_end` – Collect sample start and end timestamps
- `per_counter` – Collect every counter collection timestamp

To configure the clock firmware should use when collecting time stamps, set the following:
Where \(<\text{clock}>\) can be one of the following:

- **RTC** - Real-time clock. Default.
- **RFC** - Free-running clock

### Ngauge High Frequency

This provider is designed to support higher sampling frequencies with sub-millisecond resolution. Due to the large scale of the collected data, this provider is aimed to run ad-hoc, for a limited time period, unlike the usual DTS providers which are configured with the DTS configuration file `/opt/mellanox/doca/services/telemetry/config/dts_config.ini`.

If the DTS standard flow constitutes an endless collect-export loop, then High Frequency Telemetry (HFT) is an additional external flow designed for the Ngauge high-frequency provider, based on the HFT configuration, located in `/opt/mellanox/doca/services/telemetry/config/dts_high_freq_config.ini`. This file defines the HFT session timing parameters, provider settings, and export settings. This means that an HFT session can export to different endpoints and/or protocols than those DTS used in the standard collection loop. The standard DTS configuration file references the HFT configuration file, enabling DTS to monitor the file's status. The HFT configuration file is also the trigger for the HFT session. That is, when the HFT configuration file is modified, the current HFT session is removed, and a new HFT session is configured (if defined). Removing the HFT configuration file stops pending sessions.

### Required HFT Parameters

This table provides the details of the required HFT parameters. Refer to section "HFT Configuration File Example" for more helpful tips.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-time</td>
<td>HFT session start time. If not used, the session starts immediately. UTC epoch timestamp (in microseconds). Syntax: HH:MM:SS / HH:MM</td>
</tr>
<tr>
<td>end-time</td>
<td>HFT session end time. Ignored if start-time is missing.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num-iterations</td>
<td>If not used, end-time is calculated using num-iterations. UTC epoch timestamp (in microseconds). Syntax: HH:MM:SS / HH:MM</td>
</tr>
<tr>
<td>sample-time-us</td>
<td>Time interval between iterations (in microseconds)</td>
</tr>
<tr>
<td>provider</td>
<td>Provider to use. Should be ngauge_high_freq.</td>
</tr>
<tr>
<td>file-write</td>
<td>Whether to write collected telemetry to files. If enabled, could potentially write several MB of data every second.</td>
</tr>
<tr>
<td>data-root</td>
<td>Root folder for file writing. Ignored if file-write=false.</td>
</tr>
<tr>
<td>provider.ngauge-num-samples</td>
<td>Number of samples to collect in one iteration. Affects the buffer used by the firmware for diagnostic data.</td>
</tr>
<tr>
<td>provider.ngauge-sample-period</td>
<td>Sample period between samples (in nanoseconds). This option specifies the sample interval per iteration, as the provider collects N samples during each iteration.</td>
</tr>
<tr>
<td>provider.ngauge-yml-file</td>
<td>The Ngauge counters YAML file to use</td>
</tr>
</tbody>
</table>

**Provider Compatibility**

Both low and high frequency providers can run concurrently. The low frequency provider samples at the DTS standard frequency (defined in `dts_config.ini`), and the high frequency provider samples counters based on the HFT configuration file (`dts_high_freq_config.ini`).

To allow both providers to run concurrently, verify that the counters, the timestamp collection type, and the timestamp collection source are identical. Otherwise, when the high frequency provider starts sampling, the low frequency provider hangs until the end of the HFT session.

**HFT Configuration File Example**

```yaml
## DTS configuration file for ad-hoc high frequency collection
```
## When modified, the file is parsed and applied.
## Note that the folders path is the container path, not the host path.

## Each section defines a collection. A file may have several sections, each one defines a high frequency collection.
## Section names must be unique and will be used as collection name by clx.

[htf-collection-session]

### Time between samples in microseconds
sample-time-us=100000

### Start time of high frequency collection. Can be in the format HH:MM:SS or HH:MM or as epoch timestamp in microseconds
### Note - in container, the time is in UTC
start-time=18:00:00

### End time of high frequency collection. Can be in the format HH:MM:SS or HH:MM or as epoch timestamp in microseconds
### Note - in container, the time is in UTC
datetime=18:01:00

### Alternatively, you can set the number of iterations. This and start_time field will determine the end time
datetime=300

### Data provider to use
datetime=ngauge_high_freq

### Write data to file system. Could potentially fill up the disk
datetime=false

### Root directory to store the data
datetime=/data

### Enable busy wait between iterations, for a more accurate sample time (default is false)
datetime=true

### Set prometheus endpoint to enable http endpoint
datetime=http://0.0.0.0:9112

### Set fluentbit config dir to enable fluentbit export
datetime=/config/fluent_bit_configs

### Set open telemetry receiver to enable open telemetry export
datetime=http://0.0.0.0:9502/v1/metrics
Ngauge YAML File

For Ngauge compatibility, the counter set is defined in a YAML file.

There are 4 existing YAML files within a DTS container (one per permutation of BlueField-3 and ConnectX-7 with dual or single ports). The path to the YAMLs folder is /opt/mellanox/doca/services/telemetry/config/ngauge_configs which is mounted to /config/ngauge_configs.

By default, YAML files include a counter set that is not device-specific. This implies that the same counter set is utilized across all devices by default.

It is possible to assign a specific device within a YAML file; however, this requires maintaining a separate copy of the YAML file for each device. To manage multiple devices, use the ngauge-yml-dir option to specify a directory for YAML files, where each .yml/.yaml file is utilized. This folder should be available to the container under /opt/mellanox/doca/services/telemetry/config.
The following list describes the expected entries in the YAML file:

- **counters** – sequence of counters to collect
  - **id** – counter data ID
  - **desc** – counter description (optional)
  - **unit** – name of unit to collect from (optional)
  - **name** – name of counter to use (optional). If not specified, the generated name is based on the counter description. Otherwise, it is based on the data ID.

- **device** – name of the mlx device to collect (optional). If not used, the provider requires a single file containing a list of counters, which it then applies to all available devices on the host.

**YAML File Example**

The following is the default `all-dual-port.yml` provided in DTS:

```yaml
counters:
  - id: 0x1020000100000000
    desc: RX bytes port 0
    unit: RX port
  - id: 0x1020000100000001
    desc: RX bytes port 1
    unit: RX port
  - id: 0x1020000300000000
    desc: TX packets port 0
    unit: TX port
  - id: 0x1020000300000001
    desc: TX packets port 1
    unit: TX port
  - id: 0x1140000100000000
    desc: TX bytes port 0
    unit: TX port
  - id: 0x1140000100000001
    desc: TX bytes port 1
    unit: TX port
  - id: 0x1140000300000000
    desc: TX packets port 0
```

DOCA Services
unit: TX port
- id: 0x1140000300000001
desc: TX packets port 1
unit: TX port
- id: 0x1100000100000000
desc: CNP sent packets port 0
unit: TX Transport
- id: 0x1100000100000001
desc: CNP sent packets port 1
unit: TX Transport
- id: 0x1080000400000000
desc: CNP handled packets port 0
unit: RX Transport
- id: 0x1080000400000001
desc: CNP handled packets port 1
unit: RX Transport
- id: 0x1080000500000000
desc: ECN RoCE packets port 0
unit: RX Transport
- id: 0x1080000500000001
desc: ECN RoCE packets port 1
unit: RX Transport
- id: 0x1160000b00000000
desc: PCIe link latency total read ns
unit: PCIe
cutoff_min: 1
cutoff_max: 2e6
- id: 0x1160000c00000000
desc: PCIe link latency total read packets
unit: PCIe
cutoff_min: 1
cutoff_max: 3000
- id: 0x1160000d00000000
desc: PCIe link latency max read ns
unit: PCIe
cutoff_min: 1
cutoff_max: 3000
- id: 0x1160000e00000000
desc: PCIe link latency min read ns
unit: PCIe
cutoff_min: 1
cutoff_max: 3000
Info

The NVIDIA Adapters Programmer's Reference Manual (PRM) "Diagnostic Data" section defines the rules for data IDs.

Counters

The following counters are available from the DTS default YAML files (and correspond the YAML file example):

cnp_handled_packets_port_0
cnp_handled_packets_port_1
cnp_sent_packets_port_0
cnp_sent_packets_port_1
ecn_roce_packets_port_0
ecn_roce_packets_port_1
pcie_link_latency_max_read_ns
pcie_link_latency_min_read_ns
pcie_link_latency_total_read_ns
pcie_link_latency_total_read_packets
rx_bytes_port_0
rx_bytes_port_1
rx_packets_port_0
rx_packets_port_1
tx_bytes_port_0
tx_bytes_port_1
tx_packets_port_0
tx_packets_port_1

Data Outputs
DTS can send the collected data to the following outputs:

- Data writer (saves binary data to disk)
- Fluent Bit (push-model streaming)
- Prometheus endpoint (keeps the most recent data to be pulled)

**Data Writer**

The data writer is disabled by default to save space on BlueField. Steps for activating data write during debug can be found under section Enabling Data Write.

The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example (apt install tree):

```
tree /opt/mellanox/doca/services/telemetry/data/
    /opt/mellanox/doca/services/telemetry/data/
    {year}
        {mmd}
        {hash}
            {source_id}
                {source_tag}{timestamp}.bin
            {another_source_id}
                {another_source_tag}{timestamp}.bin
        schema
            schema_{MD5_digest}.json
```

New binary files appear when the service starts or when binary file age/size restriction is reached. If no schema or no data folders are present, refer to the Troubleshooting section.

⚠️ **Note**
source_id is usually set to the machine hostname. source_tag is a line describing the collected counters, and it is often set as the provider's name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```bash
crictl exec -it <Container ID> /opt/mellanox/collectx/bin/clx_read -s /data/schema /data/path/to/datafile.bin
```

**Note**

The path to the data file must be an absolute path.

Example output:

```json
{
  "timestamp": 1634815738799728,
  "event_number": 0,
  "iter_num": 0,
  "string_number": 0,
  "example_string": "example_str_1"
}
{
  "timestamp": 1634815738799768,
  "event_number": 1,
  "iter_num": 0,
  "string_number": 1,
  "example_string": "example_str_2"
}
...
```
**Prometheus**

The Prometheus endpoint keeps the most recent data to be pulled by the Prometheus server and is enabled by default.

To check that data is available, run the following command on BlueField:

```bash
curl -s http://0.0.0.0:9100/metrics
```

The command dumps every counter in the following format:

```
counter_name {list of meta fields} counter_value timestamp
```

Additionally, endpoint supports JSON and CSV formats:

```bash
curl -s http://0.0.0.0:9100/json/metrics
curl -s http://0.0.0.0:9100/csv/metrics
```

⚠️ **Note**

The default port for Prometheus can be changed in `dts_config.ini`.

**Configuration Details**

Prometheus is configured as a part of `dts_config.ini`.

By default, the Prometheus HTTP endpoint is set to port 9100. Comment this line out to disable Prometheus export.
Prometheus can use the data field as an index to keep several data records with different index values. Index fields are added to Prometheus labels.

The default fset index is `device_name`. It allows Prometheus to keep ethtool data up for both the `p0` and `p1` devices.

If fset index is not set, the data from `p1` overwrites `p0`'s data.

For quick name filtering, the Prometheus exporter supports being provided with a comma-separated list of counter names to be ignored:

```
#prometheus-ignore-names=counter_name1,counter_name_2
```

For quick filtering of data by tag, the Prometheus exporter supports being provided with a comma-separated list of data source tags to be ignored.

```
prometheus-ignore-tags=Fi_metrics
```

Users should add tags for all streaming data since the Prometheus exporter cannot be used for streaming. By default, `FI_metrics` are disabled.
**Prometheus Aggregator Exporter**

Prometheus aggregator exporter is an endpoint that keeps the latest aggregated data using `prometheus_aggr`.

This exporter labels data according to its source.

To enable this provider, users must set 2 parameters in `dts_config.ini`:

```
prometheus-aggr-exporter-host=0.0.0.0
prometheus-aggr-exporter-port=33333
```

**Fluent Bit**

Fluent Bit allows streaming to multiple destinations. Destinations are configured in `.exp` files that are documented in-place and can be found under:

```
/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs
```

Fluent Bit allows exporting data via "Forward" protocol which connects to the Fluent Bit/FluentD instance on customer side.

Export can be enabled manually:

1. Uncomment the line with `fluent_bit_configs=...` in `dts_config.ini`.
2. Set `enable=1` in required `.exp` files for the desired plugins.
3. Additional configurations can be set according to instructions in the `.exp` file if needed.
4. Restart the DTS.
5. Set up receiving instance of Fluent Bit/FluentD if needed.
6. See the data on the receiving side.
Export file destinations are set by configuring .exp files or creating new ones. It is recommended to start by going over documented example files. Documented examples exist for the following supported plugins:

- forward
- file
- stdout
- kafka
- es (elastic search)
- influx

**Note**

All .exp files are disabled by default if not configured by initContainer entry point through .yaml file.

**Note**

To forward the data to several destinations, create several forward_{num}.exp files. Each of these files must have their own destination host and port.

**Export File Configuration Details**

Each export destination has the following fields:

- **name** – configuration name
- **plugin_name** – Fluent Bit plugin name
• enable – 1 or 0 values to enable/disable this destination

• host – the host for Fluent Bit plugin

• port – port for Fluent Bit plugin

• msgpack_data_layout – the msgpacked data format. Default is flb_std. The other option is custom. See section Msgpack Data Layout for details.

• plugin_key=val – key-value pairs of Fluent Bit plugin parameter (optional)

• counterset/fieldset – file paths (optional). See details in section Cset/Fset Filtering.

• source_tag=source_tag1,source_tag2 – comma-separated list of data page source tags for filtering. The rest tags are filtered out during export. Event tags are event provider names. All counters can be enabled/disabled only simultaneously with a counters keyword.

**Note**

Use # to comment a configuration line.

**Msgpack Data Layout**

Data layout can be configured using .exp files by setting msgpack_data_layout=layout. There are two available layouts: Standard and Custom.

The standard flb_std data layout is an array of 2 fields:

• timestamp double value

• a plain dictionary (key-value pairs)

The standard layout is appropriate for all Fluent Bit plugins. For example:
The custom data layout is a dictionary of meta-fields and counter fields. Values are placed into a separate plain dictionary. Custom data format can be dumped with `stdout_raw` output plugin of Fluent-Bit installed or can be forwarded with `forward` output plugin.

Counters example:

```
{"timestamp":timestamp_val, "type":"counters", "source":"source_val", "values":{"key_1":val_1, "key_2":val_2,...}}
```

Events example:

```
{"timestamp":timestamp_val, "type":"events", "type_name":"type_name_val", "source":"source_val", "values":{"key_1":val_1, "key_2":val_2,...}}
```

**Cset/Fset Filtering**

Each export file can optionally use one `cset` and one `fset` file to filter UFM telemetry counters and events data.

- `cset` contains tokens per line to filter data with `"type"="counters"`.

- `fset` contains several blocks started with the header line `[event_type_name]` and tokens under that header. An Fset file is used to filter data with `"type"="events"`.

**Note**
Event type names could be prefixed to apply the same tokens to all fitting types. For example, to filter all ethtool events, use [ethtool_event_*].

If several tokens must be matched simultaneously, use `<tok1>+<tok2>+<tok3>`. Exclusive tokens are available as well. For example, the line `<tok1>+<tok2>-<tok3>-<tok4>` filters names that match both `tok1` and `tok2` and do not match `tok3` or `tok4`.

The following are the details of writing `cset` files:

```
# Put tokens on separate lines
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port"
# port+xmit # match names that contain both tokens "port" and "xmit"
# port-support # match names that contain the token "port" and do not match the "-" token "support"
#
# Tip: To disable counter export put a single token line that fits nothing
```

The following are the details of writing `fset` files:

```
# Put your events here
# Usage:
#
# [type_name_1]
# tokens
# [type_name_2]
# tokens
# [type_name_3]
# tokens
# ...
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port"
# port+xmit # match names that contain both tokens "port" and "xmit"
# port-support # match names that contain the token "port" and do not match the "-" token "support"
```
NetFlow Exporter

NetFlow exporter must be used when data is collected as NetFlow packets from the telemetry client applications. In this case, DOCA Telemetry Exporter NetFlow API sends NetFlow data packages to DTS via IPC. DTS uses NetFlow exporter to send data to the NetFlow collector (3rd party service).

To enable NetFlow exporter, set netflow-collector-ip and netflow-collector-port in dts_config.ini. netflow-collector-ip could be set either to IP or an address.

For additional information, refer to the dts_config.ini file.

DOCA Privileged Executer

DOCA Privileged Executer (DPE) is a daemon that allows specific DOCA services (DTS included) to access BlueField information that is otherwise inaccessible from a container due to technology limitations or permission granularity issues.

When enabled, DPE enriches the information collected by DTS. However, DTS can still be used if DPE is disabled (default).
**DPE Usage**

DPE is controlled by systemd, and can be used as follows:

- **To check DPE status:**
  
  ```
  sudo systemctl status dpe
  ```

- **To start DPE:**
  
  ```
  sudo systemctl start dpe
  ```

- **To stop DPE:**
  
  ```
  sudo systemctl stop dpe
  ```

DPE logs can be found in `/var/log/doca/telemetry/dpe.log`.

**DPE Configuration File**

DPE can be configured by the user. This section covers the syntax and implications of its configuration file.

**Note**

The DPU telemetry collected by DTS does not require for this configuration file to be used.
The DPE configuration file allows users to define the set of commands that DPE should support. This may be done by passing the `-f` option in the following line of `/etc/systemd/system/dpe.service`:

```
ExecStart=/opt/mellanox/doca/services/telemetry/dpe/bin/dpeserver -vv
```

To use the configuration file:

```
ExecStart=/opt/mellanox/doca/services/telemetry/dpe/bin/dpeserver -vvv -f /path/to/dpe_config.ini
```

The configuration file supports the following sections:

- **[server]** - list of key=value lines for general server configuration. Allowed keys: `socket`.
- **[commands]** - list of bash command lines that are not using custom RegEx
- **[commands_regex]** - list of bash command lines that are using custom RegEx
- **[regex Macros]** - custom RegEx definitions used in the `commands_regex` section

Consider the following example configuration file:

```
[server]
socket=/tmp/dpe.sock

[commands]
hostname
cat /etc/os-release

[commands_regex]
crictl inspect $HEXA  # resolved as "crictl inspect [a-f0-9]+"
lspci $BDF  # resolved as "lspci ([0-9a-f]{4}:|)[0-9a-f]{2}:0-f]{2}:0-9a-f]"

[regex Macros]
HEXA=[a-f0-9]+  
BDF=[0-9a-f]{4}:|0-9a-f]{2}:0-9a-f]"
```
Deploying with Grafana Monitoring

This chapter provides an overview and deployment configuration of DOCA Telemetry Service with Grafana.

Grafana Deployment Prerequisites

- BlueField DPU running DOCA Telemetry Service.
- Optional remote server to host Grafana and Prometheus.
- Prometheus installed on the host machine. Please refer to the Prometheus website for more information.
• Grafana installed on the host machine. Please refer to Grafana Labs website for more information.

Grafana Deployment Configuration

DTS Configuration (DPU Side)

Configuring DTS to export the sysfs counter using the Prometheus plugin:

1. Make sure the sysfs counter is enabled.

   ```bash
   vim /opt/mellanox/doca/services/telemetry/config/dts_config.ini
   enable-provider=sysfs
   ```

2. Enable Prometheus exporter by setting the prometheus address and port.

   ```bash
   vim /opt/mellanox/doca/services/telemetry/config/dts_config.ini
   prometheus=http://0.0.0.0:9100
   ```

Note: Sysfs is used as an example, other counters are available.
Prometheus Configuration (Remote Server)

Please download Prometheus for your platform.

Prometheus is configured via command-line flags and a configuration file, prometheus.yml.

1. Open the prometheus.yml file and configure the DPU as the endpoint target.

```bash
vim prometheus.yml
# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
  - targets: ["<dpu-ip>:<prometheus-port>"]
```

Where:

- `<dpu-ip>` is the DPU IP address. Prometheus reaches to this IP to pull data.
- `<prometheus-port>` the exporter port that set in DTS configuration.

2. Run Prometheus server:
Please download and install Grafana for your platform.

1. Setup Grafana. Please refer to [Install Grafana](#) guide in Grafana documentation.


   ```
   ./prometheus --config.file="prometheus.yml"
   ```

   **Tip**
   Prometheus services are available as Docker images. Please refer to [Using Docker](#) in Prometheus' Installation guide.

**Grafana Configuration (Remote Server)**

3. Add Prometheus as data source by navigating to Settings Data sources Add data source Prometheus.

   **Note**
   Port 3000 is the default port number set by Grafana. This can be changed if needed. The default credentials are admin/admin.
4. Configure the Prometheus data source. Under the HTTP section, set the Prometheus server address.

**Note**

The Prometheus server's default listen port is 9090. Prometheus and Grafana are both running on the same server, thus the address is localhost.

5. Save and test.
Exploring Telemetry Data

Go to the Explore page on the left-hand side, and choose a Prometheus provider.

Choose a metric to display and specify a label. The label can be used to filter out data based on the source and HCA devices.

Graph display after selecting a metric and specifying a label to filter by:

Troubleshooting

On top of the Troubleshooting section in the NVIDIA DOCA Container Deployment Guide, here are additional troubleshooting tips for DTS:

- For general troubleshooting, refer to the NVIDIA DOCA Troubleshooting Guide.
- If the pod's state fails to be marked as "Ready", refer to `/var/log/syslog`.
- Check if the service is configured to write data to the disk as this may cause the system to run out of disk space.
If a PIC bus error occurs, configure the following files inside the container:

```
crictl exec -it <container-id> /bin/bash
# Add to /config/clx.env the following line:
"
export UCX_TLS=tcp
"
```
NVIDIA DOCA UROM Service Guide

This guide provides instructions on how to use the DOCA UROM Service on top of the NVIDIA® BlueField® networking platform.

Introduction

The DOCA UROM service provides a framework for offloading significant portions of HPC software stack directly from the host and to the BlueField device.

Using a daemon, the service handles the discovery of resources, the coordination between the host and BlueField, and the spawning, management, and teardown of the BlueField workers themselves.
The first step in initiating an offload request involves the UROM host application establishing a connection with the UROM service. Upon receiving the plugin discovery command, the UROM service responds by providing the application with a list of plugins available on the BlueField. The application then attaches the plugin IDs that correspond to the desired workers to their network identifiers. Finally, the service triggers UROM worker plugin instances on the BlueField to execute the parallel computing tasks. Within the service's Kubernetes pod, workers are spawned by the daemon in response to these offload requests. Each computation can utilize either a single library or multiple computational libraries.

Requirements

Before deploying the UROM service container, ensure that the following prerequisites are satisfied:

- Allocate huge pages as needed by DOCA (this requires root privileges):
Service Deployment

For information about the deployment of DOCA containers on top of the BlueField, refer to the [NVIDIA BlueField Container Deployment Guide](#).

Service-specific configuration steps and deployment instructions can be found under the service's [container page](#).

Description

Plugin Discovery and Reporting

When the application initiates a connection request to the DOCA UROM Service, the daemon reads the `UROM_PLUGIN_PATH` environment variable. This variable stores directory paths to `.so` files for the plugins with multiple paths separated by semicolons. The daemon scans these paths sequentially and tries loading each `.so` file. Once the daemon finishes the scan, it reports the available BlueField plugins to the host application.

The host application gets the list of available plugins as a list of `doca_urom_service_plugin_info` structures:

```c
struct doca_urom_service_plugin_info {
    uint64_t id;       // Unique ID to send commands to the plugin
    uint64_t version;  // Plugin version
    char plugin_name[DOCA_UROM_PLUGIN_NAME_MAX_LEN];  // .so filename
};
```
The UROM daemon is responsible for generating unique identifiers for the plugins, which are necessary to enable the worker to distinguish between different plugin tasks.

**Loading Plugin in Worker**

During the spawning of UROM workers by the UROM daemon, the daemon attaches a list of desired plugins in the worker command line. Each plugin is passed in a format of so_path:id.

As part of worker bootstrapping, the flow iterates all .so files and tries to load them by using `dlopen` system call and look for `urom_plugin_get_iface()` symbol to get the plugin operations interface.

**Yaml File**

The .yaml file downloaded from NGC can be easily edited according to users' needs:

```
env:
  # Service-Specific command line arguments
  - name: SERVICE_ARGS
    value: "-l 60 -m 4096"
  - name: UROM_PLUGIN_PATH
    value: "/opt/mellanox/doca/samples/doca_urom/plugins/worker_sandbox;/opt/mellanox/doca/samples/doca_u"
```

- The SERVICE_ARGS are the runtime arguments received by the service:
  - `-l, --log-level <value>` – sets the (numeric) log level for the program `<10=DISABLE, 20=Critical, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>`
  - `--sdk-log-level` – sets the SDK (numeric) log level for the program `<10=DISABLE, 20=Critical, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>`
  - `-m, --max-msg-size` – specify UROM communication channel maximum message size
- The `UROM_PLUGIN_PATH` is an env variable that stores directory paths to `.so` files for the plugins.

For each plugin on the BlueField, it is necessary to add a volume mount inside the service container. For example:

```yaml
volumes:
  - name: urom-sandbox-plugin
    hostPath:
      path: /opt/mellanox/doca/samples/doca_urom/plugins/worker_sandbox
      type: DirectoryOrCreate

... volumeMounts:
  - mountPath: /opt/mellanox/doca/samples/doca_urom/plugins/worker_sandbox
    name: urom-sandbox-plugin
```

**Troubleshooting**

When troubleshooting a container deployment issues, it is highly recommended to follow the deployment steps and tips found in the "Review Container Deployment" section of the NVIDIA BlueField Container Deployment Guide.

One could also check the `/var/log/doca/urom` log files for more details about the running cycles of service components (daemon and workers).

The log file name for workers is `urom_worker_<pid>_dev.log` and for the daemon it is `urom_daemon_dev.log`.

**Pod is Marked as "Ready" and No Container is Listed Error**

When deploying the container, the pod's STATE is marked as `Ready` and an image is listed, however, no container can be seen running:

```
$ sudo crictl pods
```
In most cases, the container did start but immediately exited. This could be checked using the following command:

```
$ sudo crictl images
```

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8s.gcr.io/pause</td>
<td>3.2</td>
<td>2a060e2e7101d</td>
<td>487kB</td>
</tr>
<tr>
<td>nvcr.io/nvidia/doca/doca_urom</td>
<td>1.0.0-doca2.7.0</td>
<td>2af1e539eb7ab</td>
<td>86.8MB</td>
</tr>
</tbody>
</table>

Should the container fail (i.e., reporting a state of Exited), it is recommended to examine the UROM's main log at `/var/log/doca/urom/urom_daemon_dev.log`.

In addition, for a short period of time after termination, the container logs could also be viewed using the container's ID:

```
$ sudo crictl logs 556bb78281e1d
```

...
Pod is Not Listed

Error

When placing the container's YAML file in the Kubelet's input folder, the service pod is not listed in the list of pods:

<table>
<thead>
<tr>
<th>$ sudo crictl pods</th>
</tr>
</thead>
<tbody>
<tr>
<td>POD ID</td>
</tr>
<tr>
<td>ATTEMPT</td>
</tr>
</tbody>
</table>

Solution

In most cases, the pod has not started because of the absence of the requested hugepages. This can be verified using the following command:

<table>
<thead>
<tr>
<th>$ sudo journalctl -u kubelet -e . .</th>
</tr>
</thead>
</table>
| Oct 04 12:19 <my-dpu> kubelet[2442376]: I1004 12:19.905064 2442376 predicate.go:103] "Failed to admit pod, unexpected error while attempting to recover from admission failure" pod="default/doca-urom-service-<my-dpu>" err="preemption: error finding a set of pods to preempt: no set of running pods found to reclaim resources: [(res: hugepages-2Mi, q: 104563999874), ]"

© Copyright 2024, NVIDIA. PDF Generated on 08/15/2024