NVIDIA Base Command Platform

User Guide
# Table of Contents

Chapter 1. Introduction to NVIDIA Base Command Platform ............................................. 1  
1.1. NVIDIA Base Command Platform Terms and Concepts ............................................... 1

Chapter 2. Onboarding and Signing Up ............................................................................... 4  
2.1. Inviting Users .................................................................................................................... 4
2.2. Joining an NGC Org or Team .............................................................................................. 7  
2.2.1. Joining an NGC Org or Team Using Single Sign-on .................................................... 8
2.2.2. Joining an Org or Team with a New NVIDIA Account .................................................. 9
2.2.3. Joining an Org or Team with an Existing NVIDIA Account .......................................... 15

Chapter 3. Signing in to Your Account ............................................................................... 18

Chapter 4. Introduction to the NGC CLI ............................................................................. 20  
4.1. About NGC CLI for NVIDIA Base Command Platform ................................................... 20
4.2. Generating Your NGC API Key .......................................................................................... 21
4.3. Installing NGC CLI ............................................................................................................. 22
4.4. Getting Help Using NGC CLI ............................................................................................ 22  
4.4.1. Getting Help from the Command Line ........................................................................ 22
4.4.2. Viewing NGC CLI Documentation Online .................................................................. 23
4.5. Configuring the CLI for your Use .................................................................................... 23  
4.5.1. Configuring the Output Format ................................................................................... 23
4.6. Running the Diagnostics ................................................................................................... 24
4.7. Specifying List Columns ..................................................................................................... 25
4.8. Other Useful Command Options ....................................................................................... 26

Chapter 5. Using NGC APIs ............................................................................................... 27  
5.1. Example of Getting Basic Job Information ...................................................................... 27  
5.1.1. Using Get Request ........................................................................................................ 27
5.1.2. Code Example of Getting a Token ............................................................................... 27
5.1.3. Code Example of Getting Job Information .................................................................. 28
5.1.4. Code Example of Getting Telemetry Data .................................................................. 31
5.2. List of API Endpoints ......................................................................................................... 32

Chapter 6. NGC Catalog .................................................................................................... 34  
6.1. Accessing NGC Catalog .................................................................................................... 34
6.2. Viewing Detailed Application Information ....................................................................... 35
6.3. Using the CLI ..................................................................................................................... 35

Chapter 7. NGC Private Registry ....................................................................................... 37  
7.1. Accessing the NGC Private Registry ............................................................................... 37
Chapter 10. Jobs and GPU Instances..................................................................... 64
  10.1. Identifying Available ACE and Instances............................................. 64
  10.2. Running a Simple Job........................................................................... 64
  10.3. Running JupyterLab in a Job................................................................. 65
    10.3.1. Example of Running JupyterLab in a Job....................................... 65
    10.3.2. Connecting to JupyterLab............................................................... 65
  10.4. Cloning an Existing Job........................................................................ 66
  10.5. Launching a Job from a Template File................................................ 67
  10.6. Launching a Job Using a JSON File...................................................... 68
    10.6.1. Generating the JSON Using the Web UI....................................... 68
    10.6.2. Generating the JSON Using the CLI............................................ 68
    10.6.3. Overriding Fields in a JSON File................................................ 69
  10.7. Exec into a Running Job using CLI....................................................... 70
  10.8. Attaching to the Console of a Running Job........................................... 71
  10.9. Managing Jobs...................................................................................... 71
    10.9.4. Terminating Jobs.......................................................................... 73
    10.9.5. Deleting Results........................................................................... 74

Chapter 11. Telemetry...................................................................................... 75
  11.1. Viewing Telemetry Information from the NGC Web UI..................... 76
  11.2. Telemetry for Multinode Jobs............................................................... 82

Chapter 12. Advanced Base Command Concepts.......................................... 84
  12.1. Multi-node Jobs................................................................................... 84
    12.1.1. Defining Multi-node Jobs.............................................................. 84
    12.1.2. Understanding the --replicas argument...................................... 85
    12.1.3. Starting a Multi-node Job from the NGC Web UI....................... 85
    12.1.4. Viewing Multi-node Job Results from the NGC Web UI.............. 87
    12.1.5. Launching Multi-node Jobs Using the NGC CLI......................... 87
    12.1.6. Launching Multi-node Jobs with bcprun.................................... 89
  12.2. Job ENTRYPOINT............................................................................... 91
    12.2.1. Example Using Container ENTRYPOINT.................................... 92
    12.2.2. Example Using CLI ENTRYPOINT.............................................. 93

Chapter 13. Tutorials..................................................................................... 94
  13.1. Launching a Job from Existing Templates.......................................... 94
  13.3. Launching a Multi Node Interactive Job with JupyterLab.................. 96
  13.4. Getting Started with Tensorboard..................................................... 96
  13.5. NCCL Tests....................................................................................... 98
  13.6. StyleGAN Singleton Workload........................................................... 98
  13.7. StyleGAN MultiNode Workload......................................................... 99
13.8. Building a Dataset from S3 Cloud Storage................................................................. 100
  13.8.1. Running a Job........................................................................................................ 100
  13.8.2. Creating a Dataset using AWS CLI....................................................................... 101
  13.8.3. Creating a Dataset using AWS Boto3................................................................. 102
  13.8.4. Downloading a File.............................................................................................. 102
  13.8.5. Downloading a Folder......................................................................................... 102
13.9. Using Data Loader for Cloud Storage......................................................................... 103
  13.9.1. Running and Opening JupyterLab......................................................................... 103
  13.9.2. Utilizing the Cloud Data Loader for Training....................................................... 104

Chapter 14. Using NVIDIA Base Command Platform with Weights & Biases.............. 105
  14.1. Introduction................................................................................................................ 105
  14.2. Setup.......................................................................................................................... 105
    14.2.1. Base Command Platform Setup........................................................................... 105
    14.2.2. Weights and Biases Setup..................................................................................... 106
    14.2.3. Storing W&B Keys in Base Command Platform................................................... 106
      14.2.3.1. Option 1 | Using a Jupyter Notebook.............................................................. 106
      14.2.3.2. Option 2 | Using a Script (via curl Command)............................................... 107
  14.3. Using W&B with a JupyterLab Workload................................................................. 108
    14.3.1. Create a Jupyter Notebook, Including W&B Keys for Experiment Tracking........ 108
    14.3.2. Running a W&B Experiment in Batch Mode....................................................... 110
  14.4. Using Logged Dataset and Model Artifacts............................................................. 111
  14.5. Best Practices for Running Multiple Jobs Within the Same Project...................... 113
  14.6. Supplemental Reading............................................................................................... 114

Chapter 15. Deregistering............................................................................................... 115
Chapter 1. Introduction to NVIDIA Base Command Platform

NVIDIA Base Command is a comprehensive platform for businesses, their data scientists and IT teams, offered in a ready-to-use cloud-hosted solution that manages the end to end lifecycle of AI development, AI workflows, and resource management.

NVIDIA Base Command Platform provides

- A set of cloud-hosted tools that lets data scientists access the AI infrastructure without interfering with each other.
- A comprehensive cloud-based UI, as well as a complete command line API, to efficiently execute AI workloads with right-sized resources ranging from a single GPU to a multi-node cluster with dataset management, providing quick delivery of production-ready models and applications.
- Built-in telemetry that can be used to validate deep learning techniques, workload settings, and resource allocations as part of a constant improvement process.
- Reporting and showback capabilities for business leaders who want to measure AI projects against business goals, as well as team managers who need to set project priorities and plan for a successful future by correctly forecasting compute capacity needs.

1.1. NVIDIA Base Command Platform Terms and Concepts

The following are a description of common NVIDIA Base Command Platform terms used in this document.

Table 1. NVIDIA Base Command Platform Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerated Computing Environment (ACE)</td>
<td>An ACE is a cluster or an availability zone. Each ACE has separate storage, compute, and networking.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NGC Catalog</td>
<td>NGC Catalog is a curated set of GPU-optimized software maintained by NVIDIA and accessible to the general public. It consists of containers, pre-trained models, Helm charts for Kubernetes deployments and industry specific AI toolkits with software development kits (SDKs).</td>
</tr>
<tr>
<td>Container Images</td>
<td>All applications running in NGC are containerized as Docker containers and execute in our Runtime environment. Containers are stored in the NGC Container Registry nvcr.io, accessible from both the CLI and the Web UI.</td>
</tr>
<tr>
<td>Dataset</td>
<td>Datasets are the data inputs to a job, mounted as read-only to the location specified in the job. They can contain data or code. Datasets are covered in detail in the Datasets section.</td>
</tr>
<tr>
<td>Data Results</td>
<td>Result is a read-write mount specified by the job and captured by the system. All data written to the result is available once the job is finished, along with contents of stdout and stderr.</td>
</tr>
<tr>
<td>Instance</td>
<td>The instance determines the amount of CPU cores, RAM, and type and number of GPUs available to the Job. Instance types from 1 to 8 GPUs are available depending on the ACE.</td>
</tr>
<tr>
<td>Job</td>
<td>A Job is the fundamental unit of computation - a container running an NVIDIA Base Command Platform instance in an ACE. Job is defined by the set of attributes specified at submission.</td>
</tr>
<tr>
<td>Job Definition</td>
<td>The attributes that define a job.</td>
</tr>
<tr>
<td>Job Command</td>
<td>Each Job can specify a command to run inside the container. The command can be as simple or as complex as needed, as long as quotes are properly escaped.</td>
</tr>
<tr>
<td>Jobs – Multinode</td>
<td>A job that is run on multiple nodes.</td>
</tr>
<tr>
<td>Models</td>
<td>NGC offers a collection of State of the Art pre-trained deep learning models that can be easily used out of the box, re-trained or fine-tuned.</td>
</tr>
<tr>
<td>Org</td>
<td>The enterprise organization with its own registry space. Users are assigned (or belong) to an org.</td>
</tr>
<tr>
<td>Team</td>
<td>A sub-unit within an organization with its own registry space. Only members of the same team have access to that team’s registry space.</td>
</tr>
<tr>
<td>Users</td>
<td>Anyone with a Base Command account. Users are assigned to an org.</td>
</tr>
<tr>
<td>Private Registry</td>
<td>The NGC private registry provides you with a secure space to store and share custom</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Term</td>
<td>containers, models, resources, and helm charts within your enterprise.</td>
</tr>
<tr>
<td>Quota</td>
<td>Every user is assigned a default GPU and Storage quota. GPU quota defines the maximum number of concurrent GPUs in use by a user account. Each user is allocated a default initial storage quota. All your storage assets (datasets, results, and workspaces) are counted towards your storage quota.</td>
</tr>
<tr>
<td>Resources</td>
<td>NGC offers step-by-step instructions and scripts for creating deep learning models that can be shared within teams or the org.</td>
</tr>
<tr>
<td>Telemetry</td>
<td>NVIDIA Base Command Platform provides time series metric data collected from various system components such as GPU, Tensor Cores, CPU, Memory, and I/O.</td>
</tr>
<tr>
<td>Workspaces</td>
<td>Workspaces are shareable read-write persistent storage mountable in jobs for concurrent use. Workspaces can be mounted to a job in read-only mode too, making those ideal for configuration/code/input use cases in the comfort of knowing that the job will not corrupt/modify any of this data. Mounting workspaces in read-write mode (which is the default) in a job works well for use as a checkpoint folder.</td>
</tr>
</tbody>
</table>
Chapter 2. Onboarding and Signing Up

This chapter walks you through the process of setting up your NVIDIA Base Command Platform Account. In this chapter you will learn about signing up, signing in, installing and configuring CLI, and selecting and switching your team context.

2.1. Inviting Users

This section is for org or team administrators (with User Admin role) and describes the process for inviting (adding) users to NVIDIA Base Command Platform.

As the organization administrator, you must create user accounts to allow others to use the NVIDIA Base Command Platform within the organization.

1. Log on to the NGC web UI and and select the NGC Org associated with Base Command Platform.
2. Click Organization > Users from the left side menu.

This capability is available only to User Admins.
3. Click the “+” icon at the lower right corner, then select the Invite New User icon.
4. Select the **Personal Info** tab and then enter the display name and email where indicated.

5. Click **Next** or select the **Membership** tab and then select one or more user roles.
The following are brief descriptions of the user roles:

### Table 2.

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Command Admin</td>
<td>Admin persona with the capabilities to manage all artifacts available in Base Command. The capabilities of the Admin role include resource allocation and access management.</td>
</tr>
<tr>
<td>Base Command Viewer</td>
<td>Admin persona with the read-only access to jobs, workspaces, datasets, and results within the user’s org or team.</td>
</tr>
<tr>
<td>Registry Admin</td>
<td>Registry Admin persona for managing NGC Private Registry artifacts and with the capability for Registry User Management. The capabilities of the Registry Admin role include the capabilities of all Registry roles.</td>
</tr>
<tr>
<td>Registry Only</td>
<td>Registry User persona with capabilities to only consume the Private Registry artifacts.</td>
</tr>
<tr>
<td>Registry User</td>
<td>Registry User persona with the capabilities to publish and consume the Private Registry artifacts.</td>
</tr>
<tr>
<td>User Admin</td>
<td>User Admin persona with the capabilities to only manage users.</td>
</tr>
</tbody>
</table>

See also the section Assigning Roles for additional information.

6. (Optional) Select a team for the user and select one or more roles for the user to have within that team.

7. Click Assign.
If you want to assign the user to more than one team, select another team and role, then click **Assign**. The following example screenshot shows a user assigned to two teams:

![Assign an Organization and Team](image)

8. Click **Confirm** to complete the process.
   
   An invitation email is automatically sent to the user.

### 2.2. Joining an NGC Org or Team

Before using NVIDIA Base Command Platform, you must have an NVIDIA Base Command Platform account created by your organization administrator. You need an email address to set up an account. The process for activating an account depends on whether your email domain is mapped to your organization’s single sign-on (SSO). Choose one of the following processes depending on your situation for activating your NVIDIA Base Command Platform account.

- **Joining an NGC Org or Team Using Single Sign-on**
- **Joining an Org or Team with a New NVIDIA Account**
- **Joining an Org or Team with an Existing NVIDIA Account**
2.2.1. Joining an NGC Org or Team Using Single Sign-on

This section describes activating an account where the domain of your email address is mapped to an organization’s single sign-on.

After NVIDIA or your organization administrator adds you to a new org or team within the organization, you will receive a welcome email that invites you to continue the activation and login process.

1. Click the link in the email to open your organization’s single sign-on page.
2. Sign in using your single sign-on credentials.
   The Set Your Organization screen appears.
This screen appears any time you log in.

3. Select which organization and team you want to log in under and then click Sign In.

You can always change the org or team to any other org or team of which you are a member after you log in.

The NGC web UI opens to the Base Command Platform dashboard.

2.2.2. Joining an Org or Team with a New NVIDIA Account

This section describes activating a new account where the domain of your email address is not mapped to an organization’s single sign-on.

After NVIDIA or your organization administrator sets up your NVIDIA Base Command Platform account, you will receive a welcome email that invites you to continue the activation and login process.
1. Click the Sign In link to open the sign in dialog in your browser.

2. Click Create account to open the Create an Account screen.
3. Fill out your information, create a password, agree to the Terms and Conditions and then click **Create Account**.

You are notified to verify your email.

The verification email is sent.
4. Open the email and then click **Verify Email Address**.
5. Select your options for using recommended settings and receiving developer news and announcements, and then click **Submit**.

6. Agree to the NVIDIA Account Terms of Use and select desired options and then click **Continue**.

7. Click **Accept** at the NVIDIA GPU Cloud Terms of Use screen.
8. Click **Accept** at the **NVIDIA GPU Cloud Terms of Use** screen.

The Set Your Organization screen appears.
This screen appears any time you log in.

9. Select which organization and team you want to log in under and then click **Sign In**.
You can always change the org or team to any other org or team of which you are a member after you log in.
The NGC web UI opens to the **Base Command Platform** dashboard.

### 2.2.3. Joining an Org or Team with an Existing NVIDIA Account

This section describes activating an account where the domain of your email address is not mapped to an organization’s single sign-on (SSO).

After NVIDIA or your organization administrator adds you to a new org or team within the organization, you will receive a welcome email that invites you to continue the activation and login process.

![Welcome Email](image)

1. Click the **Sign In** link to open the sign in dialog in your browser.
2. Enter your password and then click **Log In**.
   The Set Your Organization screen appears.
This screen appears any time you log in.

3. Select which organization and team you want to log in under and then click **Sign In**.

You can always change the org or team to any other org or team of which you are a member after you log in.

The NGC web UI opens to the **Base Command Platform** dashboard.
Chapter 3. Signing in to Your Account

During the initial account setup, you are signed into your NVIDIA Base Command Platform account on the NGC web site. This section describes the sign in process that occurs at a later time. It also describes the web UI sections of Base Command Platform at a high level, including the UI areas for accessing available artifacts and actions available to various user roles.

1. Open https://ngc.nvidia.com and click Continue by one of the sign-on choices, depending on your account.
   - **NVIDIA Account**: Select this option if single sign-on (SSO) is not available.
   - **Single Sign-on (SSO)**: Select this option to use your organization’s SSO. You may need to verify with your organization or Base Command Platform administrator whether SSO is enabled.

2. Continue to sign in using your organization’s single sign-on.
3. Set the organization you wish to sign in under, then click Sign in.

You can always change the org or team to any other org or team of which you are a member. The following image and table describe the main drop-down features of the web site, including the controls for changing the org or team.
Table 3. NGC Web UI Sections

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>CATALOG</strong>: Click this menu option to access a curated set of GPU-optimized software. It consists of containers, pre-trained models, Helm charts for Kubernetes deployments, and industry-specific AI toolkits with software development kits (SDKs) that are periodically released by NVIDIA and are read-only for a Base Command user.</td>
</tr>
<tr>
<td>2</td>
<td><strong>PRIVATE REGISTRY</strong>: Click this drop-down to access the secure space to store and share custom containers, models, resources, and helm charts within your enterprise.</td>
</tr>
<tr>
<td>3</td>
<td><strong>BASE COMMAND</strong>: Click this drop-down to access controls for creating and running Base Command Platform jobs.</td>
</tr>
<tr>
<td>4</td>
<td><strong>ORGANIZATION</strong>: (User Admins only) Click this drop-down to manage users and teams.</td>
</tr>
<tr>
<td>5</td>
<td><strong>User Info</strong>: Select this drop-down list to view user information, select the org to operate under, and download the NGC CLI and API key, described later in this document.</td>
</tr>
<tr>
<td>6</td>
<td><strong>Team Selection</strong>: Select this drop-down list to select which team to operate under.</td>
</tr>
</tbody>
</table>
This chapter introduces the NGC Base Command CLI, installable on your workstation for interfacing with Base Command Platform. In this section you will learn about generic features of CLI applicable to all commands as well as CLI modules that map to the Web UI areas that you have learned about in a previous chapter.

The NGC Base Command CLI is a command-line interface for managing content within the NGC Registry and for interfacing with the NVIDIA Base Command Platform. The CLI operates within a shell and lets you use scripts to automate commands.

With NGC Base Command CLI, you can connect with:

- NGC Catalog
- NGC Private Registry
- User Management (available to org or team User Admins only)
- NVIDIA Base Command Platform workloads and entities

### 4.1. About NGC CLI for NVIDIA Base Command Platform

The NGC CLI is available to you if you are logged in with your own NGC account or with an NVIDIA Base Command Platform account, and with it you can:

- View a list of GPU-accelerated Docker containers available to you as well as detailed information about each container image.
- See a list of deep-learning models and resources as well as detailed information about them.
- Download container images, models, and resources.
- Upload and optionally share container images, models, and resources.
- Create and manage users and teams (available to administrators).
- Launch and manage jobs from the NGC registry.
- Download, upload and optionally share datasets for jobs.
- Create and manage workspaces for use in jobs.
4.2. Generating Your NGC API Key

This section describes how to obtain an API key needed to configure the CLI application so you can use the CLI to access locked container images from the NGC Catalog, access content from the NGC Private Registry, manage storage entities, and launch jobs.

The NGC API key is also used for docker login to manage container images in the NGC Private Registry with the docker client.

1. Sign in to the NGC web UI.
   a). From a browser, go to https://ngc.nvidia.com/signin/email and then enter your email
   b). Click Continue by the Sign in with Enterprise sign in option.
   c). Enter the credentials for your organization.

2. In the top right corner, click your user account icon and then select an org that belongs to the NVIDIA Base Command Platform account.

3. Click your user account icon again and select Setup.
4. Click **Get API key** to open the Setup > API Key page.
5. Click **Generate API Key** to generate your API key. A warning message appears to let you know that your old API key will become invalid if you create a new key.
6. Click **Confirm** to generate the key.

Your API key appears.

You only need to generate an API Key once. NGC does not save your key, so store it in a secure place. (You can copy your API Key to the clipboard by clicking the copy icon to the right of the API key.)

Should you lose your API Key, you can generate a new one from the NGC website. When you generate a new API Key, the old one is invalidated.

### 4.3. Installing NGC CLI

To install NGC CLI, perform the following:

2. In the top right corner, click your user account icon and select an org that belongs to the Base Command account.
3. From the user account menu, select **Setup**, then click **Downloads** under Install NGC CLI from the Setup page.
4. From the CLI Install page, click the **Windows**, **Linux**, or **MacOS** tab, according to the platform from which you will be running NGC CLI.
5. Follow the Install instructions that appear on the OS section that you selected.
6. Verify the installation by entering `ngc --version`. The output should be “NGC CLI x.y.z” where x.y.z indicates the version.

### 4.4. Getting Help Using NGC CLI

This section describes how to get help using NGC CLI.

#### 4.4.1. Getting Help from the Command Line

Explain the benefits of the task, the purpose of the task, who should perform the task, and when to perform the task in 50 words or fewer.

To run an NGC CLI command, enter “ngc” followed by the appropriate options.

To see a description of available options and command descriptions, use the option `-h` after any command or option.

**Example 1:** To view a list of all the available options for the `ngc` command, enter

```bash
$ ngc -h
```
**Example 2:** To view a description of all `ngc batch` commands and options, enter

```bash
$ ngc batch -h
```

**Example 3:** To view a description of the `dataset` commands, enter

```bash
$ ngc dataset -h
```

### 4.4.2. Viewing NGC CLI Documentation Online

The [NGC Base Command CLI documentation](#) provides a reference for all the NGC Base Command CLI commands and arguments. You can also access the CLI documentation from the NGC web UI by selecting **Setup** from the user drop down list and then clicking **Documentation** from the Install NGC CLI box.

### 4.5. Configuring the CLI for your Use

To make full use of NGC Base Command CLI, you must configure it with your API key using the `ngc config set` command.

While there are options you can use for each command to specify org and team, as well as the output type and debug mode, you can also use the `ngc config set` command to establish these settings up front.

If you have a pre-existing set up, you can check the current configuration using:

```bash
$ ngc config current
```

To configure the CLI for your use, issue the following:

```bash
$ ngc config set
Enter API key. Choices: [<VALID_APIKEY>, 'no-apikey']:
Enter CLI output format type [ascii]. Choices: [ascii, csv, json]:
Enter org [nv-eagledemo]. Choices: ['nv-eagledemo']:
Enter team [nvtest-repro]. Choices: ['nvtest-repro, 'no-team']:
Enter ace [nv-eagledemo-ace]. Choices: ['nv-eagledemo-ace, 'no-ace']:
Successfully saved NGC configuration to C:\Users\jsmith\.ngc\config
```

If you are a member of several orgs or teams, be sure to select the ones associated with NVIDIA Base Command Platform.

#### 4.5.1. Configuring the Output Format

You can configure the output format when issuing a command by using the `--format_type <fmt>` argument. This is useful if you want to use a different format than the default ascii, or different from what you set when running `ngc config set`.

The following are examples of each output format.

**Ascii**

```bash
$ ngc batch list --format-type ascii
+---------+----------+--------------+------+------------------+----------
| Id      | Replicas | Name         | Team | Status     | Duration | Status Details
|---------|----------|--------------|------|------------|----------|------------------
+---------+----------+--------------+------|------------|----------|------------------
```

**CSV**

```bash
$ ngc batch list --format-type csv
Id,Replicas,Name,Team,Status,Duration,Status Details
1,2,hello world,world,world,2023-03-15 12:00:00,Success
```
4.6. Running the Diagnostics

Diagnostic information is available which provides details to assist in isolating issues. Provide this information when reporting issues with the CLI to NVIDIA support.

The following diagnostic information is available for the NGC Base Command CLI user:

- Current time
- Operating system
- Disk usage
- Current directory size
- Memory usage
- NGC CLI installation
- NGC CLI environment variables (whether set and or not set)
- NGC CLI configuration values
- API gateway connectivity
- API connectivity to the container registry and model registry
- Data storage connectivity
- Docker runtime information
- External IP
- User information (ID, name, and email)
Introduction to the NGC CLI

- User org roles
- User team roles

**Syntax**

```
$ ngc diag [all,client,install,server,user]
```

where

- **all**
  Produces the maximum amount of diagnostic output.

- **client**
  Produces diagnostic output only for the client machine.

- **install**
  Produces diagnostic output only for the local installation.

- **server**
  Produces diagnostic output only for the remote server.

- **user**
  Produces diagnostic output only for the user configuration.

### 4.7. Specifying List Columns

Some commands provide lists, such as a list of registry images or a list of batch jobs.

**Examples:**

- `ngc batch list`
- `ngc dataset list`
- `ngc registry image list`
- `ngc registry model list`
- `ngc registry resource list`
- `ngc workspace list`

The default information includes several columns of information which can appear cluttered, especially if you are not interested in all the information.

For example, the `ngc batch list` command provides the following columns:

```
+----+----------+------+------+--------+----------+----------------+
| Id | Replicas | Name | Team | Status | Duration | Status Details |
+----+----------+------+------+--------+----------+----------------+
```

You can restrict the output to display only the columns that you specify using the `--column` argument.

For example, to display only the Name, Team, and Status, enter

```
$ ngc batch list --column name --column team --column status
```

+----+------+------+--------+
Introduction to the NGC CLI

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Team</th>
<th>Status</th>
</tr>
</thead>
</table>

Note: The Id column will always appear and does not need to be specified.

Consult the help for the \(--\text{column}\) argument to determine the exact values to use for each column.

4.8. Other Useful Command Options

Automatinc Interactive Command Process

Use the \(-y\) argument to insert a \(\text{`yes` (y)}\) response to all interactive questions.

Example:

\$ ngc workspace share \--team <team> -y <workspace>

Testing a Command

Some commands support the \(--\text{dry-run}\) argument. This argument produces output that describes what to expect with the command.

Example:

\$ ngc result remove 1893896 \--dry-run
Would remove result for job ID: 1893896 from org: <org>

Use the \(-h\) argument to see if a specific command supports the \(--\text{dry-run}\) argument.
Chapter 5. Using NGC APIs

This section provides an example of how to use NGC Base Command Platform APIs. For a detailed list of the APIs, refer to the NGC API Documentation.

5.1. Example of Getting Basic Job Information

This example shows how to get basic job information. It shows the API method for performing the steps that correspond to the NGC Base Command CLI command

```
ngc batch get-json {job-id}
```

5.1.1. Using Get Request

The following is the flow using the API Get requests.

1. Get valid authorization.
   
   Send a GET request to `https://authn.nvidia.com/token` to get a valid token.

2. Get the job information.

   Send a GET request to `https://api.ngc.nvidia.com/v2/org/{org-name}/jobs/{job-id}` with the token returned from the first request.

3. Another ask step.

5.1.2. Code Example of Getting a Token

The following is a code example of getting valid authorization (token).

```
#!/usr/bin/python3
import os, base64, json, requests

def ngc_get_token(org='nv-eagledemo', team=None):
    '''Use the api key set environment variable to generate auth token'''
```

Note: API_KEY is the key obtained from the NGC web UI and should be present in your NGC config file if you’ve used the CLI. API_KEY is the key obtained from the NGC web UI and should be present in your NGC config file if you’ve used the CLI.
scope = f'group/ngc:{org}''
if team: #shortens the token if included
    scope += f'/{team}'
querystring = {'service': 'ngc', 'scope': scope}
headers = {
    'Authorization': 'Basic
    {}'.format(base64.b64encode(auth.encode('utf-8')).decode('utf-8')),
    'Content-Type': 'application/json',
    'Cache-Control': 'no-cache',
}
url = 'https://authn.nvidia.com/token'
response = requests.request("GET", url, headers=headers, params=querystring)
if response.status_code != 200:
    raise Exception("HTTP Error %d: from '%s'" % (response.status_code, url))
return json.loads(response.text.encode('utf8'))['token']

Example output of the auth response:

```json
{
    'token':
        'eyJraWQiOiJFUkNPOklCWFY6TjY2SDpOUEgqojNMDtQ6SENVVToyrkFtOkJTkew6WkxKRDpNWk9
        Z0kRVN0o6T1V1W5S1sImsF2y1611JTMjU21n0.eyJzdWIiojP0Tc4bzhnM2JnbGVpNnV1YWx2czY
        xOHNPNSIsImFiZC16Im5nYIsImFjZVcyci1610sImlzcy161F1dGhuLm52aWRpYS5jb2diLCJ
        vCHrVb25zIjpbXSwiXHwiojoxNiYoMDM4MzUtLCJpYYQoMJjEEMjE4Mzc3NTIsImp0aSI6Ijcw
        NQw6yZBB1L7zNmTNDD1BCOM4TU3LTRmYj1LMGRiZGQ4MCJ9.TRCF8cMs1GshUtHaFvByB3p3RWNJR6
        q4SHw1wbee9ppA13ggWreT52h4442p_QJH0so5r73FLrzGeCjJd4bAMX2-Q4dfndV19wFQIZFoxEwe
        fxGByEYKWKAHvFHPSqoeO157dKfdQxwB7QzYxYoR160UbI7dcOuUvgx6vMcBp_2-lXIEGM19gq
        ZJpAFybGWJ2UFJnr41BVxXUyhxmp26uDg6UMDDropW11bTe9xxpOQ8ja5xR1j9o57f99Ld4uRgSi
        4fFMYcOhFaV0zrAcFc2d6BgnbDaxh7O1zQ51LkClurPowoizqNFXuB3L2-DMKQMBHVwVQ1QV7mrvTD
        01j4dXBDho97c8QsmQ11umU27JVLqVTuD-NBGMKzKwDkxBeUYy0nDADsa9PpjoQy45XJBHJG3C2
        Q3OCTmU_h3CCYDG6_f5Ju2Xceuutje66KYJaBFn5RvaovaTXhdWP091XivCw6Eqbhpnc7v2K6_3
        Dt1lG-80aUW-673wRZ6wNVAHTBqBoSo4yFDHALegY1BuodoaubsYrAZfiIvUTJ9St1295xvkr735
        FB-T2ghZTJ5w8glnrQjVm5017Q19mdFHP-pEFrv2ixxQGnsaQLJSz_t8NpEmCQYAcJBsM1VX8W4
        An3RzY26IAzzB0sHvVnA1hipv6HmAICICFPFqAuoQfFv4', 'expires_in': 600}
```

5.1.3. Code Example of Getting Job Information

The token is the output of the function in the Getting a Token section.

```python
def ngc_get_jobinfo(token=None, jobid=None, org=None):
    url = f'https://api.ngc.nvidia.com/v2/org/{org}/jobs/{jobid}'
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {token}'
    }
    response = requests.request("GET", url, headers=headers)
    if response.status_code != 200:
        raise Exception("HTTP Error %d: from '%s'" % (response.status_code, url))
    return response.json()
```

Output of the job info

```json
{"job": {"aceId": 357,
        "aceName": "nv-eagledemo-ace",
        "aceProvider": "NGC",
        "aceResourceInstance": "dgxa100.40g.1.norm",
        "createdDate": "2021-06-04T16:14:31.000Z",
        
```

NVIDIA Base Command Platform

DU-10422-001_v03  |  28
'datasets': [],
'gpuActiveTime': 1.0,
'gpuUtilization': 0.0,
'id': 2039271,
'jobDefinition': {'aceId': 357,
'clusterId': 'eagle-demo.nvk8s.com',
'command': 'set -x; jupyter lab '
"--NotebookApp.token='' --notebook-dir="/ 
"--NotebookApp.allow_origin='*' & date; "
'nvidia-smi; echo $NVIDIA_BUILD_ID; '
'sleep 1d',
'datasetMounts': [],
'dockerImage': 'nvidia/pytorch:21.02-py3',
'jobDataLocations': [{'accessRights': 'RW', 'mountPoint': '/result', 'protocol': 'NFSV3', 'type': 'RESULTSET'}, {'accessRights': 'RW', 'mountPoint': '/result', 'protocol': 'NFSV3', 'type': 'LOGSPACE'}],
'jobType': 'BATCH',
'name': 'NVbc-jupyterlab',
'portMappings': [{'containerPort': 8888, 'hostName': 'https://kpog9271.eagle-demo.proxy.ace.ngc.nvidia.com', 'hostPort': 0}],
'replicaCount': 1,
'resources': {'cpuCores': 30.0, 'gpus': 1, 'name': 'dgxa100.40g.1.norm', 'systemMemory': 124928.0},
'resultContainerMountPoint': '/result',
'runPolicy': {'minTimesliceSeconds': 3600, 'preemptClass': 'RESUMABLE', 'totalRuntimeSeconds': 72000},
'useImageEntryPoint': False,
'workspaceMounts': []},
'jobStatus': {'containerName': '6a977c9461f228b875b00acd6ced1b9a14905a46fca62c5bdbc393409bebe2d', 'createdAt': '2021-06-04T20:05:19.000Z', 'jobDataLocations': [{'accessRights': 'RW', 'mountPoint': '/result', 'protocol': 'NFSV3', 'type': 'RESULTSET'}, {'accessRights': 'RW', 'mountPoint': '/result', 'protocol': 'NFSV3', 'type': 'LOGSPACE'}],
'portMappings': [{'containerPort': 8888, 'hostName': 'https://kpog9271.eagle-demo.proxy.ace.ngc.nvidia.com', 'hostPort': 0}],
'resubmitId': 0,
'selectedNodes': [{'ipAddress': 'ww.x.yy.zz', 'name': 'node-02', 'serialNumber': 'ww.x.yy.zz'}],
'startedAt': '2021-06-04T16:14:42.000Z',
'status': 'RUNNING',
'statusDetails': '',
'statusType': 'OK',
'totalRuntimeSeconds': 14211},
'lastStatusUpdatedDate': '2021-06-04T20:05:19.000Z',
'orgName': 'nv-eagledemo',
'resultset': {'aceName': 'nv-eagledemo-ace',
'aceStorageServiceUrl': 'https://nv-eagledemo.dss.ace.ngc.nvidia.com',}
Using NGC APIs

DU-10422-001_v03 | 30

NVIDIA Base Command Platform
5.1.4. Code Example of Getting Telemetry Data

The token is the output from the Get Token section.

```python
#!/usr/bin/python3
# INFO: Before running this you must run 'export API_KEY=<ngc api key>' in your terminal
import os, json, base64, requests
def get_token(org='nv-eagledemo', team=None):
    # Use the api key set environment variable to generate auth token
    scope = f'group/ngc:{org}'
    if team: #shortens the token if included
        scope += f'/{team}'
    querystring = {'service': 'ngc', 'scope': scope}
    auth = '$oauthtoken:{0}'.format(os.environ.get('API_KEY'))
    auth = base64.b64encode(auth.encode('utf-8')).decode('utf-8')
    headers = {
        'Authorization': 'Basic {0}'.format(auth),
        'Content-Type': 'application/json',
        'Cache-Control': 'no-cache',
    }
    url = 'https://authn.nvidia.com/token'
    response = requests.request("GET", url, headers=headers, params=querystring)
    if response.status_code != 200:
        raise Exception("HTTP Error %d: from '%s'") % (response.status_code, url)
    return json.loads(response.text.encode('utf8'))['token']

def get_job(job_id, org, team, token):
    #''Get general information for a specific job''
    url = f'https://api.ngc.nvidia.com/v2/org/{org}/jobs/{job_id}'
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {token}'
    }
    response = requests.request("GET", url, headers=headers)
    if response.status_code != 200:
        raise Exception("HTTP Error %d: from '%s'") % (response.status_code, url)
    return response.json()

def get_telemetry(job_id, start, end, org, team, token):
    #''Get telemetry information for a specific job''
    url = f'https://api.ngc.nvidia.com/v2/org/{org}/jobs/{job_id}/telemetry'
    # INFO: See the docs for full list of telemetry
    vals = {
        'measurements': [
            {
                "type": "APPLICATION TELEMETRY",
                "aggregation": "MEAN",
                "toDate": end,
                "fromDate": start,
                "period": 60
            },
            {
                "toDate": end,
                "period": 60,
                "aggregation": "MEAN",
                "fromDate": start,
                "type": "GPU_UTILIZATION"
            }
        ]
    }
    params = {'q': json.dumps(vals)}
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {token}'
    }
    response = requests.request("GET", url, params=params, headers=headers)
```
if response.status_code != 200:
    raise Exception("HTTP Error %d: from '%s'" % (response.status_code, url))
return response.json()

# Get org/team information from account setup
org = 'nv-eagledemo'
team = 'nvbc-tutorials'

# Get job ID from GUI, CLI, or other API calls
job_id = 'TODO'

# Generate a token
token = get_token(org, team)
print(token)

# Get general job info for the job of interest
job_info = get_job(job_id, org, team, token)
print(json.dumps(job_info, indent=4, sort_keys=True))

# Get all job telemetry for the job of interest
telemetry = get_telemetry(job_id, job_info['job']['createdDate'],
                          job_info['job']['jobStatus']['endedAt'],
                          org, team, token)
print(json.dumps(telemetry, indent=4, sort_keys=True))

5.2. List of API Endpoints

By using the `--debug` flag in the CLI you can see what endpoints and arguments are used for a given command.

The listed endpoints are all for GET requests but other methods (POST, PATCH, etc...) are supported for different functions. More information can be found here: [https://docs.ngc.nvidia.com/api/](https://docs.ngc.nvidia.com/api/)

<table>
<thead>
<tr>
<th>Section</th>
<th>Endpoints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Management</td>
<td><code>/v2/users/me</code></td>
<td>Get information pertaining to your user such as roles in all teams, datasets, and workspaces that you can access</td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/teams/{team-name}</code></td>
<td>Get description and id of <code>{team-name}</code></td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/teams</code></td>
<td>Get a list of your teams in <code>{org-name}</code></td>
</tr>
<tr>
<td></td>
<td><code>/v2/orgs</code></td>
<td>Get a list of orgs that you can access</td>
</tr>
<tr>
<td>Jobs</td>
<td><code>/v2/org/{org-name}/jobs/{id}</code></td>
<td>Get detailed information about the job, including all create job options, and status history</td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/jobs</code></td>
<td>Get a list of jobs</td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/jobs/*</code></td>
<td>There are many more job commands in the above link that allow you to control jobs</td>
</tr>
<tr>
<td>Section</td>
<td>Endpoints</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Datasets</td>
<td><code>/v2/org/{org-name}/datasets</code></td>
<td>Get a list of accessible datasets in <code>{org-name}</code></td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/datasets/{id}</code></td>
<td>Get information about a dataset including a list of its files</td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/datasets/{id}/file/**</code></td>
<td>Download a file from the dataset</td>
</tr>
<tr>
<td>Telemetry</td>
<td><code>/v2/org/{org-name}/jobs/{id}/telemetry</code></td>
<td>Get telemetry information about the job.</td>
</tr>
<tr>
<td></td>
<td>`/v2/org/{org-name}/measurements/jobs/{id}/[cpu</td>
<td>gpu</td>
</tr>
<tr>
<td>Workspaces</td>
<td><code>/v2/org/{org-name}/workspaces</code></td>
<td>Get a list of accessible workspaces</td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/workspaces/{id-or-name}</code></td>
<td>Get basic information about the workspace</td>
</tr>
<tr>
<td></td>
<td><code>/v2/org/{org-name}/workspaces/{id-or-name}/file/**</code></td>
<td>Download a file from the workspace</td>
</tr>
<tr>
<td>Job Templates</td>
<td><code>/v2/org/{org-name}/jobs/templates/{id}</code></td>
<td>Get info about a job template</td>
</tr>
</tbody>
</table>
Chapter 6. NGC Catalog

This chapter describes the NGC Catalog features of Base Command Platform. NGC Catalog, a collection of software published regularly by NVIDIA and Partners, is accessible through Base Command Web UI and CLI. In this chapter you will learn how to identify and use the published artifacts with Base Command either as is or as a basis for building and publishing your own container images and models.

NGC provides a catalog of NVIDIA and partner published artifacts optimized for NVIDIA GPUs. These are a curated set of GPU-optimized software. It consists of containers, pre-trained models, Helm charts for Kubernetes deployments, and industry-specific AI toolkits with software development kits (SDKs).

Artifacts from NGC Catalog are periodically updated and can be used as a basis for building custom containers for Base Command Platform jobs.

6.1. Accessing NGC Catalog

After logging into the NGC website, click CATALOG from the left-side menu then click one of the options from the top ribbon menu.

- **Collections**: Presents collections of deep learning and AI applications.
- **Containers**: Presents the list of NGC container images.
- **Helm Charts**: Presents a list of Helm charts.
- **Models**: Presents the list of pre-trained deep learning models that can be easily re-trained or fine-tuned.
- **Resources**: Provides a list of step-by-step instructions and scripts for creating deep learning models.

You can also use the filter bar to build a search filter and sorting preference.

### 6.2. Viewing Detailed Application Information

Each card displays the container name and a brief description.

- Click the **Pull Tag** or **Fetch Helm Chart** link (depending on the artifact) to copy the pull or fetch command to your clipboard. Artifacts with a **Download** link will be downloaded to your local disk when the link is clicked.
- Click the artifact name to open to the detailed page.
  
  The top portion of the detailed page shows basic publishing information for the artifact.
  
  The bottom portion of the detailed page shows additional details about the artifact.

### 6.3. Using the CLI

To see a list of container images using the CLI, issue the following command.

```bash
$ ngc registry image list
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Repository</th>
<th>Latest Tag</th>
<th>Image Size</th>
<th>Updated Date</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>nvidia/cuda</td>
<td>11.2.1-devel-</td>
<td>2.18 GB</td>
<td>Feb 17, 2021</td>
<td>unlocked</td>
</tr>
<tr>
<td></td>
<td>ubuntu20.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Other Examples**

To see a list of container images for PyTorch, issue the following.

```bash
$ ngc registry image list nvidia/pytorch*
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Repository</th>
<th>Latest Tag</th>
<th>Image Size</th>
<th>Updated Date</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyTorch</td>
<td>nvidia/pytorch</td>
<td>21.03-py3</td>
<td>5.89 GB</td>
<td>Mar 26, 2021</td>
<td>unlocked</td>
</tr>
</tbody>
</table>

To see a list of container images under the partners registry space, issue the following.

```bash
$ ngc registry image list partners/*
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Repository</th>
<th>Latest Tag</th>
<th>Image Size</th>
<th>Updated Date</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>OmniSci (MapD)</td>
<td>partners/mapd</td>
<td>None</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>Repository</td>
<td>Version</td>
<td>Size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2O Driverless AI</td>
<td>partners/h2oai-</td>
<td>latest</td>
<td>2 GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>driverless</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaddlePaddle</td>
<td>partners/paddlepadd</td>
<td>0.11-alpha</td>
<td>1.28 GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chainer</td>
<td>partners/chainer</td>
<td>4.0.0b1</td>
<td>963.75 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kinetica</td>
<td>partners/kinetica</td>
<td>latest</td>
<td>5.35 GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATLAB</td>
<td>partners/matlab</td>
<td>r2020b</td>
<td>9.15 GB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 7. NGC Private Registry

This chapter describes the Private Registry, a dedicated registry space allocated and accessible just for your organization, which is available to you as a Base Command user. In this chapter, you will learn how to identify your team or org space, how to share container images and models with your team or org, and how to download and use those in your workloads on Base Command Platform.

NGC Private Registry has the same set of artifacts and features available in NGC Catalog. Private Registry provides the space for you to upload, publish, and share your custom artifacts with your team and org with the ability to control access based on the team and org membership. Private Registry enables your org to have your own Catalog accessible only to your org users.

7.1. Accessing the NGC Private Registry

Set your org and team from the User and Select a Team drop-down menus, then click Private Registry from the left-side menu.
Click the menu item to view a list of the corresponding artifacts available to your org or team. Click **Create** to open the screen where you can create the corresponding artifact and save it to your org or team.

**Example of Container Create page**
Example of Model Create page
7.2. Building and Sharing Private Registry Container Images

This section describes how to use a Dockerfile to customize a container from the NGC Private Registry and then push it to a shared registry space in the private registry.

**Note:** These instructions describe how to select a container image from your org and team registry space, but you can use a similar process for modifying container images from the NGC Catalog.

1. Select a container image to modify.
a). Log into the NGC website, selecting the org and team under which you want to obtain a container image.
b). Click PRIVATE REGISTRY from the left-side menu, then click either ORGANIZATIONAL CONTAINERS or TEAM CONTAINERS, depending on who you plan to share your container image with.
c). Locate the container to pull, then click Pull tag to copy the pull command to the clipboard.

2. Pull the container image using the command copied to the clipboard.

3. On your workstation with docker installed, create a subdirectory called mydocker. This is an arbitrary directory name.

4. Inside this directory, create a file called Dockerfile (capitalization is important). This is the default name that Docker looks for when creating a container. The Dockerfile should look similar to the following:

```bash
$ mkdir mydocker $ cd mydocker
$ vi Dockerfile
$ more Dockerfile
FROM nvcr.io/<org>/<team>/<container-name>:{tag}
RUN apt-get update
RUN apt-get install -y octave
```

There are three lines in the Dockerfile.

- The first line in the Dockerfile tells Docker to start with the container nvcr.io/<org>/<team>/<container-name>:<tag>. This is the base container for the new container.
- The second line in the Dockerfile performs a package update for the container. It doesn’t update any of the applications in the container, but updates the apt-get database. This is needed before we install new applications in the container.
- The third and last line in the Dockerfile tells Docker to install the package octave into the container using apt-get.

5. Build the docker container image.

```bash
$ docker build -t nvcr.io/<org>/<team>/<container-image>:<new-tag>
```

Note: This command uses the default file Dockerfile for creating the container. The command starts with docker build. The -t option creates a tag for this new container. Notice that the tag specifies the org and team registry spaces in the nvcr.io repository where the container is to be stored.

6. Verify that Docker successfully created the image.

```
$ docker images
```

7. Push the image into the repository, creating a container. docker push nvcr.io/<org>/<team>/<container-image>:<new-tag>

```bash
docker push nvcr.io/<org>/<team>/<container-image>:<new-tag>
```

8. At this point, you should log into the NGC container registry at https://ngc.nvidia.com and look under your team space to see if the container is there.

If the container supports multi-node:
a). Open the container details page, click the menu icon from the upper right corner, then click **Edit Details**.

b). Click the Multi-node Container check box.

c). Click the menu icon and then click **Save**.

If you don’t see the container in your team space, make sure that the tag on the image matches the location in the repository. If, for some reason, the push fails, try it again in case there was a communication issue between your system and the container registry (`nvcr.io`).
Chapter 8. Org, Team, and User Management

This chapter applies to organization and team administrators, and explains the tasks that an organization or team administrator can perform from the NGC website or CLI. In this chapter, you will learn about the different user roles along with their associate scopes and permissions available in Base Command Platform, and the features to manage users and teams.

8.1. Org and Team Overview

Every enterprise is assigned to an “org”, the name of which is determined by the enterprise at the time the account is set up. NVIDIA Base Command Platform provides each org with its own private registry space for running jobs, including storage and workspaces.

One or more teams can be created within the org to provide private access for groups within the enterprise. Individual users can be members of any number of teams within the org.

As the NVIDIA Base Command Platform administrator for your organization, you can invite other users to join your organization’s NVIDIA Base Command Platform account. Users can then be assigned as members of teams within your organization. Teams are useful for keeping custom work private within the organization.

The following table illustrates the interrelationship between orgs, teams, and users:

<table>
<thead>
<tr>
<th></th>
<th>ORG</th>
<th>TEAM 1</th>
<th>TEAM 2</th>
<th>TEAM 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registry Space</td>
<td>&lt;org&gt;/</td>
<td>&lt;org&gt;/&lt;team1&gt;</td>
<td>&lt;org&gt;/&lt;team2&gt;</td>
<td>&lt;org&gt;/&lt;team3&gt;</td>
</tr>
<tr>
<td>Org Admin</td>
<td>Can add users to the org/, or to any org/team. Can create teams.</td>
<td>Can add users to org/team1</td>
<td>Can add users to org/team2</td>
<td>Can add users to org/team3</td>
</tr>
<tr>
<td>Org User</td>
<td>Can access resources and launch jobs within the org, but not within teams</td>
<td>Can access and share resources and launch jobs within org/team1</td>
<td>Can access and share resources and launch jobs within org/team2</td>
<td>Can access and share resources and launch jobs within org/team3</td>
</tr>
<tr>
<td>Org Viewer</td>
<td>Can read resources and jobs within the org.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NVIDIA Base Command Platform
The general workflow for building teams of users is as follows:

1. The organization admin invites users to the organization’s NVIDIA Base Command Platform account.
2. The organization admin creates teams within the organization.
3. The organization admin adds users to appropriate teams, and typically assigns at least one user to be the team admin.
4. The organization or team admin can then add other users to the team.

8.2. NVIDIA Base Command Platform User Roles

Prior to adding users and teams, familiarize yourself with the following descriptions of each role.

**Base Command Admin**

The Base Command Admin (``BASE_COMMAND_ADMIN``) is the role assigned to the Base Command org administrator for the enterprise.

The following is a summary of the capabilities of the org administrator:

- Access to all read-write and appropriate share commands involving the following features:
  - Jobs, workspaces, datasets, and results within the org.
- Team administrators have the same capabilities as the org administrator with the following limits:
  - Capabilities are limited to the specific team.

**Base Command User Role**

The Base Command User role (``BASE_COMMAND_USER``) can make use of all NVIDIA Base Command Platform tasks. This includes all read, write, and appropriate sharing capabilities for jobs, workspaces, datasets, and results within the user’s org or team.

**Base Command Viewer Role**

The Base Command Viewer user (``BASE_COMMAND_VIEWER``) has the same scope as the Base Command Admin but with read-only access to all jobs, workspaces, datasets, and results within the scope of the role (org or team).
Registry Admin Role

The Registry Admin (\texttt{REGISTRY_USER_ADMIN}) is the role assigned to the initial org administrator for the enterprise.

The following is a summary of the capabilities of the registry admin org administrator:

- Access to all read-write and appropriate share commands involving the following features:
  - Containers, models, and resources within the org

Team administrators have the same capabilities as the org administrator with the following limits:

- Capabilities are limited to the specific team.
- Team administrators cannot create other teams or delete teams

Registry Only Role

The Registry Only (\texttt{REGISTRY_ONLY}) role has read-only access to containers, models, and resources within the user’s org or team.

Registry User Role

The Registry User (\texttt{REGISTRY_USER_USER}) can make full use of all Private Registry features. This includes all read, write, and appropriate sharing capabilities for containers, models, and resources within the user’s org or team.

User Admin Role

The User Admin (\texttt{USER_ADMIN}) manages users within the org or team. The User Admin for an org can create teams within that org.

8.3. Assigning Roles

Each role is targeted for specific capabilities. When assigning roles, keep in mind all the capabilities you want the user or admin to achieve. Most users and admins will need to be assigned multiple roles. Use the following tables for guidance:

Assigning Admin Roles

Refer to the following table for a summary of the capabilities of each admin role. You may need to assign multiple roles depending on the capabilities you want the admin to have.

<table>
<thead>
<tr>
<th>Role</th>
<th>Users or Teams</th>
<th>Jobs, Workspaces, datasets, results</th>
<th>Container, models, resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Command Admin</td>
<td>N/A</td>
<td>Read/Write</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Org, Team, and User Management

<table>
<thead>
<tr>
<th>Role</th>
<th>Users or Teams</th>
<th>Jobs, Workspaces, datasets, results</th>
<th>Container, models, resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Command Viewer</td>
<td>N/A</td>
<td>Read Only</td>
<td>N/A</td>
</tr>
<tr>
<td>Registry Admin</td>
<td>N/A</td>
<td>N/A</td>
<td>Read/Write</td>
</tr>
<tr>
<td>User Admin</td>
<td>Read/Write</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Example:** To add an admin for user management, registry management, and job management, issue the following:

```bash
$ ngc org add-user <email> "<name>" --role USER_ADMIN --role REGISTRY_USER_ADMIN --role BASE_COMMAND_ADMIN
```

Assigning User Roles

Refer to the following table for a summary of the capabilities of each user role. You may need to assign multiple roles depending on the capabilities you want the user to have.

<table>
<thead>
<tr>
<th>Role</th>
<th>Users</th>
<th>Jobs, Workspaces, datasets, results</th>
<th>Container, models, resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Command User</td>
<td>N/A</td>
<td>Read/Write</td>
<td>N/A</td>
</tr>
<tr>
<td>Registry Only</td>
<td>N/A</td>
<td>N/A</td>
<td>Read Only</td>
</tr>
<tr>
<td>Registry User</td>
<td>N/A</td>
<td>N/A</td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

**Example:** To add a user who can run jobs using custom containers, issue the following:

```bash
$ ngc org add-user <email> "<name>" --role BASE_COMMAND_USER --role REGISTRY_USER
```

8.4. Org and Team Administrator Tasks

For org or team admins the most likely commands needed are adding users. The following is the typical process for adding users using the CLI.

1. Add a user to an org:
   ```bash
   $ ngc org add-user <email> "<name>" --role <user-role>
   ```
2. Create a team:
   ```bash
   $ ngc org add-team <name> <description>
   ```
3. Add a User to a team (and to the org if they are not already a member):
   ```bash
   $ ngc team add-user --team <team> <email> "<name>" --role <user-role>
   ```

Other commands to list users, add additional admins, can be looked up with `ngc org --help` or `ngc team --help` or in the CLI documentation.

8.4.1. Creating Teams Using the Web UI
Creating teams is useful for allowing users to share images within a team while keeping them invisible to other teams in the same organization. Only organization administrators can create teams.

To create a team, do the following:

2. Select ORGANIZATION > Teams from the left side menu.
3. Click the “X” icon at the lower right corner, then select the Create Team icon.

4. In the Create Team dialog, enter a team name and description, then click Create Team.

8.4.2. Creating Users Using the Web UI
As the organization administrator, you must create user accounts to allow others to use the NVIDIA Base Command Platform within the organization.

1. Log on to the NGC website.
2. Click **ORGANIZATION > Users** from the left side menu.
3. Click the “X” icon at the lower right corner, then select the **Invite New User** icon.

4. Select the **Personal Info** tab and then enter the display name and email where indicated.

5. Click **Next** or select the **Membership** tab and then select one or more org roles.
6. [Optional] Select a team for the user and select one or more roles for the user to have within that team.

7. Click Assign.

   If you want to assign the user to more than one team, select another team and role, then click Assign.

   The following example screenshot shows a user assigned to two teams:
8. Click **Confirm** to complete the process.  
An invitation email is automatically sent to the user.
Chapter 9. NVIDIA Base Command Platform Data Concepts

This chapter describes the storage data entities available in Base Command Platform. In this chapter, you will learn datasets, workspaces, results, and storage space local to a computing instance along with their use cases. You will learn about actions that you can perform on these data storage entities from within a computing instance and from your workstations, both from the Web UI and from the CLI.

9.1. Data Types

NVIDIA Base Command Platform has three data types on network storage within the ACE;

- **Result**: Private to a job, read-write artifact, generated during a job
- **Dataset**: Shareable read-only artifact, mountable to a job
- **Workspace**: Shareable read-write artifact, mountable to a job
- **Local scratch space**: Private to a node read-write local scratch space only on full-node instances.

9.2. Managing Datasets

Datasets are intended for read-only data suitable for production workloads with repeatability, provenance, and scalability. They can be shared with your team or entire organization.

9.2.1. Determining Datasets by Org or Team

To view a list of datasets using the NGC website, click Datasets from the left-side menu, then select one of the tabs from the ribbon menu, depending on whether you want to view all datasets available to you, only datasets available to your org, or only datasets available to your team.
9.2.2. Mounting Datasets in a Job

Datasets are a critical part of a deep learning training job. They are intended as performant shareable read-only data suitable for production workload with repeatability and scalability. Multiple datasets can be mounted to the same job. Multiple jobs and users can mount a dataset concurrently.

To mount one or more datasets, specify the datasets and mount points from the NGC Job Creation page when you create a new job.

1. From the Data Input section, select the Datasets tab and then search for a dataset to mount using the available search criteria.
2. Select one or more datasets from the list.
3. Specify a unique mount point for each dataset selected.

9.2.3. Downloading a Dataset Using the Web UI

To download a dataset using the NGC website, select a dataset from the list to open the details page for the selected dataset.

Click the File Browser tab, then select one of the files to download.

The file will download to your Download folder.

9.2.4. Managing Datasets Using the NGC CLI

Concept definition.

Uploading and Sharing a Dataset

Creating, uploading, and optionally sharing a dataset is done in one step:

```
$ ngc dataset upload --source <dir> --desc "my data" <dataset_name> [--share <team_name>]
```

Example:

```
$ ngc dataset upload --source mydata/ --desc "mnist is great" mnist --share my_team1
```
To share with multiple teams, use multiple `--share` arguments.

**Example:**

```bash
$ ngc dataset upload --source mydata/ --desc "mnist is great" mnist --share my_team1 --share my_team2
```

**Tip:** While the `--share` argument is optional, using the `--share` argument when uploading the dataset is a convenient way to make sure your datasets are shared so you don’t have to remember to share them later.

**Important:** Never reuse the name of a dataset because your organization will lose the ability to repeat and validate experiments.

### Sharing a Dataset with your Team

You must share your dataset with your team in order for your team members to use it. If you did not use the `--share` argument when uploading the dataset, you can share the dataset with your team afterwards:

```bash
$ ngc dataset share --team <team_name> <dataset_id>
```

**Example:**

```bash
$ ngc dataset share --team my_team 5586
```

To share with your entire org, use `--team 'no-team'`. Please communicate to your org admin about sharing a dataset to org, as it should be documented and published before doing so.

**Example:**

```bash
$ ngc dataset share --team no-team 5586
```

### Listing Datasets

Listing existing datasets available:

```bash
$ ngc dataset list
```

This lists all the datasets available to the configured org and team.

**Example output:**

```bash
$ ngc dataset list 5586
```

Use `-h` option for list command to show all context based options including `--owned` which is useful to list only those dataset owned by the user.

### Listing Datasets Owned by you

```bash
$ ngc dataset list --owned
```

### Listing Datasets Within a Team

```bash
$ ngc dataset list --team <team_name>
```
$ ngc dataset list --team <teamname>

**Downloading a Dataset**

To download a dataset, determine the dataset ID from the NGC website, then issue the following command to download the dataset to the current folder.

$ ngc dataset download <datasetid>

To download to a specific existing folder, specify the path in the command.

$ ngc dataset download <datasetid> --dest <destpath>

**Deleting a Dataset**

To delete a dataset from NGC on an ACE:

$ ngc dataset remove <datasetid>

### 9.2.5. Converting a Checkpoint to a Dataset

For some workflows, such as for use with Transfer Learning Toolkit (TLT), you may need to save a checkpoint for a duration longer than that of the current project. These can then be shared with your team.

NVIDIA Base Command lets you save checkpoints from a training job as a dataset for long term storage and for sharing with a team. Depending on the job configuration, checkpoints are obtained from the job /results mount or the job workspace mount.

#### 9.2.5.1. Saving a Checkpoint from the /results Mount

To save a checkpoint from the /results mount, download the contents and then upload as a dataset as follows:

1. Download the /results to your local disk.
   
   $ ngc result download <job-id[:replica-id]> --dest <download-path>

   Use the -h option to see a list of options for specifying specific folders and even files within /results to download.

   The contents are downloaded to a folder labeled <job-id>.

2. Upload the file(s) to a dataset.
   
   $ ngc dataset upload <dataset-name> --source <path-to-files>

   The files are uploaded to the set ACE.

#### 9.2.5.2. Converting /result to a Dataset Using the NGC Web UI

**CAUTION:** Converting the contents of the /result files removes the /result files altogether.

You can convert /result to a dataset from the NGC web UI.
1. From either the **Base Command > Dashboard** or **Base Command > Jobs** page, click the menu icon for the job containing the /result files to convert, then select **Convert Results**.

2. Enter a name and (optionally) a description in the Convert Results to Dataset dialog.

   ![Convert Results to Dataset dialog](image)

3. Click **Convert** when done. The dataset is created, which you can view from the **Base Command > Datasets** page.

### 9.2.5.3. Converting /result to a Dataset Using the CLI

You can also convert using the NGC Base Command CLI as follows:

```bash
$ ngc dataset convert <new-dataset-name> --from-result <job-id>
```

### 9.2.5.4. Saving a Checkpoint from the Workspace
To save a checkpoint from your workspace, download the workspace and then upload as a dataset as follows:

1. Download the workspace to your local disk.

   ```bash
   $ ngc workspace download <workspace-id> --dest <download-path>
   ```

   Use the `-h` option to see a list of options for specifying specific folders and even files within the workspace to download. The contents are downloaded to a folder labeled `<workspace-id>`.

2. Upload the file(s) to a dataset.

   ```bash
   $ ngc dataset upload <dataset-name> --source <path-to-files>
   ```

   The files are uploaded to the set ACE.

### 9.2.6. Building a Dataset from External Sources

Many deep learning training jobs use publicly available datasets from the internet, licensed for specific use cases. If you need to use such datasets, NVIDIA recommends cloning the dataset to avoid running a training job using files from external sources on every run.

The best method for doing this is to run a job and then build the dataset from the job. Follow these steps to build a dataset from a website (or external public server).

1. Run an interactive job (one with Jupyter Lab works well) on a 16GB 1-GPU instance.
2. Execute commands to download and process files to /result mount.
3. After the job is finished, use ngc dataset convert to convert the processed files from /result to a new dataset.

The convert process moves the files from result type to dataset type and thus is a one-time operation.

### 9.3. Managing Workspaces

Workspaces are shareable read-write persistent storage mountable in a job for concurrent use. They are intended as a tool for read-write volumes providing scratch space between jobs or users. They have an ID and can be named. They count towards your overall storage quota.

The primary use case for a workspace is to share persistent data between jobs; for example, to use for checkpoints or for retraining.

Workspaces also provide an easy way for users in a team to work together in a shared storage space. Workspaces are a good place to store code, can easily be synced with git, or even updated while a job is running, especially an interactive job. This means you can experiment rapidly in interactive mode without uploading new containers or datasets for each code change.

### 9.3.1. Workspace Limitations

- No repeatability or other production workflow guarantees, auditing, provenance, etc.
- Read/write race conditions, with undefined write ordering.
File locking behavior is undefined.
Bandwidth and IOPS performance are limited like any shared file system.

9.3.2. Examples of Workspace Use Cases

- Multiple jobs can write to a workspace and be monitored with TensorBoard.
- Users can use a Workspace as a network home directory.
- Teams can use a Workspace as a shared storage area.
- Code can be put in a Workspace instead of the container while it’s still being iterated on and used by multiple jobs during experimentation (see dangers above)

9.3.3. Mounting Workspaces from the Web UI

Workspaces provide an easy solution for any use cases.

To mount one or more workspaces, specify the workspaces and mount points from the NGC Job Creation page when you create a new job.

1. From the Data Input section, select the Workspaces tab and then search for a workspace to mount using the available search criteria.
2. Select one or more workspaces from the list.
3. Specify a unique mount point for each workspace selected.

9.3.4. Creating a Workspace

9.3.4.1. Creating a Workspace Using the Web UI

1. Select Base Command > Workspaces from the left-side menu, then click the “X” icon on the lower right corner and select the Create New Workspace icon.
2. In the **Create a Workspace** dialog, enter a workspace name and select an ACE to associate with the workspace.

3. Click **Create**.

   The workspace is added to the workspace list.

### 9.3.4.2. Creating a Workspace Using the Base Command CLI

Creating a workspace involves a single command which outputs the resulting Workspace ID:

```
$ ngc workspace create --name <workspace-name>
```

Workspaces can be named for easy reference. It can be named only one time, i.e. a workspace can’t be renamed. You can name the workspace when it is created, or name it afterwards.

### 9.3.4.3. Using Unique Workspace Names

Since a workspace can be specified by name and id, it is imperative that those are unique across both names and ids. Workspace id is generated by the system whereas the name is specified by the user. Workspace id is always 22 chars long. In order to ensure that a user specified name does not match a future workspace id, workspace names with exactly 22 chars are not allowed.

Workspace names must follow these constraints:

- The name cannot be 22 chars long.
- The name must start with an alphanumeric.
- The name can contain alphanumeric, -, or _ characters.
- The name must be unique within the org.

These restrictions are also captured in regex `^[a-zA-Z0-9_\-]{22}$`.

### 9.3.4.4. Naming the Workspace When it is Created

```
$ ngc workspace create --name ws-demo
```

Successfully created workspace with id: XB1Cym98QWmsX79wf0n3Lw

**Workspace Information**

- **ID:** XB1Cym98QWmsX79wf0n3Lw
- **Name:** ws-demo
- **Created By:** John Smith
- **Size:** 0 B
- **ACE:** nv-us-west-2
- **Org:** nvidian
- **Description:**
- **Shared with:** None

### 9.3.4.5. Naming the Workspace after it is Created

Example of creating a workspace without naming it.

```
$ ngc workspace create
```

Successfully created workspace with id: s67Bcb_GQU6g75X0gl0n8g
If you created a workspace without naming it, you can name it later by specifying the id and using the `set -n <name>` option.

```
$ ngc workspace set -n ws-demo s67Bcb_GQU6g75X0g10n8g -y
```

Workname name for workspace with id s67Bcb_GQU6g75X0g10n8g has been set.

```
$ ngc workspace info ws-demo
```

9.3.5. **Listing Workspaces**

You can list the workspaces you have access to, and get the details of a specific workspace:

```
$ ngc workspace list
```

```
+-----------------+------------+--------------+--------------+----------------+---
| Id              | Name       | Description  | ACE          | Creator        |
|                 |            |              |              | Username       |
|-----------------+------------+--------------+--------------+----------------+---
| s67Bcb_GQU6g75X | ws-demo    |              | nv-us-west-2 | Sabu Nadarajan |
| OglOn8g         |            |              |              |                |
|-----------------+------------+--------------+--------------+----------------+---
```

```
$ ngc workspace info ws-demo
```

9.3.6. **Using Workspace in a Job**

**CAUTION:** Most of NVIDIA DL images already have a directory `/workspace` that contains NVIDIA examples. When a mount point for your workspace is specified in the job definition, take precaution that it does not conflict with the existing directory in the container. Use a directory name that is unique and does not exist in the container. In the examples below, the name of the workspace is used as the mounting point.

Access to workspace is made available in a job by specifying a mount point in the command line to run a job.

```
$ ngc batch run -i nvidia/tensorflow:18.10-py3 -in dgx1v.16g.1.norm --ace nv-us-west-2 -n HowTo-workspace --result /result --commandline 'sleep 5h'
```

```
Job Information
Id: 223282
```
A workspace is mounted in Read-Write (RW) mode by default. Mounting in Read-Only (RO) mode is also supported. In RO mode, it functions similarly to a dataset.

```bash
$ ngc batch run -i nvidia/tensorflow:18.10-py3 -in dgx1v.16g.1.norm --ace nv-us-west-2 -n HowTo-workspace --result /result --commandline 'sleep 5h' --datasetid 8181:/dataset --workspace ws-demo:/ws-demo:RO
```

Specifying a workspace in a job using a JSON file is shown below. The example below is derived from the first job definition shown in this section.

```json
{
    "aceId": 357,
    "aceInstance": "dgxa100.40g.1.norm",
    "aceName": "nv-eagledemo-ace",
    "command": "sleep 5h",
    "datasetMounts": [
        {
            "containerMountPoint": "/dataset",
            "id": 8181
        }
    ],
    "dockerImageName": "nvidia/tensorflow:18.10-py3",
    "name": "HowTo-workspace",
    "resultContainerMountPoint": "/result",
    "runPolicy": {
        "preemptClass": "RUNONCE"
    },
    "workspaceMounts": [
        {
            "containerMountPoint": "/ws-demo",
            "id": "ws-demo",
            "mountMode": "RW"
        }
    ]
}
```
9.3.7. Uploading and Downloading Workspaces

Explain what the concept is and why the reader should care about it in 50 words or fewer.

Mounting a workspace to access or transfer a few files works great. If you need to do a bulk transfer of many files like populating an empty workspace at beginning or downloading an entire workspace for archiving, workspace upload and download commands work better.

Uploading a directory to workspace is similar to uploading files to a dataset.

```bash
$ ngc workspace upload --source ngc140
  s67Bcb_GQU6g75XOgl10n8g
Total number of files is 6459.
Uploaded 170.5 MB, 6459/6459 files in 9s, Avg Upload speed: 18.82 MB/s, Curr Upload Speed: 25.9 KB/s
----------------------------------------------------
Workspace: s67Bcb_GQU6g75XOgl10n8g Upload: Completed.
Imported local path (workspace): /home/ngccli/ngc140
Files transferred: 6459
Total Bytes transferred: 17877265 B
Started at: 2018-11-17 18:26:33.399256
Completed at: 2018-11-17 18:26:43.148319/
Duration taken: 9.749063 seconds
----------------------------------------------------
```

Downloading workspace to a local directory is similar to downloading results from a job.

```bash
$ ngc workspace download --dest temp s67Bcb_GQU6g75XOgl10n8g
Downloaded 56.68 MB in 41s, Download speed: 1.38 MB/s
----------------------------------------------------
Transfer id: s67Bcb_GQU6g75XOgl10n8g Download status: Completed.
Downloaded local path: /home/ngccli/temp/s67Bcb_GQU6g75XOgl10n8g
Total files downloaded: 6459
Total downloaded size: 56.68 MB
Started at: 2018-11-17 18:31:03.530342
Completed at: 2018-11-17 18:31:45.592230
Duration taken: 42s seconds
----------------------------------------------------
```

9.3.8. Workspace Sharing and Revoking Sharing

Workspaces can be shared with a team or with the entire org.

**Important:** Each workspace is private to the user who creates it until you decide to share with your team. Once you share with your team, all team members have the same rights in that workspace, so have a sharing protocol before you share. For instance one way of using a workspace is to have a common area which only the owner updates, and multiple user directories, one per user where each user can write their own data.

Sharing a workspace with a team:

```bash
$ ngc workspace info ws-demo
```

```bash
Workspace Information
ID: s67Bcb_GQU6g75XOgl10n8g
ACE: nv-us-west-2
Org: nvidian
Description: none
Shared with: None
```
$ ngc workspace share --team nves -y ws-demo
Workspace successfully shared
$ ngc workspace info ws-demo
----------------------------------
Workspace Information
ID: s67Bcb_GQU6g75XOglOn8g
Name: ws-demo
ACE: nv-us-west-2
Org: nvidian
Description:
Shared with: nvidian/nves
----------------------------------

Revoking a shared workspace:
$ ngc workspace revoke-share --team nves -y ws-demo
Workspace share successfully revoked
$ ngc workspace info ws-demo
----------------------------------
Workspace Information
ID: s67Bcb_GQU6g75XOglOn8g
Name: ws-demo
ACE: nv-us-west-2
Org: nvidian
Description:
Shared with: None
----------------------------------

9.3.9. Removing Workspaces

$ ngc workspace remove ws-demo
Are you sure you would like to remove the workspace with ID or name: ws-demo from org: nvidian? [y/n]y
Successfully removed workspace with ID or name: ws-demo from org: nvidian.

9.4. Managing Results

The Job Result contains any files your job has written to the result mount and a joblog.log.

Downloading a Result
To download the Result of a Job, issue the following:

$ ngc result download <job_id>
The content is downloaded to a folder named <job_id>.

Remove a Result
Results remain in the system consuming quota until removed:

$ ngc result remove <job_id>

joblog.log
The output ofSTDOUT and STDERR are put into the joblog.log file in the result.
Converting Results into Datasets

1. Click JOBS from the left-side menu, then click the menu icon for the job in which you want to convert the results to a dataset and select Convert Results to Dataset.
2. In the Convert Results to Dataset dialog box, provide a name and description for your dataset, then click Convert.

When the conversion is completed, your dataset appears in the Dataset list page.

Remember to share your dataset with others in your team or org as described in Sharing a Dataset with your team.

9.5. Local Scratch Space (/raid)

All Base Command nodes come with several SSD drives configured as a RAID-0 array cache storage. This scratch space is mounted in every full-node job at /raid.

A typical use of this /raid scratch space can be to store temporary results/checkpoints that are not required to be available after a job is finished or killed. Using this local storage for intermediate results/logs will avoid heavy network storage access (such as results and workspaces) and should improve job performance. The data on this scratch space is cleared (and not automatically saved/backed-up to any other persistent storage) after a job is finished. Consider /raid to be a temporary scratch space available during the lifetime of the job.

Since the /raid volume is local to a node, the data in it is not backed-up and transferred when a job is preempted and resumed. It is the responsibility of the job/user to periodically backup the required checkpoint data to the available network storage (results or workspaces) to enable resuming a job (which is almost certainly on a different node) after a preemption.

Tip: Another use case for a local-to-node /raid volume is to copy the dataset from network storage to /raid and use that mount point for training. This works well for jobs with many epochs using datasets which are reasonable in size to replicate to local storage. Note that contents of /raid volume is not carried over to the new node when a job is preempted and resumed, and that the required info must be saved in an available network storage space for resuming the job using the data.
Chapter 10. Jobs and GPU Instances

This chapter describes Base Command features for submitting jobs to the GPU instances, and for managing and interacting with the jobs. In this chapter, you will learn how to identify GPU instances and their attributes available to you, how to define jobs to associated storage entities, and how to manage the jobs using either the Web UI or the CLI.

10.1. Identifying Available ACE and Instances

- If you are creating jobs from the NGC website (BASE COMMAND > JOBS > Create), the available ACE and instances are presented as choices for you to select.
- To determine the available ACE and instances from the CLI, issue
  
  ```
  $ ngc ace list
  ```

  The output shows the available ACEs as well as the instances for the ACE.

10.2. Running a Simple Job

The section describes how to run a simple ‘Hello world’ job.

1. Login to the NGC Dashboard and click BASE COMMAND > JOBS from the left-side menu.
2. In the upper right select Create a job.
3. Under the ACE box, select your Accelerated Computing Environment and Instance type. Select an instance size to use.
4. Under Data Output, choose a mount point to access results. This is typically /result.
   
   This is typically /result.
5. Under the Container Selection area:
   
   a). Select a container image and tag, such as
      
      - Select a Container: nvidia/tensorflow
      - Tag: 21.03-tf1-py3

   b). Enter your Command; for example, echo ‘Hello from NVIDIA’.
6. At the bottom of the screen, enter a name for your job.
7. Click **Launch**.

   Alternatively, click the copy icon in the command box and then paste the command into the command line.

Click the **JOBS** from the left-side menu to view the status of your job. The Status History column reports the following progress with the timestamps: Created -> Queued -> Starting -> Running -> Finish.

### 10.3. Running JupyterLab in a Job

This section describes how to run a simple ‘Hello world’ job incorporating JupyterLab.

NGC containers include JupyterLab within the container image. Using JupyterLab is a convenient way to run notebooks, get shell access (multiple sessions), run tensorboard, and have a file browser and text editor with syntax coloring all in one browser window. Running it in the background in your job is non-intrusive without any additional performance impact or effort and provides you an easy option to peek into your job at any time.

#### 10.3.1. Example of Running JupyterLab in a Job

The following is an example of a job that takes advantage of JupyterLab.

```bash
ngc batch run --name "jupyter-example" --preempt RUNONCE --min-timeslice 1s --total-runtime 600s --ace nv-eagledemo-ace --instance dgxa100.40g.1.norm --commandline "jupyter lab --ip=0.0.0.0 --allow-root --no-browser --NotebookApp.token='' --notebook-dir=/ --NotebookApp.allow_origin='*' & date; nvidia-smi; sleep 1d" --result /result --image "nvidia/pytorch:21.03-py3" --org nv-eagledemo --port 8888
```

**Job Information**

- **Id**: 1101563
- **Name**: jupyter-example

These are some key aspects to using JupyterLab in your job.

- Specify `--port 8888` in the job definition.
  
  The Jupyter lab port (8888 by default) must be exposed by the job.

- Jupyter should run in the background by using `&` (example `& date`).
  
  This is required for your actual job command to be executed and have Jupyter run as long as the job is running.

- The JupyterLab command must begin with the `jupyter lab`.

- Total runtime should be set to a reasonable number to access the container before it finishes the job and closes

#### 10.3.2. Connecting to JupyterLab

While the job is in a running state, you can connect to JupyterLab through the mapped URL as follows.
From the website, click the URL presented in the Mapped Port section of the job details page.

From the CLI, run `$ ngc batch info <job-id>` and then copy the URL in the Port Mappings line and paste into a browser.

Example of JupyterLab:

![JupyterLab interface](image)

### 10.4. Cloning an Existing Job

You can clone jobs, which is useful when you want to start with an existing job and make small changes for a new job.

1. Click Dashboard from the left-side menu, click the table view icon next to the search bar, then click the menu icon for the job you want to copy and select Clone Job from the menu.
The create a job page opens with the fields populated with the information from the cloned job.

2. Edit fields as needed to create a new job, enter a unique name in the **Name** field, then click **Launch**.

The job should appear in the job dashboard.

### 10.5. Launching a Job from a Template File

1. Click **BASE COMMAND > JOBS > Create** from the left-side menu and then click **Create From Templates** from the ribbon menu.

2. Click the menu icon for the template to use, then select **Apply Template**.
The create a job page opens with the fields populated with the information from the job template.

3. Edit fields as needed to create a new job or leave the fields as is, then click **Launch**.

### 10.6. Launching a Job Using a JSON File

When running jobs repeatedly from the CLI, sometimes it is easier to use a template file than the command line flags. This is currently supported in JSON. The following sections describe how to generate a JSON file from a job template and how to use it in the CLI.

#### 10.6.1. Generating the JSON Using the Web UI

Perform the following to generate a JSON file using the NGC web UI.

1. Click Dashboard from the left-side menu, click the table view icon next to the search bar, then click the menu icon for the job you want to copy and select Copy to JSON. The JSON is copied to your clipboard.
2. Open a blank text file, paste the contents into the file and then save the file using the extension **.json**. Example: **test-json.json**
3. To run a job from the file, issue the following: ```$ ngc batch run -f <file.json>``` A task step.

#### 10.6.2. Generating the JSON Using the CLI

Alternatively, you can get the JSON using the CLI if you know the job ID as follows:

```$ ngc batch get-json <job-id> > <path-to-json-file>``` The JSON is copied to the specified path and file.

**Example:**

```$ ngc batch get-json 1234567 > ./json/test-json.json``` To run a job from the file, issue the following:
$ ngc batch run -f <file.json>

**Example:**

$ ngc batch run -f ./json/test-json.json

### 10.6.3. Overriding Fields in a JSON File

The following is an example JSON:

```json
{
  "dockerImageName":"nvidia/tensorflow:19.11-tf1-py3",
  "aceName":"nv-us-west-2",
  "name":"test.exempt-demo",
  "command":"jupyter lab --ip=0.0.0.0 --allow-root --no-browser --NotebookApp.token='' --no-browser --no-browser --NotebookApp.token='' --notebook-dir=/ --NotebookApp.allow_origin='*' & date; sleep 1h",
  "description":"sample command description",
  "replicaCount":1,
  "runPolicy":{
    "totalRuntimeSeconds":3600,
    "preemptClass":"RUN_ONCE"
  },
  "workspaceMounts":[
    {
      "containerMountPoint":"/mnt/democode",
      "id":"KUlaYYvXT56IhuKpNqmorQ",
      "mountMode":"RO"
    }
  ],
  "aceId":257,
  "networkType":"ETHERNET",
  "datasetMounts":[
    {
      "containerMountPoint":"/data/imagenet",
      "id":59937
    }
  ],
  "resultContainerMountPoint":"/result",
  "aceInstance":"dgx1v.32g.8.norm.beta"
}
```

You can specify other arguments in the command, but if they are specified in the JSON file, then the argument values will override the values in the JSON file.

See table below for mapping the field in template to option name in command line.

<table>
<thead>
<tr>
<th>CLI option</th>
<th>JSON Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>--commandline</td>
<td>command</td>
</tr>
<tr>
<td>--description</td>
<td>description</td>
</tr>
<tr>
<td>--file</td>
<td>none</td>
</tr>
<tr>
<td>--help</td>
<td>none</td>
</tr>
<tr>
<td>--image</td>
<td>dockerImageName</td>
</tr>
<tr>
<td>--instance</td>
<td>aceInstance</td>
</tr>
<tr>
<td>--name</td>
<td>name</td>
</tr>
<tr>
<td>--port</td>
<td>port (pass in a list of ports [8888,6006])</td>
</tr>
<tr>
<td>--workspace</td>
<td>workspaceMounts (pass in a list of objects)</td>
</tr>
</tbody>
</table>
### CLI option | JSON Key
--- | ---
--ace | ace
--array-type | none
--coscheduling | none
--datasetid | datasetMounts (pass in a list of objects)
--debug | none
--entrypoint | none
--format_type | none
--min-availability | none
--min-timeslice | none
--network | networkType
--org | none
--preempt | runPolicy[preemptClass]
--replicas | replicaCount
--result | resultContainerMountPoint
--shell | none
--start-deadline | none
--team | none
--topology-constraint | none
--total-runtime | runPolicy[totalRuntimeSeconds]
--use-image-entrypoint | none
--waitend | none
--waitrun | none

**Example:**

Assuming the file pytorch.json is the example JSON file mentioned earlier, the following command will use instance dgx1v.16g.2.norm instead of instance dgx1v.16g.1.norm specified in the JSON.

```bash
$ ngc batch run -f pytorch.json --instance dgx1v.16g.2.norm
```

Here are some more examples of overriding JSON arguments:

```bash
$ ngc batch run -f pytorch.json --instance dgx1v.16g.4.norm --name "Jupyter Lab repro ml-model.exempt-repro"

$ ngc batch run -f pytorch.json --image nvcr.io/nvidia/pytorch:20.03-py3
```

### 10.7. Exec into a Running Job using CLI

To exec into a running container, issue the following:

```bash
$ ngc batch exec <job_id>
```
To exec a command in a running container, issue the following:

```bash
$ ngc batch exec --commandline "command" <job_id>
```

**Example using bash**

```bash
$ ngc batch exec --commandline "bash -c 'date; echo test'" <job_id>
```

### 10.8. Attaching to the Console of a Running Job

When a job is in running state, you can attach to the console of the job both from Web UI and using CLI. The console logs display outputs from both STDOUT and STDERR. These logs are also saved to the `joblog.log` file in the results mount location.

```bash
$ ngc batch attach <jobid>
```

### 10.9. Managing Jobs

This section describes various job management tasks.

#### 10.9.1. Checking Job Name, ID, Status, and Results

**Using the NGC Web UI**

Log into the NGC website, then click **JOBS** from the left-side menu.

The Jobs page lists all the jobs that you have run and shows the status, job name and ID.

The Status column reports the following progress along with timestamps: Created -> Queued -> Starting -> Running -> Finish.

When a job is in the Queued state, the Status History tab in the Web UI shows the reason for the queued state. The job info command on CLI also displays this detail.

When finished, click on your job entry from the **JOBS** page. The **Results** and **Log** tab both show the output produced by your job.

**Using the CLI**

After launching a job using the CLI, the output confirms a successful launch and shows the job details.

**Example:**

```
<table>
<thead>
<tr>
<th>Job Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id: 1854152</td>
</tr>
<tr>
<td>Name: ngc-batch-simple-job-raid-dataset-mnt</td>
</tr>
<tr>
<td>Number of Replicas: 1</td>
</tr>
</tbody>
</table>
```
The Job Status of CREATED indicates a job that was just launched.

You can monitor the status of the job by issuing:

```
$ ngc batch info <job-id>
```

This returns the same job information that is displayed after launching the job, with updated status information.

To view the stdout/stderr of a running job, issue the following:

```
$ ngc batch attach <job_id>
```

All the NGC Base Command CLI commands have additional options; issue ngc --help for details.

### 10.9.2. Monitoring Console Logs (joblog.log)

Job output (both STDOUT and STDERR) is captured in the joblog.log file.

**Using the NGC Web UI**

To view the joblog.log file, select the job from the Jobs page, then select the Log tab.

**Using the CLI**

Issue the following.

```
$ ngc result download <job-id>
```

The joblog.log file is included with the results which are downloaded to the current directory on your local disk in a folder labelled job-id.

To view the STDOUT/STDERR of a running job, issue the following:

```
$ ngc batch attach <job_id>
```

### 10.9.3. Downloading Results (interim and after completion)

**Using the NGC Web UI**

To download job results, do the following:
1. Select the job from the **Jobs** page, then select the **Results** tab.
2. From the Results page, select the file to download.

   The file is downloaded to your Download folder.

**Using the CLI**

Issue the following:

$ ngc results download <job-id>

The results are downloaded to the current directory on your local disk in a folder labelled `<job-id>`.

### 10.9.4. Terminating Jobs

**Using the NGC Web UI**

To terminate a job from the NGC website, waiting until the job appears in the Jobs page, then click the menu icon for the job and select **Kill Job**.

**Using the CLI**

Note the job ID after launching the job, then issue the following:

$ ngc batch kill <job-id>

**Example:**

$ ngc batch kill 1854178

Submitted job kill request for Job ID: '1854178'
You can also kill several jobs with one command by listing multiple job IDs as a combination of comma-separated IDs and ranges; for example ‘1-5’, ‘333’, ‘1, 2’, ‘1,10-15’.

### 10.9.5. Deleting Results

Results remain in the system consuming quota until removed:

```
$ ngc result remove <job_id>
```
Chapter 11. Telemetry

This chapter describes the system telemetry feature of Base Command Platform. In this chapter, you will learn about the different metrics collected from a workload and plotted in UI enabling you to monitor the efficiency of a workload in near real time (approximately 30 seconds). The telemetry can be accessed using both the web UI and CLI.

NVIDIA Base Command Platform provides system telemetry information for jobs and also allows jobs to send telemetry to Base Command to be recorded. This information (graphed in the Base Command dashboard and also available from the CLI in a future release) is useful for providing visibility into how jobs are running. This lets users

- Optimize jobs.
- Debug jobs.
- Analyze job efficiency.

Job telemetry is automatically generated by Base Command and provides GPU, Tensor Core, CPU, GPU Memory, and IO usage information for the job.

The following table provides a description of all the metrics that are measured and tracked in the Base Command Platform telemetry feature:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Runtime</td>
<td>How long the job has been in the RUNNING state [HH:MM:SS]</td>
</tr>
<tr>
<td>Time GPUs Active</td>
<td>The percentage of time over the entire job that the graphics engine on the GPUs have been active [GPU Active % &gt; 0%].</td>
</tr>
<tr>
<td>GPU Utilization</td>
<td>One of the primary metrics to observe. It is defined as the percentage of time one or more GPU kernels are running over the last second, which is analogous to a GPU being utilized by a job.</td>
</tr>
<tr>
<td>GPU Active %</td>
<td>Percent of GPU cores that are active. The graphics engine is active if a graphics/compute context is bound and the graphics pipe or compute pipe is busy. Effectively the GPU utilization for each GPU.</td>
</tr>
<tr>
<td>Tensor Cores Active %</td>
<td>The percentage of cycles the tensor [HMMA] pipe is active [off the peak sustained elapsed cycles].</td>
</tr>
<tr>
<td>GPU Memory Active</td>
<td>This metric represents the percentage of time that the GPU’s memory controller is utilized to either read or write from memory.</td>
</tr>
</tbody>
</table>
**Metric** | **Definition**
--- | ---
GPU Power | Shows the power used by each GPU in Watts, as well as the percentage of its total possible power draw.
GPU Memory Used (GB) | This metric shows how much of the GPU’s video memory has been used.
PCle Read/Write BW | This metric specifies the number of bytes of active PCIe read/transmit data including both header and payload. Note that this is from the perspective of the GPU, so copying data from host to device (HtoD) or device to host (DtoH) would be reflected in these metrics.
CPU Usage | This metric gives the % CPU usage over time.
System Memory | Total amount of system memory being used by the job in GB.
Raid File System | Amount of data in the /raid folder. By default the max is 2 TB. More info at [Local Scratch Space](#).
[Dataset | Workspace | Results] IOPS Read | Number of read operations per second accessing the mounted [Dataset | Workspace | Results] folders.
[Dataset | Workspace | Results] IOPS Write | Number of write operations per second accessing the mounted [Dataset | Workspace | Results] folders.
[Dataset | Workspace | Results] BW Read | Shows the total amount of data [in GB] read from the mounted [Dataset | Workspace | Results] folders.
[Dataset | Workspace | Results] BW Write | Shows the total amount of data written to the mounted [Dataset | Workspace | Results] folders.
Network BW [TX | RX] | Shows the total amount of data transmitted from the job (TX) and received by the job (RX).
NV Link BW [TX | RX] | Shows NVLink bandwidth being used in GB/s. NVLink direct is a GPU-GPU interconnect for GPUs on the same node. This is a per replica metric for Multi Node Jobs and a per node metric for partial node workloads.

### 11.1. Viewing Telemetry Information from the NGC Web UI

Click **Jobs**, select one of your jobs, then click the **Telemetry** tab.

The following are example screenshots of the Telemetry tab.

*Note: The screenshot is presented for example purpose only - the exact look may change depending on the NGC release.*
The floating window gives a breakdown of the telemetry metrics at each time slice for more informative walkthrough of the metrics. The single numbers given for attributes that are measured for each GPU is mean/average by default but we can also visualize minimum or maximum statistics using the drop down menu.

Viewing the telemetry in Min Statistics:
Viewing the telemetry in Max Statistics:

We can see the per-GPU metrics in the floating window as shown below.
The telemetry shows the Overall GPU Utilization and GPU Active Percentage along with the Job Runtime on top. Following that we have more detailed information in each section of the telemetry.

GPU Active, Tensor Cores Active, GPU Memory Active and GPU Power:
Telemetry

GPU memory Used:

PCle Read and Write BW:
NVLink BW:

CPU Usage and System Memory:
11.2. Telemetry for Multinode Jobs

By default, the telemetry shows averaged out for all the Nodes. Switching between replicas is easy by selecting which Node you want to see the metric for clicking Select Node.

The metrics then can be seen for each replica as shown below:

Replica 0:
Replica 1:
Chapter 12. Advanced Base Command Concepts

This chapter describes the more advanced features of Base Command Platform. In this chapter, you will learn about in-depth use cases of a special feature or in-depth attributes of an otherwise common feature.

12.1. Multi-node Jobs

NVIDIA Base Command Platform supports MPI-based distributed multi-node jobs in a cluster. This lets you run the same job on multiple nodes simultaneously, subject to the following requirements.

- All GPUs in a node must be used.
- Container images must include components such as OpenMPI 3.0+ and Horovod as needed.

12.1.1. Defining Multi-node Jobs

For a multi-node job, NVIDIA Base Command Platform schedules (reserves) all nodes as specified by the `--replicas` option. The specified command line in the job definition is executed only on the master node (launcher), which is identified by replica id 0. It is the responsibility of the user to execute commands on non-master nodes (replica id >0), by utilizing `mpirun` command as shown in examples in this section.

NVIDIA Base Command provides the required info, mostly exporting relevant ENV variables, to enable invocation of commands on all replicas and enable multi-node training using distributed PyTorch or Horovod.

Multi-node job command line must address the following two levels of inter-node interactions for a successful multi-node training job.

1. Invoke the command on replicas, typically all, using `mpirun`.
2. Include node details as args to distributed training scripts (such as master node address or host file).

For this need, NVIDIA Base Command sets the following variables in the job container runtime shell.
ENV Var | Definition
---|---
NGC_ARRAY_INDEX | Set to the index of the replica. Set to 0 for the master node.
NGC_ARRAY_SIZE | Set to the number of replicas in the job definition.
NGC_MASTER_ADDR | Address [DNS service] to reach Master node or Launcher. Set on all replicas. For replica 0, it points to localhost. For use with distributed training [such as PyTorch].
NGC_REPLICA_ID | Same as NGC_ARRAY_INDEX.
OMPI_MCA_orte_default_hostfile | This is only valid on Master node, or replica 0. Set to the host file location for use with distributed training [like Horovod].

12.1.2. Understanding the --replicas argument

The following table shows the corresponding node count and replica ids for the --replicas argument.

<table>
<thead>
<tr>
<th>--replicas</th>
<th>Number of nodes</th>
<th>Replica IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>--replicas 0</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>--replicas 1</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>--replicas 2</td>
<td>2 (1x master, 1x child)</td>
<td>0, 1</td>
</tr>
<tr>
<td>--replicas 3</td>
<td>3 (1x master, 2x child)</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>--replicas 4</td>
<td>4 (1x master, 3x child)</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>--replicas N</td>
<td>N (1x master, [N-1]x child)</td>
<td>0, 1, 2, ...[N-1]</td>
</tr>
</tbody>
</table>

12.1.3. Starting a Multi-node Job from the NGC Web UI

Multi-node jobs can also be started and monitored with the NGC Web UI.

Note:

In addition to conforming to the requirements of a multi-node capable container [see points under Multi-node Jobs], the container images must also be tagged as a Multi-node Container in the Web UI. This ensures the containers appear for selection when creating a multi-node job, otherwise the containers will not be available from the WebUI for multi-node jobs.

Private registry users can tag the container from the container page: Click the menu icon, select Edit, then check the Multi-node Container checkbox and save the change. Public containers that are multi-node capable must also be tagged accordingly by the publisher.
1. Login to the NGC Dashboard and select Jobs from the left-side menu.
2. In the upper right select **Create a job**.
3. Click the **Create a Multi-node Job** tab.

4. Under the **Accelerated Computing Environment** section, select your ACE and Instance type.

5. Under the **Multi-node** section, select the replica count to use.

6. Under the **Data Input** section, select the Datasets and Workspaces as needed.
7. Under the **Data Output** section, enter the result mount point.
8. Under the **Container Selection** section, select the container and tag to run, any commands to run inside the container, and an optional container port.
9. Under the **Launch Job** section, provide a name for the job and enter the total run time.
10. Click **Launch**.
12.1.4. Viewing Multi-node Job Results from the NGC Web UI

1. Click Jobs from the left-side menu.

2. Select the Job that you want to view.

3. Select one of the tabs - Overview, Telemetry, Status History, Results, or Log. The following example shows Status History. You can view the history for the overall job or for each individual replica.

12.1.5. Launching Multi-node Jobs Using the NGC CLI
Along with other arguments required for running jobs, the following are the required arguments for running multi-node jobs.

**Syntax:**

```bash
$ ngc batch run \\
  --replicas <num> \\
  --total-runtime <t> \\
  --preempt RUNONCE \\
```

**Where:**

- **--replicas** : specifies the number of nodes (including the master node) upon which to run the multi-node parallel job.
- **--total-runtime** : specifies the total time the job can run before it is gracefully shut down. Format: [nD] [nH] [nM] [nS]
- **--preempt RUNONCE** : specifies the RUNONCE job class for preemption and scheduling.

**Example 1:** for mpirun syntax.

```bash
$ ngc batch run \\
  --name "Job-nv-eagledemo-ace-399664" \\
  --preempt RUNONCE \\
  --min-timeslice 1s \\
  --total-runtime 300s \\
  --ace nv-eagledemo-ace \\
  --instance dgxa100.40g.8.norm \\
  --commandline "mpirun --allow-run-as-root -x IBV_DRIVERS=/usr/lib/libibverbs/libmlx5 -np $\{NGC_ARRAY_SIZE\} -npernode 1 bash -c 'jupyter lab --ip=0.0.0.0 --allow-root --no-browser --NotebookApp.token="" --notebook-dir=/ --NotebookApp.allow_origin="" & date; nvidia-smi;''" \\
  --result /result \\
  --array-type "MPI" \\
  --replicas "2" \\
  --image "nvidia/tensorflow:21.03-tf1-py3" \\
  --org nv-eagledemo \\
  --team nvtest-repro
```

Note that mpirun is used to execute the commands on all the replicas, specified via NGC_ARRAY_SIZE. The actual command (highlighted in a different color in the example) to run on each replica is included as a bash command input (with special chars escaped as needed).

**Example 2:** for mpirun with PyTorch.

```bash
Note the use of NGC_ARRAY_SIZE, NGC_ARRAY_INDEX, and NGC_MASTER_ADDR.
$ ngc batch run \\
  ..
  --commandline "mpirun --allow-run-as-root -x IBV_DRIVERS=/usr/lib/libibverbs/libmlx5 -np $\{NGC_ARRAY_SIZE\} -npernode 1 bash -c 'jupyter lab --ip=0.0.0.0 --allow-root --no-browser --NotebookApp.token="" --notebook-dir=/ --NotebookApp.allow_origin="" & date; python3 -m torch.distributed.launch --nproc_per_node=8 --nnodes=$\{NGC_ARRAY_SIZE\} --node_rank=$\{NGC_ARRAY_INDEX\} --master_addr=$\{NGC_MASTER_ADDR\} train.py'" \\
  --result /result \\
  --array-type "MPI" \\
  --replicas "2" \\
  --image "nvidia/tensorflow:21.03-tf1-py3" \\
  ..
```
Targeting Commands to a Specific Replica

CLI can be used to execute a command in a running job container by using the following command.

```bash
$ ngc batch exec <job_id>
```

For a multinode workload, there are multiple replicas running containers. The replicas are numbered with 0-based indexing. The above command, specifying just the job id, targets the exec command to the first replica, which is indexed at 0. You may need to run a command on a different replica in a multinode workload, which can be achieved by the following option.

```bash
$ ngc batch exec <job_id>:<replica-id>
```

When omitted, the first replica (id 0) is targeted for the command.

Viewing Multi-node Job Status and Information

The status of the overall job can be checked with the following command:

```bash
$ ngc batch info <job_id>
```

To check the status of one of the replicas, issue:

```bash
$ ngc batch info <job_id>:<replica_id>
```

Where `<replica_id>` is from 0 to `{number of replicas}-1`.

Example showing the status of each replica of a two-replica job:

```bash
$ ngc batch info 1070707:0
```

```
Replica Information
Replica: 1070707:0
Created At: 2020-03-04 22:39:00 UTC
Submitted By: John Smith
Team: swngc-mnpilot
Replica Status
Status: CREATED
```

```bash
$ ngc batch info 1070707:1
```

```
Replica Information
Replica: 1070707:1
Created At: 2020-03-04 22:39:00 UTC
Submitted By: John Smith
Team: swngc-mnpilot
Replica Status
Status: CREATED
```

12.1.6. Launching Multi-node Jobs with bcprun

NGC now includes `bcprun`, a utility for running multi-node jobs.

Syntax:

```bash
$ bcprun --cmd '<command-line>'
```

Where:

- `<command-line>` is the command to run
Example:

```bash
$ bcprun --cmd 'python train.py'
```

## Optional Arguments

<table>
<thead>
<tr>
<th>Optional Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n, --nodes &lt;n&gt;</td>
<td>Number of nodes to run on. (type: integer)</td>
</tr>
<tr>
<td></td>
<td>Range: min value: 1, max value: R, where R is the max number of replicas requested by the NGC job.</td>
</tr>
<tr>
<td></td>
<td>Default value: R</td>
</tr>
<tr>
<td>-p, --pernode &lt;p&gt;</td>
<td>Number of tasks per node to run. (type: integer)</td>
</tr>
<tr>
<td></td>
<td>Range: min value: 1, max value: (none)</td>
</tr>
<tr>
<td></td>
<td>Default value: environment variable NGC_NTASKS_PER_NODE, if set; otherwise 1.</td>
</tr>
<tr>
<td>-e, --env &lt;e&gt;</td>
<td>Environment variables to set with format 'key=value'. (type: string)</td>
</tr>
<tr>
<td></td>
<td>Each variable assignment requires a separate -e/--env flag.</td>
</tr>
<tr>
<td></td>
<td>Default value: (none)</td>
</tr>
<tr>
<td>-w, --workdir &lt;w&gt;</td>
<td>Base directory from which to run &lt;cmd&gt;. (type: string)</td>
</tr>
<tr>
<td></td>
<td>May include environment variables defined with --env.</td>
</tr>
<tr>
<td></td>
<td>Default value: environment variable PWD (current working directory)</td>
</tr>
<tr>
<td>-l, --launcher &lt;l&gt;</td>
<td>Run &lt;cmd&gt; using an external launcher program. (type: string)</td>
</tr>
<tr>
<td></td>
<td>Supported launchers: mpirun, horovodrun</td>
</tr>
<tr>
<td></td>
<td>- mpirun: maps to OpenMPI options (<a href="https://www.open-mpi.org/">https://www.open-mpi.org/</a>)</td>
</tr>
</tbody>
</table>
### Advanced Base Command Concepts

- horovodrun: maps to Horovod options
  
  [https://horovod.ai/]

**Note:** This option assumes the launcher exists and is in PATH.

Launcher-specific arguments (not part of bcprun options) can be provided as a suffix.

**Example:**

```bash
--launcher 'mpirun --allow-run-as-root'
```

Default value: [none]

---

**-a, --async**

Run with asynchronous failure support enabled, i.e. a child process of bcprun can exit on failure without halting the program.

The program will continue while at least one child is running.

The default semantics of bcprun is to halt the program when any child process launched by bcprun exits with error.

---

**-d, --debug**

Print debug info and enable verbose mode.

---

**-v, --version**

Print version info.

---

**-h, --help**

Print this help message

---

## 12.2. Job ENTRYPOINT

NGC Base Command CLI now provides the option of incorporating Docker ENTRYPOINT when running jobs.

Some NVIDIA deep learning framework containers rely on ENTRYPOINT to be called for full functionality. The following functions in these containers rely on ENTRYPOINT:

- Version banner to be printed to logs
- Warnings/errors if any platform prerequisites are missing
- MPI set up for multi-node

The following is an example of the version header information that is returned after running a TensorFlow container with the incorporated ENTRYPOINT using the docker run command:

```bash
$ docker run --runtime=nvidia --rm -it nvcr.io/nvidia/tensorflow:21.03-tf1
  nvidia-smi
```

```
== TensorFlow ==
```

```
NVIDIA Release 21.03-tf1 (build 20726338)
TensorFlow Version 1.15.5
Container image Copyright (c) 2021, NVIDIA CORPORATION. All rights reserved.
```
Without using ENTRYPOINT in the CLI, there would be no banner information in the output.

This is shown in the following example of using NGC Base Command CLI to run nvidia-smi within the TensorFlow container without using ENTRYPOINT.

```
$ ngc batch run \
--name "TensorFlow Demo" \
--preempt RUNONCE \
--min-timeslice 0s \
--total-runtime 0s \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.1.norm \
--result /result \
--image "nvidia/tensorflow:21.03-tf1-py3" \
--commandline "nvidia-smi"
```

Initial lines of the output Log File (no TensorFlow header information is generated):

```
Thu Apr 15 17:32:02 2021
+-------------------------------------------------------------------+
| NVIDIA-SMI 450.80.02 Driver Version: 450.80.02 CUDA Version: 11.2 |
|---------------------+----------------------+----------------------|
|---------------------+-----------------------|
|---------------------+-----------------------|
|---------------------+-----------------------|
+-------------------------------------------------------------------+
```

### 12.2.1. Example Using Container ENTRYPOINT

To use the container ENTRYPOINT, use the --use-image-entrypoint argument.

**Example:**

```
$ ngc batch run \
--name "TensorFlow Entrypoint Demo" \
--preempt RUNONCE \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.1.norm \
--result /result \
--image "nvidia/tensorflow:21.03-tf1-py3" \
--use-image-entrypoint \
--commandline "nvidia-smi"
```

Output log file with TensorFlow header information, including initial lines of the nvidia-smi output.

```
== TensorFlow ==
NVIDIA Release 21.03-tf1 (build 20726338)
TensorFlow Version 1.15.5
Container image Copyright (c) 2021, NVIDIA CORPORATION. All rights reserved.
Copyright 2017-2021 The TensorFlow Authors. All rights reserved.
NVIDIA Deep Learning Profiler (dlprof) Copyright (c) 2021, NVIDIA CORPORATION.
```
12.2.2. Example Using CLI ENTRYPOINT

You can also use the `--entrypoint` argument to specify an ENTRYPOINT to use that will override the container ENTRYPOINT.

The following is an example of specifying an ENTRYPOINT in the NGC Batch command to run nvidia-smi. This is instead of using the `--commandline` argument.

```
$ ngc batch run \
--name "TensorFlow CLI Entrypoint Demo" \
--preempt RUNONCE \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.1.norm \
--result /result \
--image "nvidia/tensorflow:21.03-tf1-py3" \
--entrypoint "nvidia-smi"
```

Initial lines of the output file.

```
Thu Apr 15 17:52:53 2021
+-----------------------------------------------------------------------------+    
| NVIDIA-SMI 450.80.02 Driver Version: 450.80.02 CUDA Version: 11.0             |    
|--------------------------------------------------------------------------------+    
|                                                                                |    
|                                                                                |    
```

Chapter 13. Tutorials

This chapter describes the tutorials that showcase various features of Base Command Platform. In this chapter, you will learn about ready-to-run tutorials available within the product for learning a workflow or for use as a basis for your custom workflow. This section also covers tutorials with sample commands or templates which can serve as a starting point for new users or new complex workflows.

Note: The ready-to-run tutorials are delivered as templates in nvbc-tutorials team context along with the required container images and data entities. Your org admin must add you to that team explicitly for you to be able to access these templates and run workloads based on those.

13.1. Launching a Job from Existing Templates

1. Click BASE COMMAND > Jobs the left-side menu and then click Create Job.
2. Click Templates from the ribbon menu.
3. Click the menu icon for the template to use, then select **Apply Template**.

The create a job page opens with the fields populated with the information from the job template.

4. Verify the pre-filled fields, enter a unique name, then click **Launch**.

---

**13.2. Launching an Interactive Job with JupyterLab**

From the existing templates, you can run the `nvbc-jupyterlab` template to pre-fill the job creation fields and launch an Interactive Job with jupyterLab. The following is an example of the CLI script for the same job script template.

```
$ ngc batch run \
   --name "NVbc-jupyterlab" \
   --preempt RUNONCE \
```
13.3. Launching a Multi Node Interactive Job with JupyterLab

From the existing templates, you can run the `nvbc-jupyterlab-mn` template to pre-fill the job creation fields and launch an multinode Interactive Job with 2 nodes. The following is an example of the CLI script for the same job script template.

```bash
$ ngc batch run \
--name "nvbc-jupyterlab-mn" \
--preempt RUNONCE \
--min-timeslice 0s \
--total-runtime 36000s \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.8.norm \
--commandline "mpirun --allow-run-as-root -np 2 -npernode 1 bash -c ' set -x; jupyter lab --NotebookApp.token='' --notebook-dir=/
                                        --NotebookApp.allow_origin='' & date; nvidia-smi; echo ; sleep 1d'' \
--result /result \
--array-type "MPI" \
--replicas "2" \
--image "nvidia/pytorch:21.02-py3" \
--org nv-eagledemo \
--team nvbc-tutorials \
--port 8888
```

13.4. Getting Started with Tensorboard

Tensorboard is already installed by default on standard NGC containers. Perform the following to get started using TensorBoard

1. Start a TensorFlow job.
   
The following is an example using the NGC CLI.

```bash
$ ngc batch run \
--name "NVbc-tensorboard" \
--preempt RUNONCE \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.1.norm \
--commandline "set -x; jupyter lab --allow-root --NotebookApp.token='' --notebook-dir=/
                                        --NotebookApp.allow_origin='' & date; nvidia-smi; echo ; sleep 1d" \
--result /result \
--image "nvidia/tensorflow:21.08-tf1-py3" \
--org nv-eagledemo \
```

---

Tutorials

NVIDIA Base Command Platform

DU-10422-001_v03 | 96
2. Login to the container via JupyterLab and open a terminal.
3. Download the TensorBoard tutorial notebook.
   
   ```bash
   wget https://storage.googleapis.com/tensorflow_docs/tensorboard/docs/get_started.ipynb
   ```
4. Open the downloaded notebook.
5. Run the commands in the notebook until you get to command 6.
   
   ```bash
   tensorboard --logdir logs/fit
   ```
6. Open the URL mapped to port 6006 on the container to open Tensorboard.

   The TensorBoard UI should appear similar to the following example.
13.5. NCCL Tests

NCCL tests check both the performance and the correctness of NCCL operations and you can test out the performance between GPUs using the `nvbc-MN-NCCL-Tests` template. The following is an example of the CLI script for the same NCCL Test template. The Average Bus Bandwidth for a successful NCCL test is expected to be > 175GB.

```bash
$ ngc batch run \
--name "nvbc-MN-NCCL-Tests" \
--preempt RUNONCE \
--total-runtime 86400s \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.1.norm \
--commandline "bash -c 'for i in {1..20}; do echo "******************** Run ********************"; mpirun -np $(NGC_ARRAY_SIZE) -npernode 1 /nccl-tests/build/all_reduce_perf -b 128M -e 2G -f 2 -t 8 -g 1; done'" \
--result /result \
--array-type "MPI" \
--replicas "2" \
--image "nv-eagledemo/mn-nccl-test:ibeagle" \
--org nv-eagledemo \
--team nvbc-tutorials
```

13.6. StyleGAN SingleNode Workload

From the existing templates, you can run the `nvbc-stylegan-singlenode` template to pre-fill the job creation fields and launch. The following is an example of the CLI script for StyleGAN single node workload with 8GPUs.

```bash
$ ngc batch run \
--name "StyleGAN-singlenode" \
--preempt RUNONCE \
--min-timeslice 0s \
--ace nv-eagledemo-ace \
--instance dgxa100.40g.8.norm \
--commandline "python -u -m torch.distributed.launch --nproc_per_node=8 /mnt/workspace/train.py --snap=25 --data=/dataset --batch-size=32 --lr=0.002" \
--result /output \
--image "nv-eagledemo/nvbc-tutorials/pytorch_stylegan:v1" \
--org nv-eagledemo \
--team nvbc-tutorials \
--datasetid 76731:/dataset
```

Here's an example of the telemetry once the job is launched.
13.7. **StyleGAN MultiNode Workload**

From the existing templates, you can run the `nvbc-stylegan-multinode` template to pre-fill the job creation fields and launch. The following is an example of the CLI script for the multi-node StyleGAN workload with 2 Nodes.

```
$ ngc batch run \
--name "StyleGAN-multinode" \ 
--preempt RUNONCE \ 
--min-timeslice 0s \ 
--total-runtime 230400s \ 
--ace nv-eagledemo-ace \ 
--instance dgxa100.40g.8.norm \ 
--commandline "mpirun --allow-run-as-root -np 2 -npernode 1 bash -c 'python 
-u -m torch.distributed.launch --nproc_per_node=8
--master_addr=${NGC_MASTER_ADDR} --nnodes=${NGC_ARRAY_SIZE}
--node_rank=${NGC_ARRAY_INDEX} /mnt/workspace/train.py --snap=25
--data=/dataset
--result /output
--array-type "MPI"
--replicas "2" 
--image "nv-eagledemo/nvbc-tutorials/pytorch_stylegan3:pytorch.stylegan.v1"
--org nv-eagledemo \ 
--team nvbc-tutorials \ 
--datasetid 76731:/dataset"

Here's an example of the telemetry once the job is launched.
13.8. Building a Dataset from S3 Cloud Storage

This section details an example of building a dataset with CLI and code from a cloud storage bucket.

Perform the following before starting.

1. Identify credentials and location of the cloud storage bucket.
2. Know the directory structure within the bucket.
3. Create a workspace in Base Command Platform (typically dedicated as home workspace).
   Refer to Creating a Workspace Using the Base Command CLI for instructions.
4. Have a current job running to exec into or from which to run the following example.

13.8.1. Running a Job

1. Start a Jupyter notebook job.
   Replace ACE, org, workspace, and team values arguments. The job will run for one hour.
   ```shell
   ngc batch run --name "demo-s3-cli" --preempt RUNONCE --ace {ace-name} \  
   --instance {instance-type} --commandline "jupyter lab --ip=0.0.0.0 --allow-root \  
   --no-browser --NotebookApp.token='' \  
   --notebook-dir=/ --NotebookApp.allow_origin='*' & date; sleep 1h" --result /  
   --workspace {workspace-name}:{workspace-name}:RW --image "nvidia/pytorch:21.07-py3" \  
   --org {org-name} --team {team-name} --port 8888
   ```
2. Once the job has started, access the JupyterLab terminal.
   ```shell
   ngc batch info {id}
   ```
13.8.2.  Creating a Dataset using AWS CLI

1. Obtain, unzip, and install the AWS CLI zip file.
   ```
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
   unzip awscliv2.zip
   ./aws/install -i /usr/local/aws-cli -b /usr/local/bin
   ```

2. Ensure there is access to the AWS CLI.
   ```
   aws --version
   ```

3. Run through the AWS Configuration by inputting the Access Key ID and Secret Access Key.
   These can be found underneath AWS’s IAM user panel. Refer to additional AWS CLI documentation.
   ```
   aws configure
   AWS Access Key ID [None]: <ACCESS_KEY>
   AWS Secret Access Key [None]: <SECRET_ACCESS_KEY>
   ```
Default region name [None]: us-west-2
Default output format [None]: json

4. Sync a bucket to the results folder to be saved as a dataset.

```
aws s3 sync 's3://<source-bucket>' '../results'
```

Results should now be ready to be saved as a dataset. Refer to Managing Datasets for more information.

### 13.8.3. Creating a Dataset using AWS Boto3

Boto3 is the AWS SDK for accessing S3 buckets. This section will cover downloading a specific file from an S3 bucket and then saving it to a results folder. View more documentation regarding Boto3 here.

1. Install Boto3 through pip and prepare imports in the first cell of the Jupyter notebook.

```
!pip install boto3
import boto3
import io
import os
```

2. Initialize Boto3 with an AWS Access Key and Secret Access Key.

Make sure IAM user settings has proper access and permissions to the needed S3 buckets.

```
# Let's use Amazon S3 by initializing our Access Key and Secret Access Key
s3 = boto3.resource('s3', aws_access_key_id=<ACCESS_KEY>,
                    aws_secret_access_key=<SECRET_ACCESS_KEY>)
bucket = s3.Bucket(<BUCKET_NAME>)
```

### 13.8.4. Downloading a File

Downloading a file is a function built within Boto3. It will need the Bucket Name, Object Name (referred to as a key), and the File Output Name. Refer to Amazon S3 Examples - Downloading files for additional information.

```
s3.download_file(<BUCKET_NAME>, <OBJECT_NAME>, <FILE_NAME>)
```

### 13.8.5. Downloading a Folder

The following includes a function for downloading a single-directory depth from an S3 bucket to BCP storage, either to /results mount of the job or to a Base Command Platform workspace mounted in the job.

```
def download_s3_folder(s3_folder, local_dir='..//results/s3_bucket'):
    for obj in bucket.objects.filter(Prefix=s3_folder):
        target = obj.key if local_dir is None \
            else os.path.join(local_dir, os.path.repath(obj.key, s3_folder))
        if not os.path.exists(os.path.dirname(target)):
            os.makedirs(os.path.dirname(target))
        if obj.key[-1] == '/':
            continue
        print(obj.key)
        bucket.download_file(obj.key, target)
```
To save a dataset or checkpoint from the /results mount, download the contents and then upload as a dataset as described in Converting a Checkpoint to a Dataset.

13.9. Using Data Loader for Cloud Storage

This document details an example of using a data loader from a cloud storage bucket. It is recommended that the CLI option is attempted before proceeding with the data loader as it will not save the folder hierarchy.

Perform the following before starting.

1. Identify credentials and location of the cloud storage bucket.
2. Know the directory structure within the bucket.
3. Create a workspace in Base Command Platform (typically dedicated as home workspace).
   Refer to Creating a Workspace Using the Base Command CLI for instructions.

13.9.1. Running and Opening JupyterLab

1. Mount the workspace in the job.
2. Replace ACE, org, workspace, and team arguments.
   
   ngc batch run --name "demo-s3-dataloader" --preempt RUNONCE --ace {ace-name}  
   --instance {instance-type} --commandline "jupyter lab --ip=0.0.0.0 \ 
   --allow-root --no-browser --NotebookApp.token='' --notebook-dir=/ \ 
   --NotebookApp.allow_origin='*' & date; sleep 6h" --result /results \ 
   --workspace {workspace-name}:/mount/{workspace-name}:RW --image "nvidia/ 
   pytorch:21.07-py3" \ 
   --org {org-name} --team {team-name} --port 8888

3. Open the link for the JupyterLab to access the UI.
   Do this by fetching the job's information with the batch info command. Below is an example response with the mapped port. You can ctrl+left click the link in bold to access it in your browser.

   ngc batch info {id}

   Job Information
   Id: 2233490
   ...
   Job Container Information
   Docker Image URL: nvidia/pytorch:21.07-py3
   Port Mappings
   Container port: 8888 mapped to https://tnmy3490.eagle-demo.proxy.ace.ngc.nvidia.com
   ...
   Job Status
   ...
   Status: RUNNING
   Status Type: OK
   ----------------------------------------

   You should now be prompted with options to create a file.

4. Navigate into your workspace on the sidebar, and then click on Python 3 to create your file.
13.9.2. Utilizing the Cloud Data Loader for Training

Use the code for creating a Jupyter Notebook, with these changes:

1. Do not issue `import wandb`.
2. Add the following imports:

```python
# Imports
!pip install boto3
import boto3
from botocore import UNSIGNED
from botocore.config import Config
```
3. Change the first line of #3.2.

From this:

```python
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

To this:

```python
s3 = boto3.client('s3', config=Config(signature_version=UNSIGNED))
bucket_name='mnist-testbucket'
key='mnist_2.npz'

s3_response_object = s3.get_object(Bucket=bucket_name, Key=key)
object_content = s3_response_object['Body'].read()
load_bytes = BytesIO(object_content)

with np.load(load_bytes, allow_pickle=True) as f:
    x_train, y_train = f['x_train'], f['y_train']
    x_test, y_test = f['x_test'], f['y_test']
```

4. Execute Step #3 through Step #6.
Chapter 14. Using NVIDIA Base Command Platform with Weights & Biases

14.1. Introduction

NVIDIA Base Command™ Platform is a premium infrastructure solution for businesses and their data scientists who need a world-class artificial intelligence (AI) development experience without the struggle of building it themselves. Base Command Platform provides a cloud-hosted AI environment with a fully managed infrastructure.

In collaboration with Weights & Biases (W&B), Base Command Platform users now have access to the W&B machine learning (ML) platform to quickly track experiments, version and iterate on datasets, evaluate model performance, reproduce models, visualize results, and spot regressions, and share findings with colleagues.

This guide explains how to get started with both Base Command Platform and W&B, as well as walks through a quick tutorial with an exemplary deep learning (DL) workflow on both platforms.

14.2. Setup

14.2.1. Base Command Platform Setup

1. Set up a Base Command Platform account.
   - Ask your team admin to add you to the team or org you want to join. After being added, you will receive an email invitation to join Base Command Platform. Follow the instructions in the email invite to set up your account. Refer to the section Onboarding and Signup for more information on setting the context and configuring your environment.

2. While loggin in to the web UI, install and setup the CLI.
   - Follow instructions at https://ngc.nvidia.com/setup/installers/cli. The CLI is supported for Linux, Windows, and MacOS.
3. Generate an API key.
   Once logged into Base Command Platform, go to the API key page and select “Generate API Key”. Store this key in a secure place. The API key will also be used to configure the CLI to authenticate your access to Base Command Platform.

4. Set the NGC context.
   Use the CLI to log in and enter your API key and setting preferences. The key will be stored for future commands.

   ngc config set

   You will be prompted to enter your API key and then your context, which is your org/team (if teams are used), and the ace. Your context in NGC defines the default scope you operate in for collaboration with your team members and org.

14.2.2. Weights and Biases Setup

   Your Base Command Platform subscription automatically provides you with access to the W&B Advanced version. Create and set up credentials for your W&B account as your Base Command Platform account is not directly integrated with W&B – that is, W&B cannot be accessed with your Base Command Platform credentials.

2. Create a private workspace on Base Command Platform.
   Using a private workspace is a convenient option to store your config files or keys so that you can access those in read-only mode from all your Base Command Platform workloads. TIP: Name the workspace “homews-<accountname>” for consistency. Set your ACE and org name – here, “nv_eagledemo-ace” and “nv-eagledemo”.

   ngc workspace create --name homews-<accountname> --ace nv-eagledemo-ace --org nv-eagledemo

3. Access your W&B API key.
   Once the account has been created, you can access your W&B API key via your name icon on the top of the page → “Settings” → “API keys”. Refer to the “Execution” section for additional details on storing and using the W&B API key in your runs.

14.2.3. Storing W&B Keys in Base Command Platform

Your workload running on Base Command Platform must specify the credentials and configuration for your W&B account, for tracking jobs and experiments. Saving the W&B key in a Base Command Platform workspace needs to be performed only one time. The home workspace can be mounted to any Base Command Platform workload to access the previously recorded W&B key. This section shows how to generate and save W&B API key to your workspace.

Users have two options to configure the W&B API key to the private home workspace.

14.2.3.1. Option 1 | Using a Jupyter Notebook
1. Run an interactive JupyterLab job on Base Command Platform with the workspace mounted into the job.

In our example, we use homews-demouser as workspace. Make sure to replace the workspace name and context accordingly for your own use.

CLI:

```
ngc batch run --name 'wandb_config' --ace nv-eagledemo-ace --instance dgxa100.40g.1.norm --commandline "pip install wandb; jupyter lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir='' --result /results --image "nvidia/tensorflow:21.06-tf2-py3" --org nv-eagledemo --team nvtest-demo --workspace homews-demouser:/homews-demouser:RW --port 8888
```

Note that the home workspace (here, homews-demouser) is mounted in read / write mode.

2. When the job is running, start a session by clicking on the JupyterLab URL (as displayed on the “Overview” tab within a job).

3. Create new Jupyter notebook (e.g., “config”) and copy the following script into the notebook.

```python
import wandb
import os
import requests

# 1. Login to W&B interactively to specify the API key
wandb.login()

# 2. Create a directory for configuration files
!mkdir -p /homews-demouser/bcpwandb/wandbconf

# 3. Copy the file into the configuration folder
!cp ~/.netrc /homews-demouser/bcpwandb/wandbconf/config.netrc

# 4. Set the login key to the stored W&B API key
os.environ["NETRC"] = ""/homews-demouser/bcpwandb/wandbconf/config.netrc"

# 5. Check current W&B login status and username. Validate the correct API key
res = requests.post("https://api.wandb.ai/graphql", json={"query": "query Viewer { viewer { username email } }", auth=("api", wandb.api.api_key)})
res.json()["data"]['viewer']
```

The W&B API key is now stored in the home workspace (homews-demouser).

### 14.2.3.2. Option 2 | Using a Script (via curl Command)

1. Run an interactive JupyterLab job on Base Command Platform with the workspace mounted into the job.

In our example, we use homews-demouser as workspace. Make sure to replace the workspace name and context accordingly for your own use.

CLI:

```
ngc batch run --name 'wandb_config' --ace nv-eagledemo-ace --instance dgxa100.40g.1.norm --commandline "pip install wandb; jupyter lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir='' --result /results --image "nvidia/tensorflow:21.06-tf2-py3" --org nv-eagledemo --team nvtest-demo --workspace homews-demouser:/homews-demouser:RW --port 8888
```

NVIDIA Base Command Platform
OK< so replaceNote that the home workspace (here, homews-demouser) is mounted in read / write mode.

2. When the job is running, start a session by clicking on the JupyterLab URL (as displayed on the “Overview” tab within a job).

3. Start a terminal in JupyterLab and execute the following commands to create user credentials.

   Make sure to replace the workspace name and context accordingly for your own use.

   **Terminal:**

   ```
   $ pip install wandb
   $ curl -sL https://wandb.me/bcp_login | python - config <API key>
   $ mkdir -p /homews-demouser/bcpwandb/wandbconf
   $ cp config.netrc /homews-demouser/bcpwandb/wandbconf/config.netrc
   $ python -c "os.environ["NETRC"]= "/homews-demouser/bcpwandb/wandbconf/config.netrc"
   ``

   **Terminal output:** ‘API key written to config.netrc, use by specifying the path to this file in the NETRC environment variable’.

   This command will create a configuration directory in your home workspace and store the W&B API key in this workspace (homews-demouser) via a configuration file.

### 14.3. Using W&B with a JupyterLab Workload

After having followed the previous steps, the W&B API key is securely stored in a configuration file within your private workspace (here, homews-demouser). Now, this private workspace must be attached to a Base Command Platform workload to use the W&B account and features.

In the section below, you will create a JupyterLab notebook as an example to show the stored API key. MNIST handwritten digits classification using a Convolutional Neural Network with TensorFlow and Keras is an easily accessible, open-source model and dataset that we will use for this workflow (available via Keras [here](#)).

#### 14.3.1. Create a Jupyter Notebook, Including W&B Keys for Experiment Tracking

Follow the first two steps in either option under Storing W&B Keys in Base Command Platform to create a job on Base Command Platform. After having accessed JupyterLab via the URL, start a new Jupyter notebook with the code below and save it as a file in your private workspace (/homews-demouser/bcpwandb/MNIST_example.ipynb).

The following exemplary script imports required packages, sets the environment, and initializes a new W&B run. Subsequently, it builds, trains, and evaluates the Convnet model with TensorFlow and Keras, as well as tracks several metrics with W&B.

```python
# Imports
!pip install tensorflow
import numpy as np
```
from tensorflow import keras
from tensorflow.keras import layers
import wandb
import os

# 1. Import the W&B API key from private config workspace by defining NETRC
fileos.environ["NETRC"] = "/homews-demouser/bcpwandb/wandbconf/config.netrc"

# 2. Initialize the W&B run
wandb.init(project = "nvtest-repro", id = "MNIST_run_epoch-128_bs-15", name = "NGC-JOB-ID_" + os.environ["NGC_JOB_ID"]

# 3. Prepare the data
# 3.1 Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# 3.2 Split data between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

# 3.3 Make sure images have the shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# 3.4 Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# 4. Build the model
model = keras.Sequential(
    [keras.Input(shape=input_shape), layers.Conv2D(32, kernel_size=(3, 3), activation="relu"), layers.MaxPooling2D(pool_size=(2, 2)), layers.Conv2D(64, kernel_size=(3, 3), activation="relu"), layers.MaxPooling2D(pool_size=(2, 2)), layers.Flatten(), layers.Dropout(0.5), layers.Dense(num_classes, activation="softmax"),]
)
model.summary()

# 5. Train the model
batch_size = 128
epochs = 15
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)

# 6. Evaluate the trained model
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

# 7. Track metrics with wandb
wandb.log({'loss': score[0], 'accuracy': score[1]})

# 8. Track training configuration with wandb
wandb.config.batch_size = batch_size
wandb.config.epochs = epochs

After this step, your home workspace (homews-demouser) will include the configuration file and the exemplary Jupyter notebook created above.

- Home workspace: /homews-demouser
- Configuration file: /homews-demouser/bcpwandb/wandbconf/config.netrc
- Jupyter notebook: /homews-demouser/bcpwandb/MNIST_example.ipynb

14.3.2. Running a W&B Experiment in Batch Mode

After having successfully completed all steps, including 4.1., proceed to run a W&B experiment in batch mode. Make sure to replace the workspace name and context accordingly for your own use.

Run Command:

ngc batch run --name "MNIST_example_batch" --ace nv-eagledemo-ace --instance dgxa100.40g.1.norm --commandline "pip install wandb; jupyter lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token='' --NotebookApp.allow_origin='*' --notebook-dir=/ & date; cp /homews-demouser/bcpwandb/MNIST_example.ipynb /results; touch /results/nb-executing; jupyter nbconvert --execute --to=notebook --inplace -y --no-prompt --allow-errors --ExecutePreprocessor.timeout=-1 /results/MNIST_example.ipynb; sleep 2h" --result /results --image "nvidia/tensorflow:21.06-tf2-py3" --org nv-eagledemo --team nvtest-demo --workspace homews-demouser:/homews-demouser:RO --port 8888

- pip install wandb ensures that the wandb package is installed before the job is launched.
- The last section of the code automatically executes the Jupyter notebook without the need to re-run it manually after each job launch. At the bottom of the screen, we enter a name for the job following the convention detailed here and click "Launch".

After completion of the job, the results can be accessed on the W&B dashboard which provides an overview of all projects of a given user (here, nv-testuser). Within a W&B project, users can compare the tracked metrics (here, accuracy and loss) between different runs.
14.4. Using Logged Dataset and Model Artifacts

The following example below utilizes the “Pipeline Versioning with W&B Artifacts” tutorial from the official W&B Documentation. A similar notebook with additional documentation can be found here.

1. Launch a JupyterLab job.

   Use the public Tensorflow container provided by NVIDIA, replacing the following arguments: ace-name, instance-type, org-name, and team-name.

   ```
   ngc batch run --name "jupyterlab" --preempt RUNONCE --min-timeslice 0s --total-runtime 0s --ace {ace-name} --instance {instance-type} --commandline "jupyter lab --ip=0.0.0.0 --allow-root --no-browser --NotebookApp.token='' --notebook-dir=/ --NotebookApp.allow_origin='*' & date; sleep 1h" --result /result --image "nvidia/tensorflow:21.07-tf2-py3" --org {org-name} --team {team-name} --port 8888
   ```

2. After JupyterLab is running, pull up the mapped port using the following command:

   ```
   ngc batch info {job-id}
   ```

3. Download and import libraries.

   ```
   !pip install wandb
   !pip install boto3
   !pip install pillow
   !pip install -U numpy==1.19.5

   # AWS imports
   import boto3
   from botocore import UNSIGNED
   from botocore.config import Config

   # Numpy imports for MNIST example
   from datetime import datetime
   import random
   import numpy as np
   from numpy import asarray

   # Tensorflow imports
   import tensorflow as tf
   from tensorflow import keras
   from tensorflow.keras import layers

   # Wandb imports and file libraries
   import wandb
   import os
   import requests
   ```
import shutil from collections import namedtuple

4. Download the MNIST dataset from S3.

This section demonstrates how to use the Boto3 SDK from AWS to download a file from an S3 Bucket. The cell below contains a public dataset with no Access Key or Secret Access Key configured.

```python
# Let's use Amazon S3 by initializing the client
s3 = boto3.client('s3', config=Config(signature_version=UNSIGNED))
bucket_name='mnist-testbucket'
key='mnist_2.npz'

NGC_JOB_NAME = input("Provide a name for your job here:")

# This obtains a single file from S3 through the 'Key', it then saves it to the results
# folder as a compressed numpy array file to be later stored as a dataset in BCP
# s3_response_object = s3.get_object(Bucket=bucket_name, Key=key)
# In order to use the s3 bucket as a dataset, you will need to copy it to your results folder
s3.download_file(bucket_name, key, '/result/mnist.npz')
```

5. Enter the following in a separate JupyterLab notebook cell.

```python
# Login to W&B interactively to specify the API key.
wandb.login()

# Create a directory for configuration files.
!mkdir -p /homews-demouser/bcpwandb/wandbconf

# Copy the file into the configuration folder.
!cp ~/.netrc /homews-demouser/bcpwandb/wandbconf/config.netrc

# Set the login key to the stored W&B API key.
os.environ['NETRC']="/homews-demouser/bcpwandb/wandbconf/config.netrc"

# Check current W&B login status and username. Validate the correct API key
# The command will output {"email": "xxx@wandb.com", "username": "xxxx"}
res = requests.post("https://api.wandb.ai/graphql", json={"query": "query Viewer
{ viewer { username email } }"}, auth=('api', wandb.api.api_key)) res.json()['data']['viewer']
```

6. Follow the instructions in the Pipeline Versioning with W&B Artifacts tutorial and make the following changes.

a). Ignore the import instructions and begin using the code at the #Set the random seeds line.

b). In the def log_and_log(steps) function, substitute NVIDIA code.

```
with wandb.init(project="artifacts-example", job_type="load-data") as run:
```

```
To this:

with wandb.init(project="nvtest-repro",
tags=[NGC_JOB_NAME, "MNIST_run_epoch-128 bs-15"],
name="mnist-raw-{tstamp}".format(tstamp)) as run:
```

c). In the def preprocess_and_log(steps) function, substitute NVIDIA code.

```
with wandb.init(project="artifacts-example", job_type="preprocess-data") as run:
```

```
To this:

tstamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
with wandb.init(project="nvtest-repro")
```
Using NVIDIA Base Command Platform with Weights & Biases

```python
tag=[NGC_JOB_NAME, "MNIST_run_epoch-128_bs-15"],
name="mnist-preprocess-{}``.format(tstamp)) as run:

d). In the def train_and_log(config) function, substitute NVIDIA code.

Change this:
```python
with wandb.init(project="artifacts-example", job_type="train", config=config) as run:
```

To this:
```python
tstamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
with wandb.init(project="artifacts-example", job_type="train",
tag=[NGC_JOB_NAME],
name="train-model-{}".format(tstamp),
config=config) as run:
```

e). In the def evaluate_and_log(config=None) function, substitute NVIDIA code.

Change this:
```python
with wandb.init(project="artifacts-example", job_type="report", config=config) as run:
```

To this:
```python
tstamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
with wandb.init(project="artifacts-example", job_type="report",
tag=[NGC_JOB_NAME],
name="report-{}".format(tstamp),
config=config) as run:
```

14.5. Best Practices for Running Multiple Jobs Within the Same Project

W&B only recognizes a new run upon a change in the run ID within the wandb.init( ) command. When only changing the run name, W&B will simply override the already existing run that has the same run ID. Alternatively, to log and track a new run separately, users can keep the same run ID but need to define the new run within a new project.

Runs can be customized within the wandb.init( ) command as follows:

```python
wandb.init(project = "nvtest-demo", id = "MNIST_run_epoch-128_bs-15", name = "NGC-JOB-ID_" + os.environ["NGC_JOB_ID"])
```

- **Project** = The W&B project name should correspond to your Base Command platform team name. In this example, the Base Command team name "nvtest-demo" is reflected as project name on W&B.

  Team name on Base Command Platform:

  ![nvtest-demo](image)

  Project name on W&B:
ID = The ID is unique to each run. It must be unique in a project and if a run is deleted, the ID can’t be reused. Refer to the W&B documentation for additional details. In this example, the ID is named after the Jupyter notebook and model configuration.

Name = The purpose of the run name is to identify each run in the W&B UI. In this example, we name each run according to the related NGC job ID and therefore ensure that each individual run has a different name to ensure easy differentiation between runs.

14.6. Supplemental Reading

Refer to other chapters in this document as well as the Weights & Biases documentations for additional information and details.
Chapter 15. Deregistering

This chapter describes the features and procedures for de-registering users from the system. Only org administrators can de-register users and remove artifacts (datasets, workspaces, results, container images, models etc). All artifacts owned by the user must be removed or archived before removing the user from the system.

Perform the following actions:

Remove all workspaces, datasets, and results
- To archive, download each item:
  - `ngc workspace download <workspace-id> --dest <path>`
  - `ngc dataset download <dataset-id> --dest <path>`
  - `ngc result download <result-id> --dest <path>`
- To remove the items:
  - `ngc workspace remove <workspace-id>`
  - `ngc dataset remove <dataset-id>`
  - `ngc result remove <result-id>`

Remove all container images, charts, and resources
- To archive, download each item:
  - `ngc registry image pull <repository-name>:<tag>`
  - `ngc registry chart pull <chart-name>:<version>`
  - `ngc registry resource download-version <resource-name>:<version>`
- To remove items:
  - `ngc registry image remove <repository-name>:<tag>`
  - `ngc registry chart remove <chart-name>:<version>`
  - `ngc registry resource remove <resource-name>:<version>`

Delete Users
- list users in the current team:
Deregistering

ngc team list-users

- Remove each user from the team:
  ngc team remove-user <user-email>

Delete Teams

Once all users in a team have been removed, delete the team:

ngc org remove-team <team-name>
Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Base Command are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2021 NVIDIA Corporation and affiliates. All rights reserved.