NGC Catalog

User Guide
Table of Contents

Chapter 1. What is NVIDIA NGC..................................................................................................................1
  1.1. NGC Catalog.........................................................................................................................................1

Chapter 2. Why NGC Software...................................................................................................................3

Chapter 3. Accessing NGC Software..........................................................................................................4

Chapter 4. Registering and Activating Your NGC Account........................................................................5
  4.1. Signing Up for an NGC Registered Account..........................................................................................5
  4.2. Generating Your NGC API Key............................................................................................................8

Chapter 5. Introduction to the NGC Catalog and Registry CLIs.............................................................10
  5.1. Installing NGC Catalog CLI..............................................................................................................10
  5.2. Installing NGC Registry CLI..............................................................................................................11
  5.3. Getting Help Using NGC CLI..............................................................................................................11
  5.4. Running the Diagnostics....................................................................................................................11

Chapter 6. Docker Containers.................................................................................................................13
  6.1. What Is a Docker Container?..............................................................................................................13
  6.2. Why Use A Container?.......................................................................................................................13

Chapter 7. NGC Container Images in NGC Catalog..............................................................................15
  7.1. Why NGC Container Images?..............................................................................................................15
  7.2. NVIDIA Container Toolkit..................................................................................................................16
  7.3. NVIDIA CUDA Toolkit......................................................................................................................17
  7.4. Running Singularity Containers..........................................................................................................17
    7.4.1. Prerequisites....................................................................................................................................17
    7.4.2. Converting to Singularity Image....................................................................................................18
    7.4.3. Running the Singularity Container...............................................................................................18
      7.4.3.1. Directory Access....................................................................................................................19
      7.4.3.2. Command Line Execution with Singularity..............................................................................19
      7.4.3.3. Interactive Shell with Singularity............................................................................................19

Chapter 8. Prerequisites for Using NGC Catalog Container Images.........................................................20
  8.1. Prerequisites for Using NGC Containers on DGX Systems..............................................................20
  8.2. Prerequisites for Using NGC Containers on Cloud Platforms.........................................................20
  8.3. Prerequisites for Using NGC Containers on Other NVIDIA GPUs...............................................21
  8.4. HPC Visualization Containers..........................................................................................................21
    8.4.1. Prerequisites For HPC Visualization Containers...........................................................................22
    8.4.2. Installing NVIDIA Docker 2.0....................................................................................................22
    8.4.3. ParaView With NVIDIA Holodeck..............................................................................................23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4.4. ParaView With NVIDIA IndeX</td>
<td>23</td>
</tr>
<tr>
<td>8.4.4.1. Single-Machine With GLX</td>
<td>23</td>
</tr>
<tr>
<td>8.4.4.2. Server Container With EGL</td>
<td>24</td>
</tr>
<tr>
<td>8.4.4.3. GLX Client Connecting To A Server</td>
<td>24</td>
</tr>
<tr>
<td>8.4.5. ParaView With NVIDIA OptiX</td>
<td>24</td>
</tr>
<tr>
<td>8.4.5.1. Single-Machine Container With GLX</td>
<td>25</td>
</tr>
<tr>
<td>8.4.5.2. Server Container With EGL</td>
<td>25</td>
</tr>
<tr>
<td>8.4.5.3. Running The GLX Client And Attaching To The Server</td>
<td>25</td>
</tr>
<tr>
<td>8.4.5.4. Optional: Using The ParaView .config File</td>
<td>25</td>
</tr>
<tr>
<td>Chapter 9. Pulling NGC Containers from NGC Catalog</td>
<td>27</td>
</tr>
<tr>
<td>9.1. Key NGC Container Registry Terminology</td>
<td>27</td>
</tr>
<tr>
<td>9.2. Accessing And Pulling an NGC Container Image via the Docker CLI</td>
<td>28</td>
</tr>
<tr>
<td>9.2.1. Logging in to the NGC container registry</td>
<td>28</td>
</tr>
<tr>
<td>9.2.2. Pulling A Container From The NGC container registry Using The Docker CLI</td>
<td>29</td>
</tr>
<tr>
<td>9.3. Accessing and Pulling an NGC Catalog Container Using NGC CLI</td>
<td>30</td>
</tr>
<tr>
<td>9.3.1. Viewing Container Image Information</td>
<td>30</td>
</tr>
<tr>
<td>9.3.2. Pulling a Container Image</td>
<td>31</td>
</tr>
<tr>
<td>Chapter 10. NGC Container Image Versions</td>
<td>32</td>
</tr>
<tr>
<td>Chapter 11. Running an NGC Container</td>
<td>33</td>
</tr>
<tr>
<td>11.1. Enabling GPU Support for NGC Containers</td>
<td>33</td>
</tr>
<tr>
<td>11.2. Running NGC Containers with Runtime Resources</td>
<td>35</td>
</tr>
<tr>
<td>11.2.1. Specifying A User</td>
<td>36</td>
</tr>
<tr>
<td>11.2.2. Setting The Remove Flag</td>
<td>36</td>
</tr>
<tr>
<td>11.2.3. Setting The Interactive Flag</td>
<td>36</td>
</tr>
<tr>
<td>11.2.4. Setting The Volumes Flag</td>
<td>37</td>
</tr>
<tr>
<td>11.2.5. Setting The Mapping Ports Flag</td>
<td>37</td>
</tr>
<tr>
<td>11.2.6. Setting The Shared Memory Flag</td>
<td>38</td>
</tr>
<tr>
<td>11.2.7. Setting The Restricting Exposure Of GPUs Flag</td>
<td>38</td>
</tr>
<tr>
<td>11.2.8. Container Lifetime</td>
<td>38</td>
</tr>
<tr>
<td>Chapter 12. Multi-Architecture Support for NGC Container Images</td>
<td>40</td>
</tr>
<tr>
<td>Chapter 13. Examples of Using NGC Containers</td>
<td>42</td>
</tr>
<tr>
<td>13.1. Benefits And Limitations To Customizing A Container</td>
<td>43</td>
</tr>
<tr>
<td>13.2. Example 1: Building A Container From Scratch</td>
<td>43</td>
</tr>
<tr>
<td>13.3. Example 2: Customizing A Container Using Dockerfile</td>
<td>45</td>
</tr>
<tr>
<td>13.4. Example 3: Customizing A Container Using docker commit</td>
<td>46</td>
</tr>
<tr>
<td>13.5. Example 4: Developing A Container Using Docker</td>
<td>48</td>
</tr>
<tr>
<td>13.5.1. Example 4.1: Package The Source Into The Container</td>
<td>50</td>
</tr>
</tbody>
</table>
Chapter 14. Customizing Containers

14.1. Benefits And Limitations To Customizing A Container

14.2. Example 1: Building A Container From Scratch

14.3. Example 2: Customizing A Container Using Dockerfile

14.4. Example 3: Customizing A Container Using docker commit

14.5. Example 4: Developing A Container Using Docker

14.5.1. Example 4.1: Package The Source Into The Container

Chapter 15. Models

15.1. What are Models

15.2. Downloading Models via NGC CLI

15.3. Downloading Models via WGET/cURL

15.4. Downloading Models via Web UI

Chapter 16. Resources

16.1. What are Resources

16.2. Downloading Resources

Chapter 17. Helm Charts

17.1. Setting Up an NGC Helm Repository

17.2. Searching for Available Helm Charts in the NGC Catalog

17.3. Fetching Helm Charts

17.4. Using the NGC Website to View the List of Helm Charts and Get Fetch Commands

Chapter 18. SDK and AI Toolkits

Chapter 19. Deep Learning Frameworks

19.1. GPU Operator
Chapter 1. What is NVIDIA NGC

An overview of what constitutes NVIDIA NGC.

1.1. NGC Catalog

The NGC Catalog is a curated set of GPU-optimized software for AI, HPC and Visualization.

The content provided by NVIDIA and third party ISVs simplify building, customizing and the integration of GPU-optimized software into workflows, accelerating the time to solutions for users.

The NGC Catalog consists of containers, pre-trained models, Helm charts for Kubernetes deployments and industry specific AI toolkits with software development kits (SDKs).

Containers

Containers package software applications, libraries, dependencies, and run time compilers in a self-contained environment so they can be easily deployed across various compute environments. They enable software portability and through a single command, users can pull, run and scale applications across the cloud, the datacenter and the edge. The NGC Catalog hosts a broad range of containers including deep learning frameworks, machine learning, HPC and visualization applications that maximize the utilization of GPU environments.

Models and Resources

The NGC Catalog offers pre-trained models for a wide range of common AI tasks that are optimized for NVIDIA Tensor Core GPUs. The pre-trained models can be used for inference or fine tuned with transfer learning, saving data scientists and developers valuable time. Resources provide reference neural network architectures across all domains and popular frameworks with the state-of-the-art accuracy to enable reproducibility as well as documentation and code samples which make it easy to get started with deep learning.

Helm Charts

Kubernetes is a container orchestrator that facilitates the deployment and management of containerized applications and microservices. A Helm chart is a package manager that allows DevOps to more easily configure, deploy and update applications across Kubernetes.
environments. The NGC Catalog provides Helm charts for the deployment of GPU-optimized applications and SDKs.

**Software Development Kits**

SDKs deliver all the tooling users need to build and deploy AI applications across domains such as medical imaging, conversational AI or video analytics. They include annotation tools for data labeling, pre-trained models for customization with transfer learning and SDKs that enable deployment across the cloud, the data center, or the edge for low-latency inference. Users can learn more about the NGC Catalog content by visiting [NGC directly](https://ngc.nvidia.com).
NGC provides software to meet the needs of data scientists, developers, and researchers with various levels of AI expertise.

Software hosted on NGC undergoes scans against an aggregated set of common vulnerabilities and exposures (CVEs), crypto and private keys.

In addition to security scanning, NGC software is tested against a broad range of GPU-enabled platforms including public cloud instances, workstations and OEM servers targeted for the datacenter or edge deployments. Supported GPUs include V100, A100, T4, Jetson and the RTX Quadro.

NGC software is tested and assured to scale to multiple GPUs and in some cases, to scale to multi-node ensuring users maximize the use of their gpu-powered servers out of the box.

For a select set of containers, NVIDIA offers NGC Support Services for running software on DGX platforms or certified OEM servers. The service gives enterprise IT direct access to NVIDIA subject matter experts to quickly address software issues and minimize system downtime.
Chapter 3. Accessing NGC Software

How to access NGC software.

There are three ways to access software on NGC.

**Guest Access**: Content under guest access does not require users to register or sign-in to pull or download the software.

The majority of software on the NGC Catalog falls under guest access. However, it is important to note that many third party applications require a license key which can be sourced directly from ISVs. So while pulling the container from the NGC Catalog does not require sign-in, a user might have to reach out to an ISV to obtain a license key for further use.

**Authenticated Access**: Software under authenticated access requires a user to create an NVIDIA NGC account or sign-in to the NGC Catalog. Pulling or downloading the software requires the use of their API key.

**Approved Access**: To access the software under this category, the publisher must grant user approval. Instructions to request access are provided in the overview section of the respective software. Once access is granted, the user will receive a notification with further instructions for accessing the software.
Chapter 4. Registering and Activating Your NGC Account

Instructions for registering and activating your NGC account.

4.1. Signing Up for an NGC Registered Account

This section describes the process of signing up for an NGC account.

1. From your browser, go to the NGC Catalog landing page and then click Create an Account from the upper right of the page.

2. Enter your full name and email address, and check the boxes as needed.
   - Agree to the NVIDIA Account Terms of Use (mandatory)
   - Consent to share registration data with NVIDIA partners (optional)
   - Agree to receive news and updates on NVIDIA Software (optional)
Registering and Activating Your NGC Account

Click **Sign Up** when done.

You will receive a welcome email with instructions on setting up your account access.

3. Click the link in the welcome email to open the **Set Password** page in a browser.
4. Follow the instructions to create your password.

   You will be signed in to the NGC website at this point. You can sign in at any time by opening the login page at https://ngc.nvidia.com/signin/email and then entering your email and password.

5. The first time that you sign in, you must agree to the NVIDIA GPU Cloud Terms of Use before you can enter the website.
Accept the Terms of Use and then sign in.

4.2. Generating Your NGC API Key

This section describes how to obtain an API key to access locked container images from the NGC Registry.

1. Sign in to the NGC website.
   From a browser, go to https://ngc.nvidia.com/signin/email and then enter your email and password.

2. In the top right corner, click your user account icon and select Setup.
3. Click **Get API key** to open the Setup > API Key page.
   
   The API Key is the mechanism used to authenticate your access to the NGC container registry.

4. 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.

5. 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.
Chapter 5. Introduction to the NGC Catalog and Registry CLIs

Introduction to NGC CLIs
The NGC CLIs are command-line interfaces for managing content within the NGC Registry. The CLI operates within a shell and lets you use scripts to automate commands.

NGC Catalog CLI
The NGC Catalog CLI is available to you if you have guest access to the NGC Registry, and with it you can

- View a list of GPU-accelerated Docker container images, pre-trained deep-learning models, and scripts for creating deep-learning models.
- Download container images, models, and resources.

NGC Registry CLI
The NGC Registry CLI is available to you if you are logged in with your own NGC account or with an NGC Private Registry 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 container images, models, and resources.
- Create and manage users and teams (available to NGC Private Registry administrators).

5.1. Installing NGC Catalog CLI
To install NGC Catalog CLI,

1. Enter the NGC website (https://ngc.nvidia.com) as a guest user.
2. In the top right corner, click Welcome Guest and then select Setup from the menu.
3. 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 Catalog CLI.
5. Follow the instructions to install the CLI.
6. Verify the installation by entering `ngc --version`.
   The output should be "NGC Catalog CLI x.y.z" where x.y.z indicates the version.

5.2. Installing NGC Registry CLI

To install NGC Registry CLI,
1. Log in to your enterprise account on the NGC website (https://ngc.nvidia.com).
2. In the top right corner, click your user account icon and select Setup, then click Downloads under Install NGC CLI from the Setup page.
3. From the CLI Install page, click the Windows, Linux, or MacOS tab, according to the platform from which you will be running NGC Registry CLI.
4. Follow the instructions to install the CLI.
5. Verify the installation by entering `ngc --version`.
   The output should be "NGC Registry CLI x.y.z" where x.y.z indicates the version.

5.3. Getting Help Using NGC CLI

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 ngc, enter

```
C:\>ngc -h
```

**Example 2**: To view a description of the registry image command and options, enter

```
C:\>ngc registry image -h
```

**Example 2**: To view a description of the registry image info command and options, enter

```
C:\>ngc registry image info -h
```

5.4. Running the Diagnostics

You can view the following diagnostic information for the NGC CLI user:

- Current time
- Operating system
- Disk usage
- Current directory size
- Memory usage
NGC CLI environment variables set
API gateway connectivity
API connectivity to the container registry
API connectivity to the model registry
External IP
NGC CLI configuration values
User information (ID, name, and email)
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.

Note:
nvcr.io/nvidia, nvcr.io/partners and nvcr.io/hpc are reserved namespaces and do not grant user permissions to push or delete container images, models, helm and all artifacts.
Chapter 6. Docker Containers

Over the last few years there has been a dramatic rise in the use of software containers for simplifying deployment of data center applications at scale. Containers encapsulate an application along with its libraries and other dependencies to provide reproducible and reliable execution of applications and services without the overhead of a full virtual machine.

GPU support within Docker containers enables GPU-based applications that are portable across multiple machines in a similar way to how Docker enables CPU-based applications to be deployed across multiple machines.

**Docker container**
- A Docker container is an instance of a Docker image. A Docker container deploys a single application or service per container.

**Docker image**
- A Docker image is simply the software (including the filesystem and parameters) that you run within a Docker container.

### 6.1. What Is a Docker Container?

A Docker container is a mechanism for bundling a Linux application with all of its libraries, data files, and environment variables so that the execution environment is always the same, on whatever Linux system it runs and between instances on the same host.

Unlike a VM which has its own isolated kernel, containers use the host system kernel. Therefore, all kernel calls from the container are handled by the host system kernel. Users can use Docker containers as the mechanism for deploying deep learning frameworks on-prem, in the cloud, or at the edge.

A Docker container is the running instance of a Docker image.

### 6.2. Why Use A Container?

Using containers allows you to package your application, dependencies, and environment variables into a single image rather than on each system you run on. Additional benefits to using containers include:

- Multiple versions of the same application, which may have conflicting software dependencies, can run on the same server.
Containerized applications can be deployed on premise, in the cloud, or at the edge.

Specific GPU resources can be allocated to a container for isolation and better performance.

Easily share, collaborate, and test applications across different environments.

Resolve network-port conflicts between applications by mapping container-ports to specific externally-visible ports when launching the container.
Chapter 7. NGC Container Images in NGC Catalog

NGC Containers are designed to enable a software platform centered around minimal OS requirements, Docker and driver installation on the server or workstation, and provisioning of all application and SDK software in the NGC containers through the NGC container registry.

NGC Catalog is a curated set of fully integrated and optimized container images for Deep Learning, HPC and Visualization applications.

- Deep learning framework containers that take full advantage of NVIDIA GPUs in both single GPU and multi-GPU configurations. They include CUDA Toolkit, DIGITS workflow, and the following deep learning frameworks: NVCAFFE, Caffe2, Microsoft Cognitive Toolkit (CNTK), MXNet, PyTorch, TensorFlow, Theano, and Torch. These framework containers are delivered ready-to-run, including all necessary dependencies such as CUDA runtime, NVIDIA libraries, and an operating system. NVIDIA updates these deep learning containers monthly to ensure they continue to provide peak performance.

- NGC also hosts a catalog of HPC applications such as NAMD, GROMACS, LAMMPS, RELION, CHROMA, MILC and many more.

- In addition to HPC applications, NGC also industry’s leading visualization tools, including ParaView with NVIDIA IndeX volume renderer, NVIDIA OptiX ray-tracing library and NVIDIA Holodeck for interactive real-time visualization and high-quality visuals.

- NGC also hosts popular third-party GPU ready application containers which conform to NGC container standards and best practices, making it easy to get the most out-of your NVIDIA GPU

7.1. Why NGC Container Images?

Every image hosted on NGC Catalog undergoes a security scan and GPU-performance test by the NGC team.

Container Image Security Scan for NGC Catalog Container Images

A critical function of enterprise risk and security assessment processes is to ensure all container images being used in development or deployment have been tested for known vulnerabilities. Security flaws and vulnerabilities in the software packaged within the container...
images can lead to production delays. Thus, as a foundational part of NGC mission, all NGC container images hosted on NGC Catalog undergo a security scan. This can be a primary means of defense against insecure code with images.

In mid-2019, NGC introduced a security scan tool in the publishing pipeline. Each NGC hosted image on NGC Catalog undergoes security scan for vulnerability assessment prior to publishing. These scans protect NGC end-users from a broad range of operating system vulnerabilities and an aggregated set of Common Vulnerabilities and Exposures (CVEs).

The internal scanning tool developed by NVIDIA uses the Common Vulnerabilities and Exposures (CVEs) database which provides a list of scan findings. This tool leverages Anchore (open source version). The known critical and high vulnerabilities are addressed before publishing an image to NGC. Please note that scan results may vary in time since the CVE database is updated in real time.

**GPU-performance tests for NGC Catalog Container Images**

To ensure that NVIDIA GPU owners can perform well with NGC hosted images, all container images undergo basic installation, functional, scalability and performance tests on GPU’s.

These tests are performed on an array of GPU’s ranging from NVIDIA T4’s to V100 to DGX for performance and multi-gpu for scale. The selection of supported GPUs may be limited to few architectures or single/multi GPU based on the business use case of application containers.

These tests are performed on both bare-metal installations and cloud deployed instances to ensure both on-prem and cloud deployment use-cases.

### 7.2. NVIDIA Container Toolkit

The NVIDIA Container Toolkit allows users to build and run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPUs. It integrates with many popular container runtimes including Docker, podman, CRI-O, LXC etc.

To enable portability in Docker images that leverage NVIDIA GPUs, NVIDIA developed nvidia-docker, an open-source project hosted on Github that provides the two critical components needed for portable GPU-based containers:

1. Driver-agnostic CUDA images.
2. A Docker command line wrapper that mounts the user mode components of the driver and the GPUs (character devices) into the container at launch.

nvidia-docker is a Docker command line wrapper that provisions a container with the necessary components to execute code on the GPU.

When working with containers that utilize GPUs, the only command that must be executed through an nvidia-docker command is the run command. For all other functionalities, docker commands can be used.

But for simplicity in this documentation we use nvidia-docker for all commands.
7.3. NVIDIA CUDA Toolkit

The NVIDIA® CUDA® Toolkit provides a development environment for creating high-performance GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize, and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms, and HPC supercomputers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler and a runtime library to deploy your application.

GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning, and graph analytics. For developing custom algorithms, you can use available integrations with commonly used languages and numerical packages as well as well-published development APIs. Your CUDA applications can be deployed across all NVIDIA GPU families available on premise and on GPU instances in the cloud. Using built-in capabilities for distributing computations across multi-GPU configurations, scientists and researchers can develop applications that scale from single GPU workstations to cloud installations with thousands of GPUs.

7.4. Running Singularity Containers

NGC supports both Docker and Singularity container runtimes. While Docker is prevalent primarily in enterprises, Singularity has become ubiquitous in the HPC community. Singularity was developed to better satisfy the requirements of HPC users and system administrators, including the ability to run containers without superuser privileges.

NGC containers can be easily used with Singularity. Let’s use the NGC NAMD container to illustrate. NAMD is a parallel molecular dynamics application designed for high-performance simulation of large biomolecular systems, developed by the Theoretical and Computational Biophysics Group at the University of Illinois.

7.4.1. Prerequisites

These instructions assume the following.

- You have Singularity v2.6+ installed on your system
- You have performed the following steps from the NGC website.
  - Signed up for an NGC account at https://ngc.nvidia.com/signup.
  - Created an NGC API key for access to the NGC container registry
  - Browsed the NGC website and identified an available NGC container and tag to run
- Ensure you have correctly set udev rules which are detailed [here]

Note:
7.4.2. Converting to Singularity Image

Before running with Singularity you must set NGC container registry authentication credentials. This is most easily accomplished by setting the following environment variables.

```bash
$ export SINGULARITY_DOCKER_USERNAME='$oauthtoken'
$ export SINGULARITY_DOCKER_PASSWORD=<NVIDIA NGC API key>
```

```tcsh
> setenv SINGULARITY_DOCKER_USERNAME '$oauthtoken'
> setenv SINGULARITY_DOCKER_PASSWORD <NVIDIA NGC API key>
```

More information describing how to obtain and use your NVIDIA NGC API key can be found here.

Once credentials are set in the environment, the NGC container can be pulled to a local Singularity image.

```bash
$ singularity build <app_tag>.simg docker://nvcr.io/<repository>/<app:tag>
```

This will save the container to the current directory as `<app_tag>.simg`.

For example to convert the HPC application NAMD hosted on NGC to a Singularity image, run

```bash
$ singularity build namd_2.12-171025.simg docker://nvcr.io/hpc/namd:2.12-171025
```

After the build has finished the Singularity image file, `namd_2.12-171025.simg`, will be available for use in the current working directory.

7.4.3. Running the Singularity Container

Once the local Singularity image has been pulled, the following modes of running are supported.

- Command line execution with Singularity
- Interactive shell with Singularity

To leverage NVIDIA GPUs, you must use the Singularity flag `--nv` when running the containers. More singularity flags are explained here.

---

**Important:**

For Amazon Machine Image Users

Amazon Machine Images on Amazon Web Service have a default root umask of 077. Singularity must be installed with a umask of 022 to run properly. To [re]install Singularity with correct permissions:

- Uninstall Singularity (if it is installed)
7.4.3.1. Directory Access

Singularity containers are themselves ostensibly read only. In order to provide application input and persist application output we’ll bind a host directory into our container, this is accomplished through the Singularity `\(-B\)` flag. The format of this flag is `\(-B <host_src_dir>:<container_dst_dir>\)`. Once a host directory is bound into the container we can interact with this directory from within the container exactly as we can outside the container.

It is also often convenient to use the `\(--pwd <container_dir>\)` flag, which will set the present working directory of the command to be run within the container.

The Singularity commands below will mount the present working directory on the host to `\(/host_pwd\)` in the container process and set the present working directory of the container process to `\(/host_pwd\)`. With this set of flags the `\(<cmd>\)` to be run will be launched from the host directory Singularity was called from.

```bash
$ singularity exec --nv -B $(pwd):/host_pwd --pwd /host_pwd <image.simg> <cmd>
```

---

**Note:**

Binding to a directory which doesn’t exist within the container image requires kernel and configuration support that may not be available on all systems, particularly those running older kernels such as CentOS/RHEL 6. When in doubt contact your system administrator.

7.4.3.2. Command Line Execution with Singularity

Running the container with Singularity from the command line looks similar to the command below.

```bash
$ singularity exec --nv <app_tag>.simg <command_to_run>
```

For example, to run the NAMD executable in the container

```bash
$ singularity exec --nv namd_2.12-171025.simg /opt/namd/namd-multicore
```

7.4.3.3. Interactive Shell with Singularity

To start a shell within the container, run the command below

```bash
$ singularity exec --nv <app_tag>.simg /bin/bash
```

For example, to start an interactive shell in the NAMD container

```bash
$ singularity exec --nv namd_2.12-171025.simg
```
To enable portability in Docker images that leverage GPUs, three methods of providing GPU support for Docker containers have been developed.

- Running Docker-ce 19.03 or later for Native GPU support
- `nvidia-docker2`
- `nvidia-docker`

Each of these methods mount the user mode components of the NVIDIA driver and the GPUs into the Docker container at launch. They allow NGC containers to take full advantage of NVIDIA GPUs.

8.1. Prerequisites for Using NGC Containers on DGX Systems

DGX™ users should follow the instructions in the Preparing Your DGX System For Use With NVIDIA Container Runtime.

8.2. Prerequisites for Using NGC Containers on Cloud Platforms

On each of the major public cloud providers, NVIDIA publishes customized virtual machine images (VMIs) with updated OS’s and NVIDIA driver that have been tested with NGC containers. These VMIs have the NVIDIA Container Toolkit pre-installed allowing you to begin pulling and running Docker containers from NGC right away. To use the NVIDIA NGC GPU-optimized VMIs on cloud platforms, you would need -
8.3. Prerequisites for Using NGC Containers on Other NVIDIA GPUs

- Users running NGC containers on other NVIDIA GPUs or in a virtual GPU environment should follow the corresponding instructions for TITAN PCs, Quadro PCs, or vGPUs.
- Other users should follow the nvidia-docker installation documentation at [nvidia-docker installation](https://github.com/nvidia/docker) and install the latest NVIDIA drivers for your GPU product type and series for your operating system. If NVIDIA drivers are not already configured on your system, then install them from here: [Download Drivers](https://www.nvidia.com/Download/index.aspx).  
- Ensure you have an NVIDIA GPU supporting Compute Unified Device Architecture® (CUDA) version with compute capability 6.0.0 or higher. For example, Pascal GPU architecture generation or later.

8.4. HPC Visualization Containers

In addition to accessing the NVIDIA optimized frameworks and HPC containers, the NVIDIA GPU Cloud (NGC) container registry also hosts the following scientific visualization containers for HPC. These containers rely on the popular scientific visualization tool called [ParaView](https://www.paraview.org/).

Visualization in an HPC environment typically requires remote visualization, that is, data resides and is processed on a remote HPC system or in the cloud, and the user graphically interacts with this application from their workstation. As some visualization containers require specialized client applications, the HPC visualization containers consist of two components:
Server container
The server container needs access to the files on your server system. Details on how to grant this access are provided below. The server container can run both in serial mode or in parallel. For this alpha release, we are focusing on the serial node configuration. If you are interested in parallel configuration, contact hpcviscontainer@nvidia.com.

Client container
To ensure matching versions of the client application and the server container, NVIDIA provides the client application in a container. Similarly, to the server container, the client container needs access to some of the ports to establish connection with the server container.
In addition, the client container needs access to the users’ X server for displaying the graphical user interface.

8.4.1. Prerequisites For HPC Visualization Containers

- Install docker-ce and nvidia-docker2. First install docker-ce, then install nvidia-docker2 for your operating system and Docker version. For a script to install nvidia-docker2, see Installing NVIDIA Docker 2.0.

  Note: If you already have nvidia-docker1 installed and intend to keep it, you can install nvidia-container-runtime.

- Install the NVIDIA Display driver version 384.57 or onwards depending on your GPU product type and series for your operating system. For more information, see Download Drivers.

- Ensure you have an NVIDIA GPU supporting Compute Unified Device Architecture® (CUDA) version with compute capability 6.0.0 or higher. For example, Pascal GPU architecture generation or later.

- Log into the NVIDIA® GPU Cloud (NGC) Container Registry located at nvcr.io using your NGC API key. For step-by-step instructions on how to gain access and get your API key, see Generating Your NGC API Key.

8.4.2. Installing NVIDIA Docker 2.0

The following script installs NVIDIA Docker 2.0 which is a prerequisite to pulling the ParaView with NVIDIA IndeX HPC visualization container.

Full support for concurrent graphics and compute capabilities in containers is supported in NVIDIA Docker 2.0. Current installations of NGC run on NVIDIA Docker 1.0. Prior to using a container on any of these instances, NVIDIA Docker 2.0 must be installed.

Use the following script below to install NVIDIA Docker 2.0 on your instance.

```
# Install NVIDIA Docker 2.0
docker volume ls -q -f driver=nvidia-docker | \
  xargs -r -I{} -n1 docker ps -q -a -f volume={} | xargs -r docker rm -f
sudo apt-get purge -y nvidia-docker
curl -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
sudo tee /etc/apt/sources.list.d/nvidia-docker.list <<< \
"deb https://nvidia.github.io/libnvidia-container/ubuntu16.04/amd64 /"  
```
Prerequisites for Using NGC Catalog Container Images

deb https://nvidia.github.io/nvidia-container-runtime/ubuntu16.04/amd64/
deb https://nvidia.github.io/nvidia-docker/ubuntu16.04/amd64/

sudo apt-get -y update
sudo apt-get install -y nvidia-docker2
sudo pkill -SIGHUP dockerd

# Tests
#docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi

8.4.3. ParaView With NVIDIA Holodeck

Currently, the ParaView with NVIDIA Holodeck container requires a running X server both on the server host and the client host. Therefore, only a single container image is required.

1. Create X-forwarding variables for your container.

   ```
   XSOCK=/tmp/.X11-unix; XAUTH=/tmp/.docker.xauth;
touch /tmp/.docker.xauth;
xauth nlist :0 | sed -e 's/^..../ffff/' | xauth -f /tmp/.docker.xauth nmerge -
   ```

2. On the server host, start the ParaView Holodeck server:

   ```
   docker run --rm -it --runtime=nvidia 
   -v /tmp/.X11-unix:/tmp/.X11-unix -v /tmp/.docker.xauth:/tmp/.docker.xauth 
   -e XAUTHORITY=/tmp/.docker.xauth -e DISPLAY=:0 
   -p 11111:11111 
   --shm-size=4g 
   nvcr.io/nvidia-hpcvis/paraview-holodeck:glx-17.11.13-beta 
   ./service.sh externalvis pvserver
   ```

   The Holodeck render window showing a space scene displays.

   The server container is ready after you receive a message similar to the following:

   "Accepting connection(s): [...]::11111"

3. Set up X access and start the client container on the client host. Ensure you replace your_server_hostname.

   ```
   XSOCK=/tmp/.X11-unix; XAUTH=/tmp/.docker.xauth 
touch /tmp/.docker.xauth 
xauth nlist :0 | sed -e 's/^..../ffff/' 
| xauth -f /tmp/.docker.xauth nmerge -docker run --rm -it --runtime=nvidia 
- v /tmp/.X11-unix:/tmp/.X11-unix - v /tmp/.docker.xauth:/tmp/.docker.xauth 
- e XAUTHORITY=/tmp/.docker.xauth - e DISPLAY=:0 
 nvcr.io/nvidia-hpcvis/paraview-holodeck:glx-17.11.13-beta 
 sh -c paraview\ --server-url=cs://your_server_hostname:11111
   ```

   The ParaView user interface displays.

8.4.4. ParaView With NVIDIA IndeX

To support both X-enabled and headless hosts, the ParaView IndeX container image is available with GLX and EGL support. The following section shows how to launch the IndeX container with different use cases.

For more information about ParaView, see the ParaView User's Guide and the NVIDIA IndeX SDK.

8.4.4.1. Single-Machine With GLX
1. Login to the docker repository and pull the X display-enabled container on your workstation:
   docker pull nvcr.io/nvidia-hpcvis/paraview-index:glx-17.11.13-beta

2. Specify X-forwarding variables:
   XSOCK=/tmp/.X11-unix; XAUTH=/tmp/.docker.xauth
   touch /tmp/.docker.xauth
   xauth nlist :0 | sed -e 's/^..../ffff/' |
   | xauth -f /tmp/.docker.xauth nmerge

3. Run the image. In this example, host system data in the current directory $(pwd) are mounted to both /work in the container. This should be modified as desired by the user.
   docker run --rm -it --runtime=nvidia 
   -v /tmp/.X11-unix:/tmp/.X11-unix -v /tmp/.docker.xauth:/tmp/.docker.xauth 
   -v $(pwd)/work -e XAUTHORITY=/tmp/.docker.xauth -e DISPLAY=:0 
   nvcr.io/nvidia-hpcvis/paraview-index:glx-17.11.13-beta 
   sh -c paraview

8.4.4.2. Server Container With EGL

In a typical client-server setup, one container acting as the server will run remotely on a display-less machine, connected to a second container that runs locally on a workstation and provides the graphical front end.

Use the following command to pull the EGL-enabled, no-display container from the NGC registry on the server host:
   docker pull nvcr.io/nvidia-hpcvis/paraview-index:egl-17.11.13-beta

Run the server component on the server host. We listen on the default port 11111:
   docker run --runtime=nvidia -p 11111:11111 --rm -it 
   nvcr.io/nvidia-hpcvis/paraview-index:egl-17.11.13-beta sh -c pvserver

8.4.4.3. GLX Client Connecting To A Server

Pull the X display-enabled container on your workstation:
   docker pull nvcr.io/nvidia-hpcvis/paraview-index:glx-17.11.13-beta

Set up X access and launch the client application container [make sure to replace your_server_hostname with the address of your ParaView server host]:
   XSOCK=/tmp/.X11-unix; XAUTH=/tmp/.docker.xauth
   touch /tmp/.docker.xauth
   xauth nlist :0 | sed -e 's/^..../ffff/' |
   | xauth -f /tmp/.docker.xauth nmerge -
   docker run --rm -it --runtime=nvidia 
   -v /tmp/.X11-unix:/tmp/.X11-unix -v /tmp/.docker.xauth:/tmp/.docker.xauth 
   -e XAUTHORITY=/tmp/.docker.xauth -e DISPLAY=:0 
   nvcr.io/nvidia-hpcvis/paraview-index:glx-17.11.13-beta 
   sh -c paraview --server-url=cs://your_server_hostname:11111

8.4.5. ParaView With NVIDIA OptiX

The ParaView with NVIDIA OptiX container is designed to run ParaView as a user normally would outside a container. The following sections show how to launch the OptiX container with different use cases.

For more information about ParaView see the ParaView User’s Guide and the NVIDIA OptiX SDK.
8.4.5.1. Single-Machine Container With GLX

On systems with a physical display, or when running a ParaView client, users will wish to launch a container with GLX support. This can be done as follows.

1. Pull the docker image:
   ```bash
docker pull nvcr.io/nvidia-hpcvis/paraview-optix:glx-17.11.13-beta
   ```

2. Set up X11 forwarding variables:
   ```bash
   XSOCK=/tmp/.X11-unix; XAUTH=/tmp/.docker.xauth;
touch /tmp/.docker.xauth;
xauth nlist :0 | sed -e 's/^..../ffff/' | xauth -f /tmp/.docker.xauth nmerge -
   ```

3. Run the image. In this example, host system data in the current directory $(pwd) are mounted to both /work in the container. This should be modified as desired.
   ```bash
   ```

8.4.5.2. Server Container With EGL

Launching a ParaView server on GPU HPC resources often requires EGL support, requiring a separate build of ParaView for which we have a separate container.

1. Pull the container:
   ```bash
docker pull nvcr.io/nvidia-hpcvis/paraview-optix:egl-17.11.13-beta
   ```

2. Specify the connection port and launch the container as follows (in this example, we listen on the default port 11111):
   ```bash
docker run --runtime=nvidia --server-url=cs://your.server.address:11111 --rm -it 
vncr.io/nvidia-hpcvis/paraview-optix:egl-17.11.13-beta sh -c pvserver
   ```

3. For users who wish to run the server on a GLX-capable workstation, it is equally possible to use the GLX image with the pvserver argument.

8.4.5.3. Running The GLX Client And Attaching To The Server

With the server launched, it is then straightforward to use the GLX image to run a client, and connect to the server as follows. Here we assume the server is listening on port 11111, addressable at your.server.address.

```bash
docker pull nvcr.io/nvidia-hpcvis/paraview-optix:glx-17.11.13-beta
   ```

   ```bash
   XSOCK=/tmp/.X11-unix; XAUTH=/tmp/.docker.xauth;
touch /tmp/.docker.xauth;
xauth nlist :0 | sed -e 's/^..../ffff/' \ | xauth -f /tmp/.docker.xauth nmerge -
docker pull nvcr.io/nvidia-hpcvis/paraview-optix:glx-17.11.13-beta
   ```

8.4.5.4. Optional: Using The ParaView .config File

...
It is helpful to reuse ParaView configuration files to maintain settings across ParaView sessions. To do this, first create a new directory for ParaView to store its settings.

```
mkdir pvsettings
```

When issuing the `docker run` command, add the following command as an argument:

```
-v $(pwd)/pvsettings:/home/paraview/.config/ParaView
```

Insert the command before the image URL. For example,

```
docker run --rm -it --runtime=nvidia \
 -v /tmp/.X11-unix:/tmp/.X11-unix -v /tmp/.docker.xauth:/tmp/.docker.xauth \
 -e XAUTHORITY=/tmp/.docker.xauth -e DISPLAY=:0 \
 nvcr.io/nvidia-hpcvis/paraview-optix:glx-17.11.13-beta \
 -v $(pwd)/pvsettings:/home/paraview/.config/ParaView \
 sh -c paraview\ --server-url=cs://your.server.address:11111
```
Chapter 9.  Pulling NGC Containers from NGC Catalog

Before you using a container from the NGC Catalog, review the prerequisites described in the Prerequisites section.

To become a registered NGC user, follow the steps explained in the Registering and Activating Your NGC Account.

9.1.  Key NGC Container Registry Terminology

In order to issue the pull and run commands, ensure that you are familiar with the following concepts.

A pull command looks similar to:

```
docker pull nvcr.io/nvidia/caffe2:17.10
```

A run command looks similar to:

```
docker run --gpus all -it --rm -v local_dir:container_dir nvcr.io/nvidia/caffe2:<xx.xx>
```

Note: The base command `docker run --gpus all` assumes that your system has Docker 19.03-CE installed. See the section Enabling GPU Support for NGC Containers for the command to use for earlier versions of Docker.

The following concepts describe the separate attributes that make up the both commands.

- **nvcr.io**
  The name of the container registry, which for the NGC container registry is `nvcr.io`.

- **nvidia**
  The name of the space within the registry that contains the container. For containers provided by NVIDIA, the registry space is `nvidia`. For more information, see NGC Container Images in NGC Catalog.

- **-it**
  You want to run the container in interactive mode.

- **--rm**
  You want to delete the container when finished.
You want to mount the directory.

**local_dir**
The directory or file from your host system (absolute path) that you want to access from inside your container. For example, the `local_dir` in the following path is `/home/jsmith/data/mnist`.

```
-v /home/jsmith/data/mnist:/data/mnist
```

If you are inside the container, for example, using the command `ls /data/mnist`, you will see the same files as if you issued the `ls /home/jsmith/data/mnist` command from outside the container.

**container_dir**
The target directory when you are inside your container. For example, `/data/mnist` is the target directory in the example:

```
-v /home/jsmith/data/mnist:/data/mnist
```

The tag. For example, 17.10.

You can access the NGC container registry by running a Docker commands from any Linux computer with Internet access on which Docker is installed. You can access the NGC container registry at `nvcr.io` through the Docker CLI.

Before accessing the NGC container registry, see NGC Getting Started Guide for instructions on how to access the website and, if you intend to access locked NGC content, know how to sign up for an NGC account and obtain an API key.

### 9.2. Accessing And Pulling an NGC Container Image via the Docker CLI

#### 9.2.1. Logging in to the NGC container registry

Before accessing locked NGC content, you must sign up for an NGC account and obtain an API key as explained in the section Generating Your NGC API Key. Then log in to the NGC registry from the command line as follows.

1. Log in to the NGC container registry.
   ```
   $ docker login nvcr.io
   ```
2. When prompted for your user name, enter the following text:
   ```
   $oauthtoken
   ```
   The `$oauthtoken` username is a special user name that indicates that you will authenticate with an API key and not a username and password.
3. When prompted for your password, enter your NGC API key as shown in the following example.
   ```
   Username: $oauthtoken
   ```
4. Confirm “Login Success”.

9.2.2. Pulling A Container From The NGC container registry Using The Docker CLI

You can browse the available containers in the NGC container registry by opening the NGC website using a web browser.

1. Browse the NGC Catalog, select the image to pull, then copy the pull command. The following image shows the pull command on the PyTorch container page.

2. Click the download icon to copy the pull command to the clipboard.

3. Log in to nvcr.io, then paste the command in your terminal. For example, to pull the PyTorch container:

   ```$ docker pull nvcr.io/nvidia/pytorch:20.03-py3```

   See the NGC Getting Started Guide for details on using the NGC website.

4. List the Docker images on your system to confirm that the container was pulled.

   ```$ docker images```

   For more information pertaining to your specific container, refer to the `/workspace/README.md` file inside the container.

After pulling a container, you can run jobs in the container to run scientific workloads, train neural networks, deploy deep learning models, or perform AI analytics.
9.3. Accessing and Pulling an NGC Catalog Container Using NGC CLI

If you will be pulling containers from the registry, you will need Docker installed on your local machine. To install Docker on your client machine, follow one of these instructions:

- For Ubuntu Linux: https://docs.docker.com/engine/installation/linux/ubuntulinux/
- For Windows: https://docs.docker.com/engine/installation/windows/#/docker-for-windows
- For MacOS X: https://docs.docker.com/engine/installation/mac/

This document provides an introduction to using the NGC Catalog CLI. For a complete list of commands and options, use the `--help` option as explained in Getting Help Using NGC CLI.

9.3.1. Viewing Container Image Information

There are several commands for viewing information about available container images.

**To list container images:**

```
C:\> ngc registry image list
```

Example output

```
+-----------+----------------+------------+------------+--------------+
| Name      | Repository      | Latest Tag | Image Size | Updated Date |
| Permission| +----------------+------------+------------+--------------+
|-----------+----------------+------------+------------+--------------+
| BigDFTZ   | hpv/bigdftfe   | cuda10-ubun| 2.37 GB    | Oct 18, 2019 | unlocked     |
|           |                | tu1804-ompi|            | 2019         |
| CANDLE    | hpc/candle     | 201803263  | 1.52 GB    | Oct 18, 2019 | unlocked     |
| ...       |                |            |            |              |
```

To view detailed information about a specific image, specify the image and the tag.

**Example:**

```
C:\> ngc registry image info nvidia/caffe:19.02-py2
```

```
Image Information
Name: nvidia/caffe:19.02-py2
Architecture: amd64
Schema Version: 1
```
9.3.2. Pulling a Container Image

With the NGC Registry CLI you can pull (download) images to your system.

To pull an image to your registry space, specify the image and, optionally, the tag.

C:\> ngc registry image pull <image-name>[:<tag>]

If a tag is not specified, then the tag 'latest' will be used.

Note: nvcr.io/nvidia, nvcr.io/partners and nvcr.io/hpc is reserved namespace and does not grant permissions to users to push or delete container images, models, helm and all artifacts.
Each release of an NGC image is identified by a version “tag”. For simpler images this version tag usually contains the version of the major software package in the image. More complex images which contain multiple software packages or versions may use a separate version solely representing the containerized software configuration. One common scheme is versioning by the year and month of the image release. For example, the 20.01 release of an image was released in January, 2020.

An image name consists of two parts separated by a colon. The first part is the name of the container in the repository and the second part is the “tag” associated with the container. These two pieces of information are shown in Figure 2, which is the output from issuing the `docker images` command.

Figure 1. Output from `docker images` command

Figure 2 shows simple examples of image names, such as:

- `nvidia-cuda:8.0-devel`
- `ubuntu:latest`
- `nvcr.io/nvidia/tensorFlow:17.01`

If you choose not to add a tag to an image, by default the word “latest” is added as the tag, however all NGC containers have an explicit version tag.

In the next sections, you will use these image names for running containers. Later in the document, there is also a section on creating your own containers or customizing and extending existing containers.
Chapter 11. Running an NGC Container

Before you can run an NGC deep learning framework container, your Docker environment must support NVIDIA GPUs. To run a container, issue the appropriate command as explained in this chapter, specifying the registry, repository, and tags.

11.1. Enabling GPU Support for NGC Containers

To obtain the best performance when running NGC containers, three methods of providing GPU support for Docker containers have been developed:

- Native GPU support (included with Docker-ce 19.03 or later)
- NVIDIA Container Runtime for Docker [nvidia-docker2 package]
- Docker Engine Utility for NVIDIA GPUs [nvidia-docker package]

The method implemented in your system depends on the DGX OS version installed [for DGX systems], the specific NGC Cloud Image provided by a Cloud Service Provider, or the software that you have installed in preparation for running NGC containers on TITAN PCs, Quadro PCs, or vGPUs.

Refer to the following table to assist in determining which method is implemented in your system.

<table>
<thead>
<tr>
<th>GPU Support Method</th>
<th>When Used</th>
<th>How to Determine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native GPU Support</td>
<td>Included with Docker-ce 19.03 or later</td>
<td>Run <code>docker version</code> to determine the installed Docker version.</td>
</tr>
<tr>
<td>NVIDIA Container Runtime for Docker</td>
<td>If the nvidia-docker2 package is installed</td>
<td>Run <code>nvidia-docker version</code> and check for NVIDIA Docker version 2.0 or later</td>
</tr>
<tr>
<td>Docker Engine Utility for NVIDIA GPUs</td>
<td>If the nvidia-docker package is installed</td>
<td>Run <code>nvidia-docker version</code> and check for NVIDIA Docker version 1.x</td>
</tr>
</tbody>
</table>

Each method is invoked by using specific Docker commands, described as follows.
Running an NGC Container

Using Native GPU support

Note: If Docker is updated to 19.03 on a system which already has nvidia-docker or nvidia-docker2 installed, then the corresponding methods can still be used.

To use the native support on a new installation of Docker, first enable the new GPU support in Docker.

```bash
$ sudo apt-get install -y docker nvidia-container-toolkit
```

This step is not needed if you have updated Docker to 19.03 on a system with nvidia-docker2 installed. The native support will be enabled automatically.

Use docker run --gpus to run GPU-enabled containers.

- Example using all GPUs
  ```bash
  $ sudo docker run --gpus all ...
  ```

- Example using two GPUs
  ```bash
  $ sudo docker run --gpus 2 ...
  ```

Examples using specific GPUs

```bash
$ sudo docker run --gpus "device=1,2" ...

$ docker run --gpus "device=1,2" ...
```

Using the NVIDIA Container Runtime for Docker

With the NVIDIA Container Runtime for Docker installed (nvidia-docker2), you can run GPU-accelerated containers in one of the following ways.

- Use docker run and specify runtime=nvidia.
  ```bash
  $ sudo docker run --runtime=nvidia ...
  ```

- Use nvidia-docker run.
  ```bash
  $ sudo nvidia-docker run ...
  ```

The new package provides backward compatibility, so you can still run GPU-accelerated containers by using this command, and the new runtime will be used.

Use docker run with nvidia as the default runtime.

You can set nvidia as the default runtime, for example, by adding the following line to the /etc/docker/daemon.json configuration file as the first entry.

```
"default-runtime": "nvidia",
```

The following is an example of how the added line appears in the JSON file. Do not remove any pre-existing content when making this change.

```
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
```
Running an NGC Container

You can then use docker run to run GPU-accelerated containers.

```
$ sudo docker run ...
```

**CAUTION:** If you build Docker images while `nvidia` is set as the default runtime, make sure the build scripts executed by the Dockerfile specify the GPU architectures that the container will need. Failure to do so may result in the container being optimized only for the GPU architecture on which it was built. Instructions for specifying the GPU architecture depend on the application and are beyond the scope of this document. Consult the specific application build process for guidance.

Using the Docker Engine Utility for NVIDIA GPUs

With the Docker Engine Utility for NVIDIA GPUs installed (nvidia-docker), run GPU-enabled containers as follows.

```
$ sudo nvidia-docker run ...
```

### 11.2. Running NGC Containers with Runtime Resources

On a system with GPU support for NGC containers, the following occurs when running a container.

- The Docker Engine loads the image into a container which runs the software.
- You define the runtime resources of the container by including additional flags and settings that are used with the command. These flags and settings are described in the following sections.
- The GPUs are explicitly defined for the Docker container (defaults to all GPUs, can be specified using `NV_GPU` environment variable).

**Note:** The base command `docker run --gpu all` assumes that your system has Docker 19.03-CE installed. See the section Enabling GPU Support for NGC Containers for the command to use for earlier versions of Docker.

1. As a user, run the container interactively.

```
$ docker run --gpus all -it --rm -v local_dir:container_dir
nvcr.io/nvidia/<repository>:<xx.xx>
```

The following example runs the December 2016 release (16.12) of the NVCAFFE container in interactive mode. The container is automatically removed when the user exits the container.

```
$ docker run --gpus all --rm -ti nvcr.io/nvidia/caffe:16.12
```

-----------
== Caffe ==
-----------
Running an NGC Container

2. From within the container, start the job that you want to run.

   The precise command to run depends on the deep learning framework in the container that you are running and the job that you want to run. For details see the /workspace/README.md file for the container.

   The following example runs the `caffe time` command on one GPU to measure the execution time of the `deploy.prototxt` model.

   ```bash
   # caffe time -model models/bvlc_alexnet/ -solver deploy.prototxt -gpu=0
   ```

3. Optional: Run the December 2016 release (16.12) of the same NVCaffe container but in non-interactive mode.

   ```bash
   % docker run --gpus all --rm nvcr.io/nvidia/caffe:16.12 caffe time -model /workspace/models/bvlc_alexnet -solver /workspace/deploy.prototxt -gpu=0
   ```

11.2.1. Specifying A User

   Unless otherwise specified, the user inside the container is the root user.

   When running within the container, files created on the host operating system or network volumes can be accessed by the root user. This is unacceptable for some users and they will want to set the ID of the user in the container. For example, to set the user in the container to be the currently running user, issue the following:

   ```bash
   % docker run --gpus all -ti --rm -u $(id -u):$(id -g) nvcr.io/nvidia/<repository>:<tag>
   ```

   Typically, this results in warnings due to the fact that the specified user and group do not exist in the container. You might see a message similar to the following:

   ```bash
   groups: cannot find name for group ID 1000 I have no name! @c177b61e5a93:/workspace$
   ```

   The warning can usually be ignored.

11.2.2. Setting The Remove Flag

   By default, Docker containers remain on the system after being run. Repeated pull or run operations use up more and more space on the local disk, even after exiting the container. Therefore, it is important to clean up the nvidia-docker containers after exiting.

   ```bash
   Note: Do not use the --rm flag if you have made changes to the container that you want to save, or if you want to access job logs after the run finishes.
   ```

   To automatically remove a container when exiting, add the --rm flag to the run command.

   ```bash
   % docker run --gpus all --rm nvcr.io/nvidia/<repository>:<tag>
   ```

11.2.3. Setting The Interactive Flag
By default, containers run in batch mode; that is, the container is run once and then exited without any user interaction. Containers can also be run in interactive mode as a service.

To run in interactive mode, add the `-ti` flag to the run command.

```
% docker run --gpus all -ti --rm nvcr.io/nvidia/<repository>:<tag>
```

### 11.2.4. Setting The Volumes Flag

There are no data sets included with the containers, therefore, if you want to use data sets, you need to mount volumes into the container from the host operating system. For more information, see Manage data in containers.

Typically, you would use either Docker volumes or host data volumes. The primary difference between host data volumes and Docker volumes is that Docker volumes are private to Docker and can only be shared amongst Docker containers. Docker volumes are not visible from the host operating system, and Docker manages the data storage. Host data volumes are any directory that is available from the host operating system. This can be your local disk or network volumes.

**Example 1**

Mount a directory `/raid/imagedata` on the host operating system as `/images` in the container.

```
% docker run --gpus all -ti --rm -v /raid/imagedata:/images nvcr.io/nvidia/<repository>:<tag>
```

**Example 2**

Mount a local docker volume named `data` (must be created if not already present) in the container as `/imagedata`.

```
% docker run --gpus all -ti --rm -v data:/imagedata nvcr.io/nvidia/<repository>:<tag>
```

### 11.2.5. Setting The Mapping Ports Flag

Applications such as Deep Learning GPU Training System™ (DIGITS) open a port for communications. You can control whether that port is open only on the local system or is available to other computers on the network outside of the local system.

Using DIGITS as an example, in DIGITS 5.0 starting in container image 16.12, by default the DIGITS server is open on port 5000. However, after the container is started, you may not easily know the IP address of that container. To know the IP address of the container, you can choose one of the following ways:

- Expose the port using the local system network stack (`--net=host`) where port 5000 of the container is made available as port 5000 of the local system.

or

- Map the port (`-p 8080:5000`) where port 5000 of the container is made available as port 8080 of the local system.

In either case, users outside the local system have no visibility that DIGITS is running in a container. Without publishing the port, the port is still available from the host, however not from the outside.
11.2.6. Setting The Shared Memory Flag

Certain applications, such as PyTorch™ and the Microsoft® Cognitive Toolkit™, use shared memory buffers to communicate between processes. Shared memory can also be required by single process applications, such as MXNet™ and TensorFlow™, which use the NVIDIA® Collective Communications Library™ (NCCL) (NCCL).

By default, Docker containers are allotted 64MB of shared memory. This can be insufficient, particularly when using all 8 GPUs. To increase the shared memory limit to a specified size, for example 1GB, include the `--shm-size=1g` flag in your `docker run` command.

Alternatively, you can specify the `--ipc=host` flag to re-use the host’s shared memory space inside the container. Though this latter approach has security implications as any data in shared memory buffers could be visible to other containers.

11.2.7. Setting The Restricting Exposure Of GPUs Flag

From inside the container, the scripts and software are written to take advantage of all available GPUs. To coordinate the usage of GPUs at a higher level, you can use this flag to restrict the exposure of GPUs from the host to the container. For example, if you only want GPU 0 and GPU 1 to be seen in the container, you would issue the following:

Using native GPU support

```bash
$ docker run --gpus "device=0,1" ...
```

Using `nvidia-docker2`

```bash
$ NV_GPU=0,1 docker run --runtime=nvidia ...
```

Using `nvidia-docker`

```bash
$ NV_GPU=0,1 nvidia-docker run ...
```

This flag creates a temporary environment variable that restricts which GPUs are used. Specified GPUs are defined per container using the Docker device-mapping feature, which is currently based on Linux cgroups.

11.2.8. Container Lifetime

The state of an exited container is preserved indefinitely if you do not pass the `--rm` flag to the `docker run` command. You can list all of the saved exited containers and their size on the disk with the following command:

```bash
$ docker ps --all --size --filter Status=exited
```

The container size on the disk depends on the files created during the container execution, therefore the exited containers take only a small amount of disk space.

You can permanently remove a exited container by issuing:

`docker rm [CONTAINER ID]`
By saving the state of containers after they have exited, you can still interact with them using the standard Docker commands. For example:

- You can examine logs from a past execution by issuing the `docker logs` command.
  
  ```bash
  $ docker logs 9489d47a054e
  ```

- You can extract files using the `docker cp` command.
  
  ```bash
  $ docker cp 9489d47a054e:/log.txt .
  ```

- You can restart a stopped container using the `docker restart` command.
  
  ```bash
  $ docker restart <container name>
  ```

For the NVCaffe™ container, issue this command:

```bash
$ docker restart caffe
```

- You can save your changes by creating a new image using the `docker commit` command.

For more information, see Example 3: Customizing a Container using...

**Note:** Use care when committing Docker container changes, as data files created during use of the container will be added to the resulting image. In particular, core dump files and logs can dramatically increase the size of the resulting image.
Chapter 12. Multi-Architecture Support for NGC Container Images

The NGC Container Registry allows users to leverage docker multi-architecture. It can support multiple architectures, which means that a single image may contain variants for different CPU architectures like ARM, x86, IBM POWER and others; and sometimes for different operating systems, such as Windows. When running an image, docker will automatically select an image variant which matches the target deployment OS and architecture.

Manifest Lists and Tags

The NGC Container Registry supports the manifest list schema application/vnd.docker.distribution.manifest.list.v2+json providing the ability to assign multiple tags per image. To inspect the manifest list, please follow the instructions here.

NGC UI allows users to easily identify multi-architecture containers and see the supported CPU architectures.

Basic Docker Commands for Multi-architecture Images

**Inspect Manifest**: Display an image manifest or manifest list

```
$ docker manifest inspect <container_name>:<tag>
```

More help on manifest: [link](#)

**Pull a specific image for an architecture**
In order to pull an image with a specific architecture, first do docker manifest which lists multiple platforms and then pull with a specific platform name that matches with the manifest digest.

$ docker pull --platform=<arch>

More help on the above docker command: [link]
Chapter 13. Examples of Using NGC Containers

NGC images come prepackaged, tuned, and ready to run; however, you may want to build a new image from scratch or augment an existing image with custom code, libraries, data, or settings for your corporate infrastructure. This section will guide you through exercises that will highlight how to create a container from scratch, customize a container, extend a deep learning framework to add features, develop some code using that extended framework from the developer environment, then package that code as a versioned release.

By default, you do not need to build a container. The NGC container registry, nvcr.io, has a number of containers that can be used immediately. These include containers for deep learning, scientific computing and visualization, as well as containers with just the CUDA Toolkit.

One of the great things about containers is that they can be used as starting points for creating new containers. This can be referred to as “customizing” or “extending” a container. You can create a container completely from scratch, however, since these containers are likely to run on a GPU system, it is recommended that you are least start with a nvcr.io container that contains the OS and CUDA. However, you are not limited to this and can create a container that runs on the CPUs in the system which does not use the GPUs. In this case, you can start with a bare OS container from Docker. However, to make development easier, you can still start with a container with CUDA - it is just not used when the container is used.

If you are running a DGX Station, you can push or save your modified/extended containers to the NGC Private Registry. This makes it easy to share with other users of the DGX system but this requires some administrator help. Details on how to get a private registry can be found in the document ../ngc/ngc-private-registry-user-guide/

Currently, you cannot save customized containers from the NGC container registry (cloud based) solution to nvcr.io. The customized or extended containers can be saved to a user’s private container repository. The customized or extended containers can be saved to a user’s private container repository.

It is important to note that all NGC deep learning framework images include the source to build the framework itself as well as all of the prerequisites.

ATTENTION: Do not install an NVIDIA driver into the Docker® image at Docker build time. nvidia-docker is essentially a wrapper around Docker that transparently provisions a container with the necessary components to execute code on the GPU.
NVIDIA provides a large set of images in the NGC container registry that are already tested, tuned, and are ready to run. You can pull any one of these images to create a container and add software or data of your choosing.

A best-practice is to avoid `docker commit` usage for developing new docker images, and to use Dockerfiles instead. The Dockerfile method provides visibility and capability to efficiently version-control changes made during development of a docker image. The `docker commit` method is appropriate for short-lived, disposable images only (see Example 3: Customizing A Container Using `docker commit` for an example).

For more information on writing a Docker file, see the best practices documentation.

13.1. Benefits And Limitations To Customizing A Container

You can customize a container to fit your specific needs for numerous reasons; for example, you depend upon specific software that is not included in the container that NVIDIA provides. No matter your reasons, you can customize a container.

The container images do not contain sample data-sets or sample model definitions unless they are included with the framework source. Be sure to check the container for sample data-sets or models.

13.2. Example 1: Building A Container From Scratch

Docker uses Dockerfiles to create or build a Docker image. Dockerfiles are scripts that contain commands that Docker uses successively to create a new Docker image. Simply put, a Dockerfile is the source code for the container image. Dockerfiles always start with a base image to inherit from.

For more information, see Best practices for writing Dockerfiles.

1. Create a working directory on your local hard-drive.
2. In that directory, open a text editor and create a file called `Dockerfile`. Save the file to your working directory.
3. Open your `Dockerfile` and include the following:

```bash
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y curl
CMD echo "hello from inside a container"
```

Where the last line `CMD`, executes the indicated command when creating the container. This is a way to check that the container was built correctly.

For this example, we are also pulling the container from the Docker repository and not the DGX™ system repository. There will be subsequent examples using the NVIDIA® repository.
4. Save and close your Dockerfile.

5. Build the image. Issue the following command to build the image and create a tag.

   ```
   $ docker build -t <new_image_name>:<new_tag> .
   ```

   **Note:** This command was issued in the same directory where the Dockerfile is located.

   The output from the docker build process lists "Steps"; one for each line in the Dockerfile.

   For example, let us name the container `test1` and tag it with `latest`. Also, for illustrative purposes, let us assume our private DGX system repository is called `nvidian_sas`. The command below builds the container. Some of the output is shown below so you know what to expect.

   ```
   $ docker build -t test1:latest .
   Sending build context to Docker daemon 3.072 kB
   Step 1/3 : FROM ubuntu:14.04
   14.04: Pulling from library/ubuntu
   ...
   Step 2/3 : RUN apt-get update && apt-get install -y curl
   ...
   Step 3/3 : CMD echo "hello from inside a container"
   ---> Running in 1f491b9235d8
   ---> 934785072daf
   Removing intermediate container 1f491b9235d8
   Successfully built 934785072daf
   ```

   For information about building your image, see `docker build`. For information about tagging your image, see `docker tag`.

6. Verify that the build was successful. You should see a message similar to the following:

   ```
   Successfully built  934785072daf
   ```

   This message indicates that the build was successful. Any other message and the build was not successful.

   **Note:** The number, 934785072daf, is assigned when the image is built and is random.

7. Confirm you can view your image by issuing the following command and view your container.

   ```
   $ docker images
   REPOSITORY TAG IMAGE ID CREATED SIZE
   test1 latest 934785072daf 19 minutes ago 222 MB
   ```

   The new container is now available to be used.

   **Note:** The container is local to this DGX system. If you want to store the container in your private repository, follow the next step.

8. Store the container in your private Docker repository by pushing it.

   **Note:** This applies if you have a private registry account associated with the DGX system purchased by your organization.

   a). The first step in pushing it, is to tag it.

   ```
   $ docker tag test1 nvcr.io/nvidian_sas/test1:latest
   ```

   b). Now that the image has been tagged, you can push it to, for example, a private project on nvcr.io named nvidian_sas.

   ```
   $ docker push nvcr.io/nvidian_sas/test1:latest
   ```
The push refers to a repository [nvcr.io/nvidian_sas/test1]

c). Verify that the container appears in the nvidian_sas repository.

13.3. Example 2: Customizing A Container Using Dockerfile

This example uses a Dockerfile to customize the NVCaffe container in nvcr.io. Before customizing the container, you should ensure the NVCaffe 17.03 container has been loaded into the registry using the docker pull command before proceeding.

$ docker pull nvcr.io/nvidia/caffe:17.03

As mentioned earlier in this document, the Docker containers on nvcr.io also provide a sample Dockerfile that explains how to patch a framework and rebuild the Docker image. In the directory /workspace/docker-examples, there are two sample Dockerfiles. For this example, we will use the Dockerfile.customcaffe file as a template for customizing a container.

1. Create a working directory called my_docker_images on your local hard drive.
2. Open a text editor and create a file called Dockerfile. Save the file to your working directory.
3. Open your Dockerfile again and include the following lines in the file:

```bash
FROM nvcr.io/nvidia/caffe:17.03
# APPLY CUSTOMER PATCHES TO CAFFE
# Bring in changes from outside container to /tmp
# (assumes my-caffe-modifications.patch is in same directory as Dockerfile)
COPY my-caffe-modifications.patch /tmp

WORKDIR /opt/caffe
# Apply modifications
RUN patch -p1 < /tmp/my-caffe-modifications.patch
RUN mkdir build && cd build && \
    cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr/local -DUSE_NCCL=ON \
    -DUSE_CUDNN=ON -DCUDA_ARCH_NAME=Manual -DCUDA_ARCH_BIN="35 52 60 61" \
    -DCUDA_ARCH_PTX="61" .. && \
    make -j"$(nproc)" install && \
    make clean && \
    cd .. && rm -rf build

WORKDIR /workspace
```

Save the file.

4. Build the image using the docker build command and specify the repository name and tag. In the following example, the repository name is corp/caffe and the tag is 17.03.1PlusChanges. For the case, the command would be the following:

$ docker build -t corp/caffe:17.03.1PlusChanges .
5. Run the Docker image using the command appropriate to the method of GPU support installed.

$ docker run --gpus all -ti --rm corp/caffe:17.03.1PlusChanges .

Note: The base command docker run --gpu all assumes that your system has Docker 19.03-CE installed. See the section Enabling GPU Support for NGC Containers for the command to use for earlier versions of Docker.

13.4. Example 3: Customizing A Container

Using docker commit

This example uses the docker commit command to flush the current state of the container to a Docker image. This is not a recommended best practice, however, this is useful when you have a container running to which you have made changes and want to save them. In this example, we are using the apt-get tag to install packages which requires that the user run as root.

Note:
- The NVCaffe image release 17.04 is used in the example instructions for illustrative purposes.
- Do not use the --rm flag when running the container. If you use the --rm flag when running the container, your changes will be lost when exiting the container.

1. Pull the Docker container from the nvcr.io repository to the DGX system. For example, the following command will pull the NVCaffe container:

$ docker pull nvcr.io/nvidia/caffe:17.04

2. Run the container on the DGX system.

$ docker run --gpus all -ti nvcr.io/nvidia/caffe:17.04

===============
NVIDIA Caffe ==
===============

NVIDIA Release 17.04 (build 26740)

Container image Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved. Copyright (c) 2014, 2015, The Regents of the University of California (Regents) All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION. All rights reserved. NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

NOTE: The SHMEM allocation limit is set to the default of 64MB. This may be insufficient for NVIDIA Caffe. NVIDIA recommends the use of the following flags:

nvidia-docker run --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 ...


Examples of Using NGC Containers

3. You should now be the root user in the container (notice the prompt). You can use the command `apt` to pull down a package and put it in the container.

**Note:** The NVIDIA containers are built using Ubuntu which uses the `apt-get` package manager. Check the container release notes Deep Learning Documentation for details on the specific container you are using.

In this example, we will install Octave; the GNU clone of MATLAB, into the container.

```
# apt-get update
# apt install octave
```

**Note:** You have to first issue `apt-get update` before you install Octave using `apt`.

4. Exit the workspace.

```
# exit
```

5. Display the list of containers using `docker ps -a`. As an example, here is some of the output from the `docker ps -a` command:

```
$ docker ps -a
CONTAINER ID    IMAGE                        CREATED       ...
1fe228556a97    nvcr.io/nvidia/caffe:17.04   3 minutes ago ...
```

6. Now you can create a new image from the container that is running where you have installed Octave. You can commit the container with the following command.

```
$ docker commit 1fe228556a97 nvcr.io/nvidian_sas/caffe_octave:17.04
```

7. Display the list of images.

```
$ docker images
REPOSITORY                  TAG              IMAGE ID     ...
nvidian_sas/caffe_octave    17.04            75211f8ec225 ...
```

8. To verify, let’s run the container again and see if Octave is actually there.

**Note:** This only works for the DGX-1 and the DGX Station.

```
$ docker run --gpus all -ti nvidian_sas/caffe_octave:17.04
==================
== NVIDIA Caffe ==
==================
NVIDIA Release 17.04 (build 26740)
```

Container image Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved. Copyright (c) 2014, 2015, The Regents of the University of California (Regents) All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION. All rights reserved. NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

**NOTE:** The SHMEM allocation limit is set to the default of 64MB. This may be insufficient for NVIDIA Caffe. NVIDIA recommends the use of the following flags:
**Examples of Using NGC Containers**

```
nvidia-docker run --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 ...
```

root@2fc3608ad9d8:/workspace# octave
octave: X11 DISPLAY environment variable not set
octave: disabling GUI features
GNU Octave, version 4.0.0
Copyright (C) 2015 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at http://www.octave.org.
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

octave:1>
```

Since the Octave prompt displayed, Octave is installed.

9. If you want to save the container into your private repository (Docker uses the phrase “push”), then you can use the command `docker push` ....

```
$ docker push nvcr.io/nvidian_sas/caffe_octave:17.04
```

The new Docker image is now available for use. You can check your local Docker repository for it.

### 13.5. Example 4: Developing A Container Using Docker

There are two primary use cases for a developer to extend a container:

1. Create a development image that contains all of the immutable dependencies for the project, but not the source code itself.
2. Create a production or testing image that contains a fixed version of the source and all of the software dependencies.

The datasets are not packaged in the container image. Ideally, the container image is designed to expect volume mounts for datasets and results.

In these examples, we mount our local dataset from `/raid/datasets` on our host to `/dataset` as a read-only volume inside the container. We also mount a job specific directory to capture the output from a current run.

In these examples, we will create a timestamped output directory on each container launch and map that into the container at `/output`. Using this method, the output for each successive container launch is captured and isolated.
Examples of Using NGC Containers

Including the source into a container for developing and iterating on a model has many awkward challenges that can over complicate the entire workflow. For instance, if your source code is in the container, then your editor, version control software, dotfiles, etc. also need to be in the container.

However, if you create a development image that contains everything you need to run your source code, you can map your source code into the container to make use of your host workstation’s developer environment. For sharing a fixed version of a model, it is best to package a versioned copy of the source code and trained weights with the development environment.

As an example, we will work through a development and delivery example for the open source implementation of the work found in [Image-to-Image Translation with Conditional Adversarial Networks](https://arxiv.org/abs/1611.07004) by Isola et. al. and is available at pix2pix. Pix2Pix is a Torch implementation for learning a mapping from input images to output images using a Conditional Adversarial Network. Since online projects can change over time, we will focus our attention on the snapshot version `d7e7b8b557229e75140cbe42b7f5dbf85a67d097` change-set.

In this section, we are using the container as a virtual environment, in that the container has all the programs and libraries needed for our project.

Note: We have kept the network definition and training script separate from the container image. This is a useful model for iterative development because the files that are actively being worked on are persistent on the host and only mapped into the container at runtime.

The differences to the original project can be found here [Comparing changes](https://github.com/phillipi/pix2pix/compare/d7e7b8b557229e75140cbe42b7f5dbf85a67d097...master).

If the machine you are developing on is not the same machine on which you will be running long training sessions, then you may want to package your current development state in the container.

1. Create a working directory on your local hard-drive.
   ```bash
   mkdir Projects
   $ cd ~/Projects
   ```
2. Git clone the Pix2Pix git repository.
   ```bash
   $ git clone https://github.com/phillipi/pix2pix.git
   $ cd pix2pix
   ```
3. Run the git checkout command.
   ```bash
   $ git checkout -b devel d7e7b8b557229e75140cbe42b7f5dbf85a67d097
   ```
4. Download the dataset:
   ```bash
   bash ./datasets/download_dataset.sh facades
   ``
   I want to put the dataset on my fast /raid storage.
   ```bash
   $ mkdir -p /raid/datasets
   $ mv ./datasets/facades /raid/datasets
   ```
5. Create a file called Dockerfile, and add the following lines:
   ```bash
   FROM nvcr.io/nvidia/torch:17.03
   RUN luarocks install nngraph
   RUN luarocks install https://raw.githubusercontent.com/szym/display/master/display-scm-0.rockspec
   ```
Examples of Using NGC Containers

WORKDIR /source

6. Build the development Docker container image (build-devel.sh).
   
   docker build -t nv/pix2pix-torch:devel .

7. Create the following train.sh script:

   ```bash
   #!/bin/bash -x
   ROOT="${ROOT:-/source}"
   DATASET="${DATASET:-facades}"
   DATA_ROOT="${DATA_ROOT:-/datasets/$DATASET}"
   DATA_ROOT=$DATA_ROOT name="${DATASET}_generation"
   which_direction=BtoA th train.lua
   ```

   If you were actually developing this model, you would be iterating by making changes to the files on the host and running the training script which executes inside the container.

8. **Optional:** Edit the files and execute the next step after each change.

9. Run the training script (run-devel.sh).

   ```bash
   docker run --gpus all --rm -ti -v $PWD:/source -v /raid/datasets:/datasets nv/pix2pix-torch:devel ./train.sh
   ```

**Note:** The base command docker run --gpu all assumes that your system has Docker 19.03-CE installed. See the section Enabling GPU Support for NGC Containers for the command to use for earlier versions of Docker.

13.5.1. Example 4.1: Package The Source Into The Container

Packaging the model definition and script into the container is very simple. We simply add a `COPY` step to the Dockerfile.

We’ve updated the run script to simply drop the volume mounting and use the source packaged in the container. The packaged container is now much more portable than our devel container image because the internal code is fixed. It would be good practice to version control this container image with a specific tag and store it in a container registry.

The updates to run the container are equally subtle. We simply drop the volume mounting of our local source into the container.
Chapter 14. Customizing Containers

NGC images come prepackaged, tuned, and ready to run; however, you may want to build a new image from scratch or augment an existing image with custom code, libraries, data, or settings for your corporate infrastructure. This section will guide you through exercises that will highlight how to create a container from scratch, customize a container, extend a deep learning framework to add features, develop some code using that extended framework from the developer environment, then package that code as a versioned release.

By default, you do not need to build a container. The NGC container registry, nvcr.io, has a number of containers that can be used immediately. These include containers for deep learning, scientific computing and visualization, as well as containers with just the CUDA Toolkit.

One of the great things about containers is that they can be used as starting points for creating new containers. This can be referred to as “customizing” or “extending” a container. You can create a container completely from scratch, however, since these containers are likely to run on a GPU system, it is recommended that you are least start with a nvcr.io container that contains the OS and CUDA. However, you are not limited to this and can create a container that runs on the CPUs in the system which does not use the GPUs. In this case, you can start with a bare OS container from Docker. However, to make development easier, you can still start with a container with CUDA - it is just not used when the container is used.

In the case of the DGX-1 and the DGX Station, you can push or save your modified/extended containers to the NVIDIA DGX container registry, nvcr.io. They can also be shared with other users of the DGX system but this requires some administrator help.

Currently, you cannot save customized containers from the NGC container registry (cloud based) solution to nvcr.io. The customized or extended containers can be saved to a user’s private container repository. The customized or extended containers can be saved to a user’s private container repository.

It is important to note that all NGC deep learning framework images include the source to build the framework itself as well as all of the prerequisites.

**ATTENTION:** Do not install an NVIDIA driver into the Docker® image at Docker build time. nvidia-docker is essentially a wrapper around Docker that transparently provisions a container with the necessary components to execute code on the GPU.

NVIDIA provides a large set of images in the NGC container registry that are already tested, tuned, and are ready to run. You can pull any one of these images to create a container and add software or data of your choosing.
A best-practice is to avoid `docker commit` usage for developing new docker images, and to use Dockerfiles instead. The Dockerfile method provides visibility and capability to efficiently version-control changes made during development of a docker image. The `docker commit` method is appropriate for short-lived, disposable images only (see Example 3: Customizing A Container Using `docker commit` for an example).

For more information on writing a Docker file, see the best practices documentation.

14.1. Benefits And Limitations To Customizing A Container

You can customize a container to fit your specific needs for numerous reasons; for example, you depend upon specific software that is not included in the container that NVIDIA provides. No matter your reasons, you can customize a container.

The container images do not contain sample data-sets or sample model definitions unless they are included with the framework source. Be sure to check the container for sample data-sets or models.

14.2. Example 1: Building A Container From Scratch

Docker uses Dockerfiles to create or build a Docker image. Dockerfiles are scripts that contain commands that Docker uses successively to create a new Docker image. Simply put, a Dockerfile is the source code for the container image. Dockerfiles always start with a base image to inherit from.

For more information, see Best practices for writing Dockerfiles.

1. Create a working directory on your local hard-drive.
2. In that directory, open a text editor and create a file called `Dockerfile`. Save the file to your working directory.
3. Open your `Dockerfile` and include the following:

```bash
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y curl
CMD echo "hello from inside a container"
```

Where the last line `CMD`, executes the indicated command when creating the container. This is a way to check that the container was built correctly.

For this example, we are also pulling the container from the Docker repository and not the DGX™ system repository. There will be subsequent examples using the NVIDIA® repository.

4. Save and close your `Dockerfile`.
5. Build the image. Issue the following command to build the image and create a tag.
$ docker build -t <new_image_name>:<new_tag> .

Note: This command was issued in the same directory where the Dockerfile is located.
The output from the docker build process lists "Steps"; one for each line in the Dockerfile.
For example, let us name the container test1 and tag it with latest. Also, for illustrative purposes, let us assume our private DGX system repository is called nvidian_sas. The command below builds the container. Some of the output is shown below so you know what to expect.

$ docker build -t test1:latest .
Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM ubuntu:14.04
14.04: Pulling from library/ubuntu
... Step 2/3 : RUN apt-get update && apt-get install -y curl
... Step 3/3 : CMD echo "hello from inside a container"
 ---> Running in 1f491b9235d8
 ---> 934785072daf
Removing intermediate container 1f491b9235d8
Successfully built 934785072daf

For information about building your image, see docker build. For information about tagging your image, see docker tag.

6. Verify that the build was successful. You should see a message similar to the following:
Successfully built 934785072daf
This message indicates that the build was successful. Any other message and the build was not successful.

Note: The number, 934785072daf, is assigned when the image is built and is random.

7. Confirm you can view your image by issuing the following command and view your container.

$ docker images
REPOSITORY      TAG            IMAGE ID        CREATED                SIZE
test1           latest         934785072daf    19 minutes ago         222 MB

The new container is now available to be used.

Note: The container is local to this DGX system. If you want to store the container in your private repository, follow the next step.

8. Store the container in your private Docker repository by pushing it.

Note: This applies if you have a private registry account associated with the DGX system purchased by your organization.

a). The first step in pushing it, is to tag it.

$ docker tag test1 nvcr.io/nvidian_sas/test1:latest

b). Now that the image has been tagged, you can push it to, for example, a private project on nvcr.io named nvidian_sas.

$ docker push nvcr.io/nvidian_sas/test1:latest
The push refers to a repository [nvcr.io/nvidian_sas/test1]

...
c) Verify that the container appears in the nvidia_sas repository.

14.3. Example 2: Customizing A Container Using Dockerfile

This example uses a Dockerfile to customize the NVCaffe container in nvcr.io. Before customizing the container, you should ensure the NVCaffe 17.03 container has been loaded into the registry using the docker pull command before proceeding.

```
$ docker pull nvcr.io/nvidia/caffe:17.03
```

As mentioned earlier in this document, the Docker containers on nvcr.io also provide a sample Dockerfile that explains how to patch a framework and rebuild the Docker image. In the directory /workspace/docker-examples, there are two sample Dockerfiles. For this example, we will use the Dockerfile.customcaffe file as a template for customizing a container.

1. Create a working directory called my_docker_images on your local hard drive.
2. Open a text editor and create a file called Dockerfile. Save the file to your working directory.
3. Open your Dockerfile again and include the following lines in the file:

```
FROM nvcr.io/nvidia/caffe:17.03
# APPLY CUSTOMER PATCHES TO CAFFE
# Bring in changes from outside container to /tmp
# (assumes my-caffe-modifications.patch is in same directory as Dockerfile)
#COPY my-caffe-modifications.patch /tmp

# Change working directory to NVCaffe source path
WORKDIR /opt/caffe

# Apply modifications
#RUN patch -p1 < /tmp/my-caffe-modifications.patch
# Note that the default workspace for caffe is /workspace
RUN mkdir build && cd build && \
    cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr/local -DUSE_NCCL=ON \
    -DUSE_CUDNN=ON -DCUDA_ARCH_NAME=Manual -DCUDA_ARCH_BIN="35 52 60 61" \
    -DCUDA_ARCH_PTX="61" ... && \
    make -j$(nproc) " install && \
    make clean && \
    cd .. && rm -rf build

# Reset default working directory
WORKDIR /workspace
```

Save the file.

4. Build the image using the docker build command and specify the repository name and tag. In the following example, the repository name is corp/caffe and the tag is 17.03.1PlusChanges. For this case, the command would be the following:

```
$ docker build -t corp/caffe:17.03.1PlusChanges .
```

5. Run the Docker image using the command appropriate to the method of GPU support installed.
$ docker run --gpus all -ti --rm corp/caffe:17.03.1PlusChanges.

Note: The base command `docker run --gpus all` assumes that your system has Docker 19.03-CE installed. See the section Enabling GPU Support for NGC Containers for the command to use for earlier versions of Docker.

14.4. Example 3: Customizing A Container Using `docker commit`

This example uses the `docker commit` command to flush the current state of the container to a Docker image. This is not a recommended best practice, however, this is useful when you have a container running to which you have made changes and want to save them. In this example, we are using the `apt-get` tag to install packages which requires that the user run as root.

Note:

- The NVCAffe image release 17.04 is used in the example instructions for illustrative purposes.
- Do not use the `--rm` flag when running the container. If you use the `--rm` flag when running the container, your changes will be lost when exiting the container.

1. Pull the Docker container from the `nvcr.io` repository to the DGX system. For example, the following command will pull the NVCAffe container:

   ```bash
   $ docker pull nvcr.io/nvidia/caffe:17.04
   ```

2. Run the container on the DGX system.

   ```bash
   $ docker run --gpus all -ti nvcr.io/nvidia/caffe:17.04
   ```

   ================
   == NVIDIA Caffe ==
   ================

   NVIDIA Release 17.04 (build 26740)

   Container image Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved.
   Copyright (c) 2014, 2015, The Regents of the University of California (Regents)
   All rights reserved.

   Various files include modifications (c) NVIDIA CORPORATION. All rights reserved.
   NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

   NOTE: The SHMEM allocation limit is set to the default of 64MB. This may be insufficient for NVIDIA Caffe. NVIDIA recommends the use of the following flags:
   - `nvidia-docker run --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 ...`
Note: The base command `docker run --gpu all` assumes that your system has Docker 19.03-CE installed. See the section Enabling GPU Support for NGC Containers for the command to use for earlier versions of Docker.

3. You should now be the root user in the container (notice the prompt). You can use the command `apt` to pull down a package and put it in the container.

Note: The NVIDIA containers are built using Ubuntu which uses the `apt-get` package manager. Check the container release notes Deep Learning Documentation for details on the specific container you are using.

In this example, we will install Octave; the GNU clone of MATLAB, into the container.

```
# apt-get update
# apt install octave
```

Note: You have to first issue `apt-get update` before you install Octave using `apt`.

4. Exit the workspace.

```
# exit
```

5. Display the list of containers using `docker ps -a`. As an example, here is some of the output from the `docker ps -a` command:

```
CONTAINER ID    IMAGE                        CREATED       ...
1fe228556a97    nvcr.io/nvidia/caffe:17.04   3 minutes ago ...
```

6. Now you can create a new image from the container that is running where you have installed Octave. You can commit the container with the following command.

```
$ docker commit 1fe228556a97 nvcr.io/nvidia/caffe_octave:17.04
```

7. Display the list of images.

```
$ docker images
```

```
REPOSITORY                  TAG              IMAGE ID     ...
nvidian_sas/caffe_octave    17.04            75211f8ec225 ...
```

8. To verify, let’s run the container again and see if Octave is actually there.

Note: This only works for the DGX-1 and the DGX Station.

```
$ docker run --gpus all -ti nvidian_sas/caffe_octave:17.04
```

```
==================
== NVIDIA Caffe  
==================
NVIDIA Release 17.04 (build 26740)

Container image Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved. Copyright (c) 2014, 2015, The Regents of the University of California (Regents) All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION. All rights reserved. NVIDIA modifications are covered by the license terms that apply to the underlying project or file.

NOTE: The SHMEM allocation limit is set to the default of 64MB. This may be insufficient for NVIDIA Caffe. NVIDIA recommends the use of the following flags:
nvidia-docker run --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 ... 

root@2fc3608ad9d8:/workspace# octave
octave: X11 DISPLAY environment variable not set
octave: disabling GUI features
GNU Octave, version 4.0.0
Copyright (C) 2015 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at http://www.octave.org.
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

octave:1>

Since the Octave prompt displayed, Octave is installed.

9. If you want to save the container into your private repository (Docker uses the phrase
"push"), then you can use the command docker push ....
$ docker push nvcr.io/nvidian_sas/caffe_octave:17.04

The new Docker image is now available for use. You can check your local Docker repository for
it.

14.5. Example 4: Developing A Container
Using Docker

There are two primary use cases for a developer to extend a container:

1. Create a development image that contains all of the immutable dependencies for the
   project, but not the source code itself.
2. Create a production or testing image that contains a fixed version of the source and all of
   the software dependencies.

The datasets are not packaged in the container image. Ideally, the container image is designed
to expect volume mounts for datasets and results.

In these examples, we mount our local dataset from /raid/datasets on our host to /
dataset as a read-only volume inside the container. We also mount a job specific directory to
capture the output from a current run.

In these examples, we will create a timestamped output directory on each container launch
and map that into the container at /output. Using this method, the output for each successive
container launch is captured and isolated.
Including the source into a container for developing and iterating on a model has many awkward challenges that can over complicate the entire workflow. For instance, if your source code is in the container, then your editor, version control software, dotfiles, etc. also need to be in the container.

However, if you create a development image that contains everything you need to run your source code, you can map your source code into the container to make use of your host workstation’s developer environment. For sharing a fixed version of a model, it is best to package a versioned copy of the source code and trained weights with the development environment.

As an example, we will work though a development and delivery example for the open source implementation of the work found in Image-to-Image Translation with Conditional Adversarial Networks by Isola et. al. and is available at pix2pix. Pix2Pix is a Torch implementation for learning a mapping from input images to output images using a Conditional Adversarial Network. Since online projects can change over time, we will focus our attention on the snapshot version d7e7b8b557229e75140cbe42b7f5d6b85a67d097 change-set.

In this section, we are using the container as a virtual environment, in that the container has all the programs and libraries needed for our project.

Note: We have kept the network definition and training script separate from the container image. This is a useful model for iterative development because the files that are actively being worked on are persistent on the host and only mapped into the container at runtime.

The differences to the original project can be found here Comparing changes.

If the machine you are developing on is not the same machine on which you will be running long training sessions, then you may want to package your current development state in the container.

1. Create a working directory on your local hard-drive.
   ```
   mkdir Projects
   $ cd ~/Projects
   ```

2. Git clone the Pix2Pix git repository.
   ```
   $ git clone https://github.com/phillipi/pix2pix.git
   $ cd pix2pix
   ```

3. Run the `git checkout` command.
   ```
   $ git checkout -b devel d7e7b8b557229e75140cbe42b7f5d6b85a67d097
   ```

4. Download the dataset:
   ```
   bash ./datasets/download_dataset.sh facades
   ```
   I want to put the dataset on my fast /raid storage.
   ```
   $ mkdir -p /raid/datasets
   $ mv ./datasets/facades /raid/datasets
   ```

5. Create a file called `Dockerfile`, and add the following lines:
   ```
   FROM nvcr.io/nvidia/torch:17.03
   RUN luarocks install nngraph
   RUN luarocks install https://raw.githubusercontent.com/szym/display/master/display-scm-0.rockspec
   ```
WORKDIR /source

6. Build the development Docker container image (build-devel.sh).

   docker build -t nv/pix2pix-torch:devel .

7. Create the following train.sh script:

   ```bash
   #!/bin/bash -x
   ROOT="${ROOT:-/source}"
   DATASET="${DATASET:-facades}"
   DATA_ROOT="${DATA_ROOT:-/datasets/$DATASET}"
   DATA_ROOT=${DATA_ROOT} name="${DATASET}_generation"
   which_direction=BtoA th train.lua
   ```

   If you were actually developing this model, you would be iterating by making changes to the files on the host and running the training script which executes inside the container.

8. Optional: Edit the files and execute the next step after each change.

9. Run the training script (run-devel.sh).

   ```bash
   docker run --gpus all --rm -ti -v $PWD:/source  -v /raid/datasets:/datasets nv/pix2pix-torch:devel ./train.sh
   ```

   **Note:** The base command `docker run --gpu all` assumes that your system has Docker 19.03-CE installed. See the section [Enabling GPU Support for NGC Containers](#) for the command to use for earlier versions of Docker.

14.5.1. Example 4.1: Package The Source Into The Container

Packaging the model definition and script into the container is very simple. We simply add a `COPY` step to the Dockerfile.

We’ve updated the run script to simply drop the volume mounting and use the source packaged in the container. The packaged container is now much more portable than our `devel` container image because the internal code is fixed. It would be good practice to version control this container image with a specific tag and store it in a container registry.

The updates to run the container are equally subtle. We simply drop the volume mounting of our local source into the container.
Chapter 15. Models

A discussion of NGC models and how to download them from the NGC Catalog.

15.1. What are Models

Many AI applications have common needs: classification, object detection, language translation, text-to-speech, recommender engines, sentiment analysis, and more. When developing applications with these capabilities, it is much faster to start with a model that is pre-trained and then tune it for a specific use case.

NGC offers pre-trained models for a large number of AI tasks that are optimized for NVIDIA Tensor Core GPUs, and can be easily re-trained by updating just a few layers, saving valuable time.

So whether you’re looking for content you can simply retrain for your specific use case or a complete model you can grab and deploy right away, NGC has you covered.

15.2. Downloading Models via NGC CLI

A discussion of NGC models and how to download them from the NGC Catalog.

To download a model from the registry to your local disk, specify the model name and version.

$ ngc registry model download-version <org-name>/<model-name:version>

Example

$ ngc registry model download-version nvidia/ngcdocsmodel:1

The following is an example showing the output confirming completion of the download:

Downloaded 230.92 MB in 38s, Download speed: 6.07 MB/s
Transfer id: trt_onnx_vgg16_v100_16g_int8_v1 Download status: Completed.
Downloaded local path: C:\ngcdocsmodel
Total files downloaded: 3
Total downloaded size: 230.92 MB
Started at: 2019-03-18 14:09:31.664000
Completed at: 2019-03-18 14:10:09.712000
Duration taken: 38s seconds

The model is downloaded to a folder that corresponds to the model name in the current directory. You can specify another path using the -d . option.
**Example:** Downloading a model to a specific directory (/models).

```bash
$ nge registry model download-version nvidia/ngcdocsmoodel:1 -d ./models
```

Downloaded 230.92 MB in 38s, Download speed: 6.07 MB/s
Transfer id: trt_onnx_vgg16_v1.1_g_int8_v1 Download status: Completed.
Downloaded local path: C:\ngcdocsmoodel
Total files downloaded: 3
Total downloaded size: 230.92 MB
Started at: 2019-03-18 14:09:31.664000
Completed at: 2019-03-18 14:10:09.712000
Duration taken: 38s seconds

---

## 15.3. Downloading Models via WGET/cURL

How to download models via WGET/cURL.

To get a copy of a single file from an NGC Model via wget/curl you can use the URL provided in the NGC UI.

1. Open the NGC UI, click **Models** from the top content type ribbon and then locate and click the model you want to download.
2. Click the **File Browser** tab.
3. Click the appropriate version from the left menu,
4. From the **Actions** menu, select **Copy curl cmd** or **Copy wget cmd** to copy the relevant code to your clipboard.

The following diagram illustrates the steps.

The following will then be added to your clipboard:

```bash
```
15.4. **Downloading Models via Web UI**

How to download NGC models via the Web UI.

**Downloading the Latest Version of a Model**

To download the latest version of a model to your local machine, performing the following.

1. Open the NGC UI, click **Models** from the top content type ribbon and then locate and click the model you want to download.
2. Click the actions menu from the top right corner of the model detail page and then click **Download <version>**.

**Example**

The following diagram illustrates the steps.

**Downloading a Specific Version of a Model**

To download a specific version of a model, performing the following.

1. Open the NGC UI, click **Models** from the top content type ribbon and then locate and click the model you want to download.
2. Click the Version History tab, then click the actions icon for the version to download and then click **Download**.

The following diagram illustrates the steps.
BertBaseCasedForNeMo

Publisher: Hugging Face
Application: NLP
Framework: PaddlePaddle
Model: BERT

Description: BertBase Model trained on 9 corpora and fine-tuned on a sequence length of 128.

<table>
<thead>
<tr>
<th>Version</th>
<th>Accuracy</th>
<th>Epochs</th>
<th>Batch Size</th>
<th>GPU</th>
<th>Memory</th>
<th>File Size</th>
<th>Owner</th>
<th>Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>84.67</td>
<td>1</td>
<td>256</td>
<td>DGX1 8x32GB</td>
<td>-</td>
<td>465.8 GB</td>
<td>Nvidia</td>
<td>Feb 27, 2021</td>
</tr>
<tr>
<td>2</td>
<td>84.67</td>
<td>1</td>
<td>256</td>
<td>DGX1 8x32GB</td>
<td>-</td>
<td>919.8 GB</td>
<td>Nvidia</td>
<td>Feb 27, 2021</td>
</tr>
<tr>
<td>3</td>
<td>84.67</td>
<td>1</td>
<td>120</td>
<td>DGX1 32GB</td>
<td>25</td>
<td>413.2 GB</td>
<td>Nvidia</td>
<td>Feb 27, 2021</td>
</tr>
</tbody>
</table>
Chapter 16. Resources

Description of NGC resources, their purpose, and how to use them.

16.1. What are Resources

NGC offers documentation, code samples and a number of other assets such as deployment pipelines and step-by-step instructions and scripts for creating deep learning models. These scripts have sample performance and accuracy metrics that allow you to compare your results. These scripts provide expert guidance on building DL models for image classification, language translation, text-to-speech and more. Data scientists can quickly build performance-optimized models by easily adjusting the hyperparameters.

16.2. Downloading Resources

Description of NGC resources, their purpose, and how to use them.

To download a resource from the registry to your local disk, specify the resource name and version.

\$ ngc registry resource download-version nvidia/<resource-name:version>

**Example:** Downloading a resource to the current directory.

\$ ngc registry resource download-version nvidia/gnmt_v2_for_tensorflow:1

The following is an example showing the output confirming completion of the download:

Downloaded 130.6 KB in 1s, Download speed: 130.6 KB/s

Transfer id: gnmt_v2_for_tensorflow_v1 Download status: Completed.
Downloaded local path: C:\gnmt_v2_for_tensorflow_v1
Total files downloaded: 35
Total downloaded size: 130.6 KB
Started at: 2019-03-20 11:24:00.168000
Completed at: 2019-03-20 11:24:01.176000
Duration taken: 1s seconds

The model is downloaded to a folder that corresponds to the model name in the current directory. You can specify another path using the -d option.

**Example:** Downloading a resource to a specific directory (/resources).

\$ ngc registry resource download-version nvidia/gnmt_v2_for_tensorflow:1 -d ./resources
Downloaded 130.6 KB in 1s, Download speed: 130.6 KB/s
====================================
Transfer id: gnmt_v2_for_tensorflow_v1 Download status: Completed.
Downloaded local path: C:\resources\gnmt_v2_for_tensorflow_v1
Total files downloaded: 35
Total downloaded size: 130.6 KB
Started at: 2019-03-20 11:24:00.168000
Completed at: 2019-03-20 11:24:01.176000
Duration taken: 1s seconds
====================================
Chapter 17. Helm Charts

Helm is an application package manager running on top of Kubernetes. This section describes how to use access Helm charts from the NGC Catalog.

Prerequisites
These instructions assume the following prerequisites are met.

- Helm v 2.16.1 installed
  The helm push plugin does not support Helm v3 yet - make sure you are using v2.16.x.
  See https://github.com/helm/helm/releases/tag/v2.16.1 for the Helm download and installation instructions.
- NGC organization account
  See Setting Up and Activating Your NGC Account for instructions.

17.1. Setting Up an NGC Helm Repository

1. Obtain an NGC API Key.
   See Generating Your NGC API Key for instructions.
2. Export the API Key for use in commands.
   $ export NGC_API_KEY=<your-api-key>
3. Add the NGC org to your Helm repository.
   $ helm repo add <repo-name> https://helm.ngc.nvidia.com/<org-name> --username=
   $oauthtoken --password=$NGC_API_KEY
   Where <repo-name> is a name of your choosing by which you will reference the repository.

17.2. Searching for Available Helm Charts in the NGC Catalog

To view a list of available Chart packages in your org, issue the following.
17.3. Fetching Helm Charts

To download (or “fetch”) a Helm chart package from the repo, issue the following.

```
$ helm fetch <repo-name>/<chart-name>
```

17.4. Using the NGC Website to View the List of Helm Charts and Get Fetch Commands

From the NGC website you can

- View the contents of the Helm chart repository.
- Get the push command for a specific Helm chart in the repository.

2. If you are a member of more than one org, select the one that contains the Helm charts that you are interested in, then click Sign In.
3. Click Helm Charts from the left-side navigation pane.

The page presents cards for each available Helm chart.
4. Select one of the Helm chart cards.

The page for each Helm chart provides information about the chart.

The **Fetch Command** section shows the command to use for downloading the Helm chart package.

Click either the **Fetch** download button from the upper right corner or the copy icon next to the fetch command to copy the fetch command to the clipboard.

The **File Browser** tab lets you see the file content of the Helm chart package.
<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>ngocdoc-test</td>
<td>395B</td>
<td>8 days ago</td>
</tr>
<tr>
<td>templates</td>
<td>2kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>tests</td>
<td>2kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>test-connection.yaml</td>
<td>3kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>.helpers.tpl</td>
<td>1kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>deployment.yaml</td>
<td>2kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>ingress.yaml</td>
<td>1kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>NOTES.txt</td>
<td>2kB</td>
<td>8 days ago</td>
</tr>
<tr>
<td>service.yaml</td>
<td>373B</td>
<td>8 days ago</td>
</tr>
<tr>
<td>serviceaccount.yaml</td>
<td>215B</td>
<td>8 days ago</td>
</tr>
<tr>
<td>.helmignore</td>
<td>342B</td>
<td>8 days ago</td>
</tr>
<tr>
<td>Chart.yaml</td>
<td>108B</td>
<td>8 days ago</td>
</tr>
<tr>
<td>values.yaml</td>
<td>421B</td>
<td>8 days ago</td>
</tr>
</tbody>
</table>
Chapter 18. SDK and AI Toolkits

NVIDIA AI toolkits provide libraries and tools to train, fine-tune, optimize and deploy pre-trained NGC models across a broad domain of industries and AI applications.

These include

- An **AI-assisted annotation tool** to help users label their datasets for training.
- A **transfer learning toolkit** to fine-tune pre-trained models with user data, saving the cost of training from scratch.
- **Federated learning** that preserves privacy by allowing users to collaborate and train AI models without sharing private data between clients.
- The **NeMo toolkit** to quickly build state-of-the-art models for speech recognition and natural language processing.
- The Service Maker toolkit that exposes trained models as a gRPC service that can be scaled and easily deployed on a Kubernetes service.
- Models built using the toolkits can be integrated inside client applications and deployed in production with the software development kits offered by NVIDIA. These SDKs leverage **TensorRT** and the **Triton Inference Server** as foundational building blocks.

NVIDIA provides AI toolkits and application frameworks targeted towards industries or specific use cases. These are listed in the table below.

<table>
<thead>
<tr>
<th>Industry or AI Application</th>
<th>AI Toolkits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthcare and Life Sciences</td>
<td>NVIDIA Clara</td>
</tr>
<tr>
<td>Streaming video analytics</td>
<td>NVIDIA Metropolis</td>
</tr>
<tr>
<td></td>
<td>NVIDIA DeepStream</td>
</tr>
<tr>
<td>Robotics</td>
<td>NVIDIA Isaac Sim</td>
</tr>
<tr>
<td>5G acceleration</td>
<td>NVIDIA Aerial</td>
</tr>
<tr>
<td>Recommender Systems</td>
<td>NVIDIA Merlin</td>
</tr>
<tr>
<td>Conversational AI</td>
<td>NVIDIA Jarvis</td>
</tr>
</tbody>
</table>
Chapter 19. Deep Learning Frameworks

Deep learning frameworks offer building blocks for designing, training and validating deep neural networks, through a high level programming interface. Widely used deep learning frameworks such as MXNet, PyTorch, TensorFlow and others rely on GPU-accelerated libraries such as cuDNN, NCCL and DALI to deliver high-performance multi-GPU accelerated training.

Developers, researchers and data scientists can get easy access to NVIDIA optimized deep learning framework containers with deep learning examples, that are performance tuned and tested for NVIDIA GPUs. This eliminates the need to manage packages and dependencies or build deep learning frameworks from source.

19.1. GPU Operator

The Operator Framework within Kubernetes takes operational business logic and allows the creation of an automated framework for the deployment of applications within Kubernetes using standard Kubernetes APIs and kubectl. The NVIDIA GPU Operator automates the management of all NVIDIA software components needed to provision GPUs within Kubernetes. NVIDIA, Red Hat, and others in the community have collaborated on creating the GPU Operator. The GPU Operator is an important component of the NVIDIA EGX software-defined platform that is designed to make large-scale hybrid-cloud and edge operations possible and efficient.