BEST PRACTICES FOR DOCKER

DG-08863-001 _v001 | September 2018

Best Practices

TABLE OF CONTENTS

Chapter 1.
NVIDIA CONTAINER BEST PRACTICES

A Docker container is composed of layers. The layers are combined to create the container. You can think of layers as intermediate images that add some capability to the overall container. If you make a change to a layer through a DockerFile (see Building Containers), than Docker rebuilds that layer and all subsequent layers but not the layers that are not affected by the build. This reduces the time to create containers and also allows you to keep them modular.

Docker is also very good about keeping one copy of the layers on a system. This saves space and also greatly reduces the possibility of "version skew" so that layers that should be the same are not duplicated.

1.1. Hello World For Containers

To make sure you have access to the NVIDIA containers, start with the proverbial “hello world” of Docker commands.

For DGX-2, DGX-1, and DGX Station, simply log into the system. For NGC consult the NGC documentation for details about your specific cloud provider. In general, you will start a cloud instance with your cloud provider using the NVIDIA Volta Deep Learning Image. After the instance has booted, log into the instance.

Next, you can issue the **`docker --version`** command to list the version of Docker for DGX-2, DGX-1, and DGX Station. The output of this command tells you the version of Docker on the system (**`17.05-ce, build 89658be`**).

At any time, if you are not sure about a Docker command, issue the **`$ docker --help`** command.

1.2. Logging Into Docker

If you have a DGX-2, DGX-1, or DGX Station, the first time you login, you are required to set-up access to the NVIDIA NGC Registry (https://ngc.nvidia.com). For more information, see the NGC Getting Started Guide.

1.3. Listing Docker Images

Typically, one of the first things you will want to do is get a list of all the Docker images that are currently available on the local computer. When the Docker containers are *stored* in a repository, they are said to be a *container*. When you *pull* the container from a repository to a system, such as the DGX-2 or DGX-1, it is then said to be a Docker *image*. This means the image is local.

Issue the **`$ docker images`** command to list the images on the server. Your screen will look similar to the following:

```
[jelayton@hsw218 ~]$ docker images
REPOSITORY                                TAG                                 IMAGE ID
mxnet-dec                                 latest                              65a48e11da96
<none>                                    <none>                              bfc4512ca5f2
nvcr.io/nvidian_general/adlr_pytorch      resumes                             a134a09668a8
<none>                                    <none>                              0f4ab6d62241
<none>                                    <none>                              97274da5c898
nvcr.io/nvidian_sas/gamess-libcchem       cuda8                               3dc13f8347f9
nvidia/cuda                               latest                              614dcdafa05c
ubuntu                                    latest                              d355ed3537e9
deeper_photo                              latest                              932634514d5a
```

Figure 1 Listing of Docker images

and

```
nvcr.io/nvidia/torch                      17.06                               b7b62dacdeb1
nvidia/cuda                               8.0-devel-centos7                   6e3e5b71176e
nvcr.io/nvidia/tensorflow                 17.06                               56f2980b1e37
nvidia/cuda                               8.0-cudnn5-devel-ubuntu14.04        22afb0578249
nvidia/cuda                               8.0-devel                           a760a0cfca82
mxnet/python                              gpu                                 7e7c9176319c
nvcr.io/nvidian_sas/chainer               latest                              2ea707c58bea
deep_photo                                latest                              ef4510510506
<none>                                    <none>                              9124236672fe
nvcr.io/nvidia/cuda                       8.0-cudnn6-devel-ubuntu16.04        02910409eb5d
nvcr.io/nvidia/tensorflow                 17.05                               9dda0d5c344f
bfolkens/docker-opencv                    2.4.12-cuda7.0-cudnn4               6f925a3e242b
[jelayton@hsw218 ~]$
```

Figure 2 Listing of Docker images

In this example, there are a few Docker containers that have been pulled down to this system. Each image is listed along with its *tag*, the corresponding *Image ID*, also known as *container version*. There are two other columns that list when the container was created (approximately), and the approximate size of the image in GB. These columns have been cropped to improve readability.

The output from the command will vary.

At any time, if you need help, issue the **`$ docker images --help`** command.

1.4. Pulling A Container

Before you can pull a container from the NGC container registry, you must have Docker and nvidia-docker installed. For DGX users, this is explained in Preparing to use NVIDIA Containers Getting Started Guide.

For users other than DGX, follow the NVIDIA® GPU Cloud™ (NGC) container registry nvidia-docker installation documentation based on your platform.

You must also have access and logged into the NGC container registry as explained in the NGC Getting Started Guide.

Pulling a container to the system makes the container an image. When the container is pulled to become an image, all of the *layers* are downloaded. Depending upon how many layers are in the container and how the system is connected to the Internet, it may take some time to download.

The **`$ docker pull nvcr.io/nvidia/tensorflow:17.06`** command *pulls* the container from the NVIDIA repository to the local system where the command is run. At that point, it is a *Docker image*. The structure of the *pull* command is:

```
$ docker pull <repository>/nvidia/<container>:<tag>
```

Where:

- **`<repository>`** is the path to where the container is stored (the Docker repo). In the following example, the repository is **`nvcr.io/nvidia`** (NVIDIA's private repository).
- **`<container>`** is the name of the container. In the following example we use **`tensorflow`**.
- **`<xx.xx>`** is the specific version of the container. In the following example we use **`17.06`**.

Below is a picture of a TensorFlow image that is pulled using the following command:

```
$ docker pull nvrc.io/nvidia/tensorflow:17.06
```

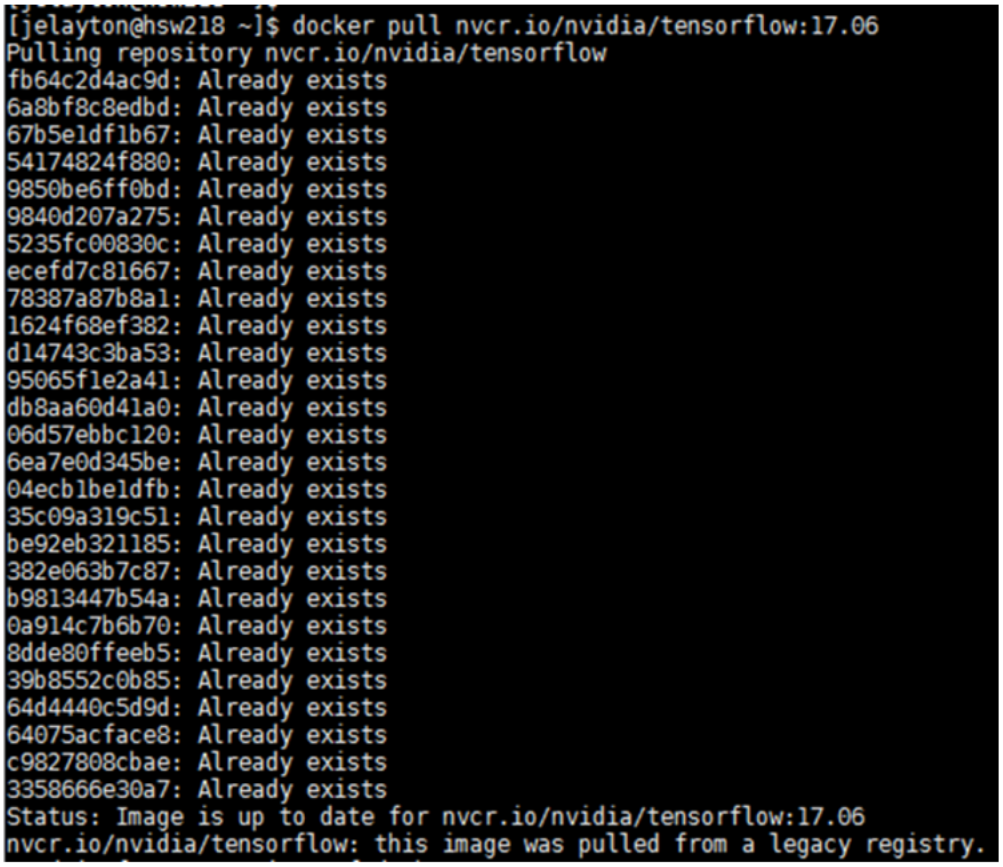

Figure 3 Example of pulling TensorFlow 17.06

As you can tell, the container had already been pulled down on this particular system (some of the output from the command has been cut off). At this point the image is ready to be run.

In most cases, you will not find a container already downloaded to the system. Below is some sample output for the case when the container has to be pulled down from the registry, using the command:

```
$ docker pull nvrc.io/nvidia/tensorflow:17.06
```

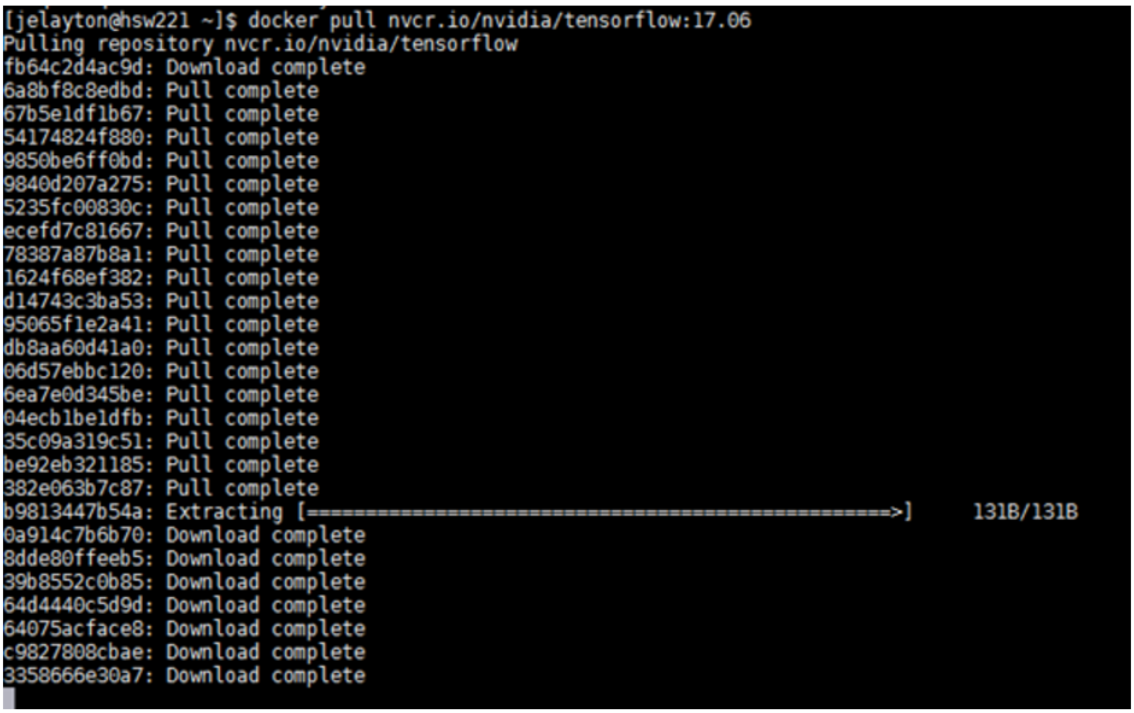

Figure 4 Example of pulling TensorFlow 17.06 that had not already been loaded onto the server

Below is the output after the *pull* is finished, using the command:

```
$ docker pull nvrc.io/nvidia/tensorflow:17.06
```

```
[jelayton@hsw221 ~]$ docker pull nvcr.io/nvidia/tensorflow:17.06
Pulling repository nvcr.io/nvidia/tensorflow
fb64c2d4ac9d: Pull complete
6a8bf8c8edbd: Pull complete
67b5e1df1b67: Pull complete
54174824f880: Pull complete
9850be6ff0bd: Pull complete
9840d207a275: Pull complete
5235fc00830c: Pull complete
ecefd7c81667: Pull complete
78387a87b8a1: Pull complete
1624f68ef382: Pull complete
d14743c3ba53: Pull complete
95065f1e2a41: Pull complete
db8aa60d41a0: Pull complete
06d57ebbc120: Pull complete
6ea7e0d345be: Pull complete
04ecb1be1dfb: Pull complete
35c09a319c51: Pull complete
be92eb321185: Pull complete
382e063b7c87: Pull complete
b9813447b54a: Pull complete
0a914c7b6b70: Pull complete
8dde80ffeeb5: Pull complete
39b8552c0b85: Pull complete
64d4440c5d9d: Pull complete
64075acface8: Pull complete
c9827808cbae: Pull complete
3358666e30a7: Pull complete
Status: Downloaded newer image for nvcr.io/nvidia/tensorflow:17.06
nvcr.io/nvidia/tensorflow: this image was pulled from a legacy registry.
```

Figure 5 Pulling of the container is complete

The screen capture has been cropped in the interest of readability.

1.5. Running A Container

After the nvidia-docker container is pulled down to the system, creating a Docker image, you can *run* or execute the image.

Important Use the **`nvidia-docker`** command to ensure that the correct NVIDIA drivers and libraries are used. The next section discusses nvidia-docker.

A typical command to run the container is:

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

Where:

- **`-it`** means interactive
- **`--rm`** means delete the image when finished
- **`-v`** means mount directory

- **local_dir** is 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, **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** is 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
```

- **<container>** is the name of the container.
- **<xx.xx>** is the container version, also known as tag. For example, **18.01**.

1.6. Verifying

After a Docker image is running, you can verify by using the classic ***nix** option **ps**. For example, issue the **$ docker ps -a** command.

```
[jelayton@hsw218 ~]$ docker ps -a
CONTAINER ID        IMAGE                             COMMAND                  CREATED
12a4854ba738        nvcr.io/nvidia/tensorflow:17.06   "/usr/local/bin/nv..."   51 seconds ago
[jelayton@hsw218 ~]$
```

Figure 6 Verifying a Docker image is running

Without the **-a** option, only running instances are listed.

Important It is best to include the **-a** option in case there are hung jobs running or other performance problems.

You can also stop a running container if you want. For example:

```
[jelayton@hsw218 ~]$ docker ps -a
CONTAINER ID        IMAGE                             COMMAND
 PORTS               NAMES
12a4854ba738        nvcr.io/nvidia/tensorflow:17.06   "/usr/local/bin/nv..."
 6006/tcp            brave_neumann
[jelayton@hsw218 ~]$
[jelayton@hsw218 ~]$ docker stop 12a4854ba738
12a4854ba738
[jelayton@hsw218 ~]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
  NAMES
[jelayton@hsw218 ~]$
```

Figure 7 Stopping a container from running

Notice that you need the *Container ID* of the image you want to stop. This can be found using the **$ docker ps -a** command.

Another useful command or Docker option is to remove the image from the server. Removing or deleting the image saves space on the server. For example, issue the following command:

```
$ docker rmi nvcr.io/nvidia.tensorflow:17.06
```

```
[jelayton@hsw218 ~]$ docker rmi nvcr.io/nvidia/tensorflow:17.06
Untagged: nvcr.io/nvidia/tensorflow:17.06
Deleted: sha256:56f2980b1e37e528edc3f883bc85ab2ae01e2333629663fafc9b52dbc7187f86
Deleted: sha256:cb25b07a3320dbda322e69a3c0188b3d8b328aa96bfef184a948d31a8a6afcea
Deleted: sha256:3358666e30a79f2bed5fbfa662e293b78aad0351ff294dbdcf3a7d5f4fffcaf0
Deleted: sha256:c9827808cbae4bfc50df35ed6b242c93d898ae0e55157a4d8cc720566ac4900d
Deleted: sha256:64075acface83b17a83e4398f3e0bf5aba24abc880995f6fc908a6aed0af9ac0
Deleted: sha256:64d4440c5d9d3e710895fd7336a9be8820a3bfda15e12cf44415592ee52d00e7
Deleted: sha256:39b8552c0b857313540a33e81e56b81274ce895913d08ab87365d69d115c3707
Deleted: sha256:8dde80ffeeb58e1d57e31779c4f59d91b817ea3a331c725be7ab885d40ebccfe
Deleted: sha256:0a914c7b6b701d36c09029d1b769f2ee7e2ba7c365259e1760356e16d302dffb
Deleted: sha256:b9813447b54a2c184089b3ffb75185549c4eba227d0ea5d93c147d32997fb8bc
Deleted: sha256:382e063b7c87976f7afe9917d69fed06bdf005498903ccd5aeba811e2784af32
Deleted: sha256:be92eb321185ffd6d30149bea7fb8ced8f1eb6662230fe33198ec183d0df49e6
Deleted: sha256:35c09a319c5181975304dba5da76cd40ea8bd565ddbb5fbf12a08f3568f85a1b
Deleted: sha256:04ecb1be1dfbdfcfc6127921daff9299227fb8d52d0695c1b2a235a13d4e5a3e
Deleted: sha256:6ea7e0d345be5f6c741fd101684c68c7f5bddcdf1cb60828779b84ec5a9fdc8b
Deleted: sha256:06d57ebbc1205c5e8d5a5f266358757c38058a44f2ba55e36b7c7f98f5343aa3
Deleted: sha256:db8aa60d41a007b1c318628315bb4174d79efd4b0cf65efc9b8a677275da5a54
```

Figure 8 Removing an image from the server

If you list the images, **`$ docker images`**, on the server, then you will see that the image is no longer there.

```
[jelayton@hsw218 ~]$ docker images
REPOSITORY                                TAG                 IMAGE ID
mxnet-dec                                 latest              65a48e11da96
<none>                                    <none>              bfc4512ca5f2
nvcr.io/nvidian_general/adlr_pytorch      resumes             a134a09668a8
<none>                                    <none>              0f4ab6d62241
<none>                                    <none>              97274da5c898
nvcr.io/nvidian_sas/gamess-libcchem       cuda8               3dc13f8347f9
nvidia/cuda                               latest              614dcdafa05c
ubuntu                                    latest              d355ed3537e9
deeper_photo                              latest              932634514d5a
```

Figure 9 Confirming the image is removed from the server

and

```
nvcr.io/nvidia/torch                      17.06                           b7b62dacdeb1
nvidia/cuda                               8.0-devel-centos7               6e3e5b71176e
nvidia/cuda                               8.0-cudnn5-devel-ubuntu14.04    22afb0578249
nvidia/cuda                               8.0-devel                       a760a0cfca82
mxnet/python                              gpu                             7e7c9176319c
nvcr.io/nvidian_sas/chainer               latest                          2ea707c58bea
deep_photo                                latest                          ef4510510506
<none>                                    <none>                          9124236672fe
nvcr.io/nvidia/cuda                       8.0-cudnn6-devel-ubuntu16.04    02910409eb5d
nvcr.io/nvidia/tensorflow                 17.05                           9dda0d5c344f
bfolkens/docker-opencv                    2.4.12-cuda7.0-cudnn4           6f925a3e242b
```

Figure 10 Confirming the image is removed from the server

Chapter 2.
DOCKER BEST PRACTICES

You can run an nvidia-docker container on any platform that is Docker compatible allowing you to move your application to wherever you need. The containers are platform-agnostic, and therefore, hardware agnostic as well. To get the best performance and to take full advantage of the tremendous performance of a NVIDIA GPU, specific kernel modules and user-level libraries are needed. NVIDIA GPUs introduce some complexity because they require kernel modules and user-level libraries to operate.

One approach to solving this complexity when using containers is to have the NVIDIA drivers installed in the container and have the character devices mapped corresponding to the NVIDIA GPUs such as **`/dev/nvidia0`**. For this to work, the drivers on the host (the system that is running the container), must match the version of the driver installed in the container. This approach drastically reduces the portability of the container.

2.1. nvidia-docker Containers Best Practices

To make things easier for Docker® containers that are built for GPUs, NVIDIA® has created nvidia-docker. It is and open-source project hosted on GitHub. It is basically a wrapper around the **`docker`** command that takes care of orchestrating the GPU containers that are needed for your container to run.

Important It is highly recommended you use **`nvidia-docker`** when running a Docker container that uses GPUs.

Specifically, it provides two components 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.

The nvidia-docker containers focus solely on helping you run images that contain GPU dependent applications. Otherwise, it passes the arguments to the regular Docker commands. A good introduction to nvidia-docker is here.

Important Some things to always remember:

- Use the **nvidia-docker** command when you are running and executing containers.
- When building containers for NVIDIA GPUs, use the base containers in the repository. This will ensure the containers are compatible with nvidia-docker.

Let's assume the TensorFlow 17.06 container has been pulled down to the system and is now an image that is ready to be run. The following command can be used to execute it.

```
$ nvidia-docker run --rm -ti nvcr.io/nvidia/tensorflow:17.06
```

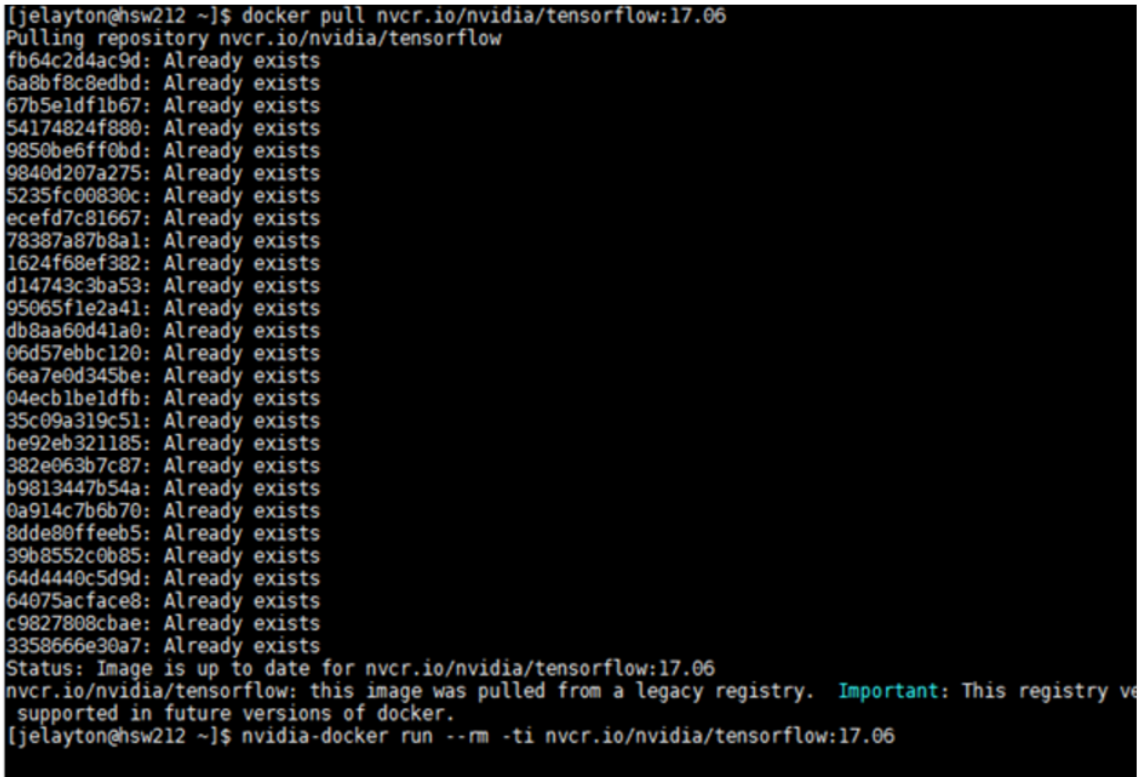

Figure 11 Executing the **run** command

This takes you to a command prompt inside the container.

Remember You are root inside the container.

Running the TensorFlow container didn't really do anything; it just brought up a command line inside the image where you are root. Below is a better example where the CUDA container is pulled down and the image is executed along with a simple command. This view at least gives you some feedback.

```
[jelayton@hsw212 ~]$ docker pull nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04
Pulling repository nvcr.io/nvidia/cuda
4a9bbba46489: Pull complete
cb11ba605400: Pull complete
1118f33a0ee7: Pull complete
11a2a269cf6e: Pull complete
80e85c818fa0: Pull complete
dd864b96a38e: Pull complete
1956df38caba: Pull complete
cf5bb1ce7253: Pull complete
4cc823847e48: Pull complete
0582f7cede97: Pull complete
7bfc48a9c5e5: Pull complete
Status: Downloaded newer image for nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04
nvcr.io/nvidia/cuda: this image was pulled from a legacy registry.  Important: This
rted in future versions of docker.
[jelayton@hsw212 ~]$ nvidia-docker run --rm -ti nvcr.io/nvidia/cuda:8.0-cudnn6-devel
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Tue_Jan_10_13:22:03_CST_2017
Cuda compilation tools, release 8.0, V8.0.61
```

Figure 12 Running an image to give you feedback

This docker image actually executed a command, **`nvcc --version`**, which provides some output, for example, the **`version`** of the **`nvcc`** compiler). If you want to get a bash shell in the image then you can run **`bash`** within the image.

```
[jelayton@hsw215 ~]$ nvidia-docker run --rm -ti nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04 bash
root@a09853992589:/# nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Tue_Jan_10_13:22:03_CST_2017
Cuda compilation tools, release 8.0, V8.0.61
root@a09853992589:/#
```

Figure 13 Getting a bash shell in the image

The frameworks that are part of the nvidia-docker repository, **`nvcr.io`**, have some specific options for achieving the best performance. This is true for DGX-2, DGX-1, and DGX Station. For more information, see Frameworks And Scripts Best Practices.

In the section Using And Mounting File Systems, some options for mounting external file systems in the running image are explained.

Important This allows you to keep data and code stored in one place on the system outside of the containers, while keeping the containers intact.

This also allows the containers to stay generic so they don't start proliferating when each user creates their own version of the container for their data and code.

2.2. `docker exec`

There are times when you will need to connect to a running container. You can use the **docker exec** command to connect to a running container to run commands. You can use the **bash** command to start an interactive command line terminal or bash shell.

```
$ docker exec -it <CONTAINER_ID_OR_NAME> bash
```

As an example, suppose one starts a Deep Learning GPU Training System™ (DIGITS) container with the following command:

```
$ nvidia-docker run -d --name test-digits \
  -u $(id -u):$(id -g) -e HOME=$HOME -e USER=$USER -v $HOME:$HOME \
  nvcr.io/nvidia/digits:17.05
```

After the container is running, you can now connect to the container instance.

```
$ docker exec -it test-digits bash
```

test-digits is the name of the container. If you don't specifically name the container, you will have to use the container ID.

Important By using **docker exec**, you can execute a snippet of code, a script, or attach interactively to the container making the **docker exec** command very useful.

For detailed usage of the **docker exec** command, see docker exec.

2.3. nvcr.io

Building deep learning frameworks can be quite a bit of work and can be very time consuming. Moreover, these frameworks are being updated weekly, if not daily. On top of this, is the need to optimize and tune the frameworks for GPUs. NVIDIA has created a Docker repository, named **nvcr.io**, where deep learning frameworks are tuned, optimized, tested, and containerized for your use.

NVIDIA creates an updated set of nvidia-docker containers for the frameworks monthly. Included in the container is source (these are open-source frameworks), scripts for building the frameworks, Dockerfiles for creating containers based on these containers, markdown files that contain text about the specific container, and tools and scripts for pulling down datasets that can be used for testing or learning. Customers who purchase a DGX-2, DGX-1 or DGX Station have access to this repository for pushing containers (storing containers).

To get started with DGX-2, DGX-1 or DGX Station, you need to create a system admin account for accessing **nvcr.io**. This account should be treated as an admin account so that users cannot access it. Once this account is created, the system admin can create accounts for projects that belong to the account. They can then give users access to these projects so that they can store or share any containers that they create.

2.4. Building Containers

You can build and store containers in the **nvcr.io** registry as a project within your account if you have a DGX-2, DGX-1, or DGX Station (for example, no one else can access the container unless you give them access).

This section of the document applies to Docker containers in general. You can use this general approach for your own Docker repository as well, but be cautious of the details.

Using a DGX-2, DGX-1, or DGX Station, you can either:

1. Create your container from scratch
2. Base your container on an existing Docker container
3. Base your container on containers in **nvcr.io**.

Any one of the three approaches are valid and will work, however, since the goal is to run the containers on a system which has GPUs, it's logical to assume that the applications will be using GPUs. Moreover, these containers are already tuned for GPUs. All of them also include the needed GPU libraries, configuration files, and tools to rebuild the container.

Important Based on these assumptions, it's recommended that you start with a container from **nvcr.io**.

An existing container in **nvcr.io** should be used as a starting point. As an example, the TensorFlow 17.06 container will be used and Octave will be added to the container so that some post-processing of the results can be accomplished.

1. Pull the container from the NGC container registry to the server. See Pulling A Container.
2. On the server, create a subdirectory called **mydocker**.

This is an arbitrary directory name.

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

```
[jelayton@hsw215 ~]$ mkdir mydocker
[jelayton@hsw215 ~]$ cd mydocker
[jelayton@hsw215 mydocker]$ vi Dockerfile
[jelayton@hsw215 mydocker]$ more Dockerfile
FROM nvcr.io/nvidia/tensorflow:17.06

RUN apt-get update

RUN apt-get install -y octave
[jelayton@hsw215 mydocker]$
```

Figure 14 Example of a Dockerfile

There are three lines in the **Dockerfile**.

- The first line in the **Dockerfile** tells Docker to start with the container **nvcr.io/nvidia/tensorflow:17.06**. This is the base container for the new container.
- The second line in the **Dockerfile** performs a package update for the container. It doesn't update any of the applications in the container, but updates the **apt-get** database. This is needed before we install new applications in the container.
- The third and last line in the **Dockerfile** tells Docker to install the package **octave** into the container using **apt-get**.

The Docker command to create the container is:

```
$ docker build -t nvcr.io/nvidian_sas/tensorflow_octave:17.06_with_octave
```

This command uses the default file **Dockerfile** for creating the container.

The command starts with **docker build**. The **-t** option creates a tag for this new container. Notice that the tag specifies the project in the **nvcr.io** repository where the container is to be stored. As an example, the project **nvidian_sas** was used along with the repository **nvcr.io**.

Projects can be created by your local administrator who controls access to **nvcr.io**, or they can give you permission to create them. This is where you can store your specific containers and even share them with your colleagues.

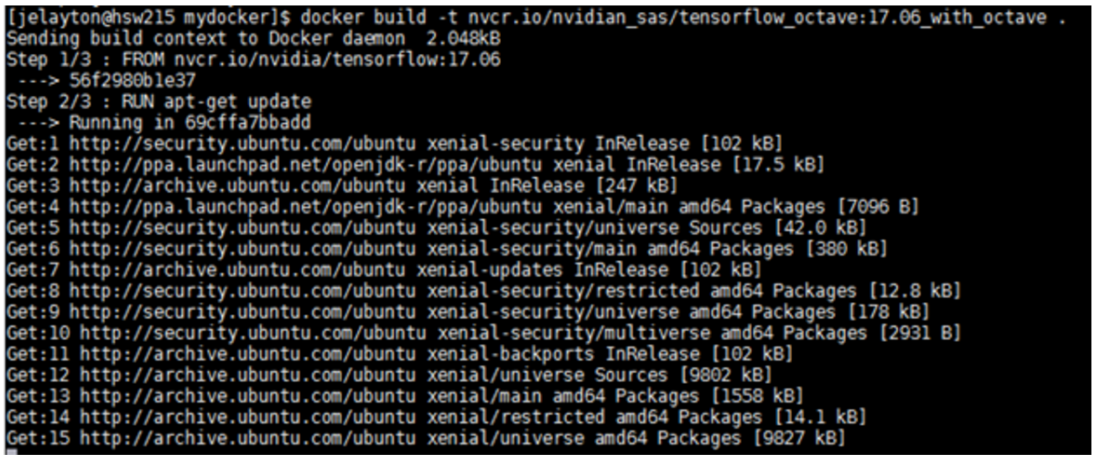

Figure 15 Creating a container using the Dockerfile

In the brief output from the **`docker build`** … command shown above, each line in the Dockerfile is a *Step*. In the screen capture, you can see the first and second steps (commands). Docker echos these commands to the standard out (**`stdout`**) so you can watch what it is doing or you can capture the output for documentation.

After the image is built, remember that we haven't stored the image in a repository yet, therefore, it's a **`docker image`**. Docker prints out the image id to **`stdout`** at the very end. It also tells you if you have successfully created and tagged the image.

If you don't see **`Successfully ...`** at the end of the output, examine your **`Dockerfile`** for errors (perhaps try to simplify it) or try a very simple **`Dockerfile`** to ensure that Docker is working properly.

4. Verify that Docker successfully created the image.

```
$ docker images
```

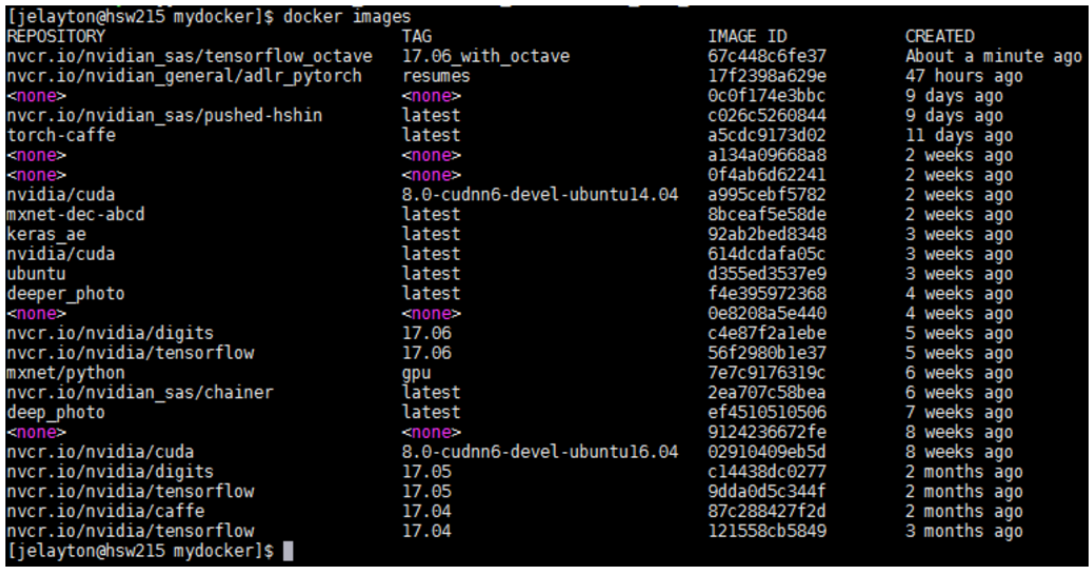

Figure 16 Verifying Docker created the image

The very first entry is the new image (about 1 minute old).

5. Push the image into the repository, creating a container.

```
docker push <name of image>
```

```
[jelayton@hsw215 mydocker]$ docker push nvcr.io/nvidian_sas/tensorflow_octave:17.06_with_octave
The push refers to a repository [nvcr.io/nvidian_sas/tensorflow_octave]
1b81f494d27d: Image successfully pushed
023cdba2c5b6: Image successfully pushed
8dd41145979c: Image successfully pushed
7cb16b9b8d56: Image already pushed, skipping
bd5775db0720: Image already pushed, skipping
bc0c86a33aa4: Image already pushed, skipping
cc73913099f7: Image already pushed, skipping
d49f214775fb: Image already pushed, skipping
5d6703088aa0: Image already pushed, skipping
7822424b3bee: Image already pushed, skipping
e999e9a30273: Image already pushed, skipping
e33eae9b4a84: Image already pushed, skipping
4a2ad165539f: Image already pushed, skipping
7efc092a9b04: Image already pushed, skipping
914009c26729: Image already pushed, skipping
4a7ea614f0c0: Image already pushed, skipping
550043e76f4a: Image already pushed, skipping
9327bc01581d: Image already pushed, skipping
6ceab726bc9c: Image already pushed, skipping
362a53cd605a: Image already pushed, skipping
4b74ed8a0e09: Image already pushed, skipping
1f926986fb96: Image already pushed, skipping
832ac06c43e0: Image already pushed, skipping
4c3abd56389f: Image already pushed, skipping
d8b353eb3025: Image already pushed, skipping
f2e85bc0b7b1: Image already pushed, skipping
fc9e1e5e38f7: Image already pushed, skipping
fe9a3f9c4559: Image already pushed, skipping
6a8bf8c8edbd: Image already pushed, skipping
Pushing tag for rev [67c448c6fe37] on {https://nvcr.io/v1/repositories/nvidian_sas/tensorflow_oct
[jelayton@hsw215 mydocker]$
```

Figure 17 Example of the **`docker push`** command

The above screen capture is after the **`docker push`** ... command pushes the image to the repository creating a container. At this point, you should log into the NGC container registry at https://ngc.nvidia.com and look under your project to see if the container is there.

If you don't see the container in your project, make sure that the tag on the image matches the location in the repository. If, for some reason, the push fails, try it again in case there was a communication issue between your system and the container registry (**`nvcr.io`**).

To make sure that the container is in the repository, we can pull it to the server and run it. As a test, first remove the image from your DGX system using the command **`docker rmi`** .... Then, pull the container down to the server using **`docker pull`** .... The image can be run using **`nvidia-docker`** as shown below.

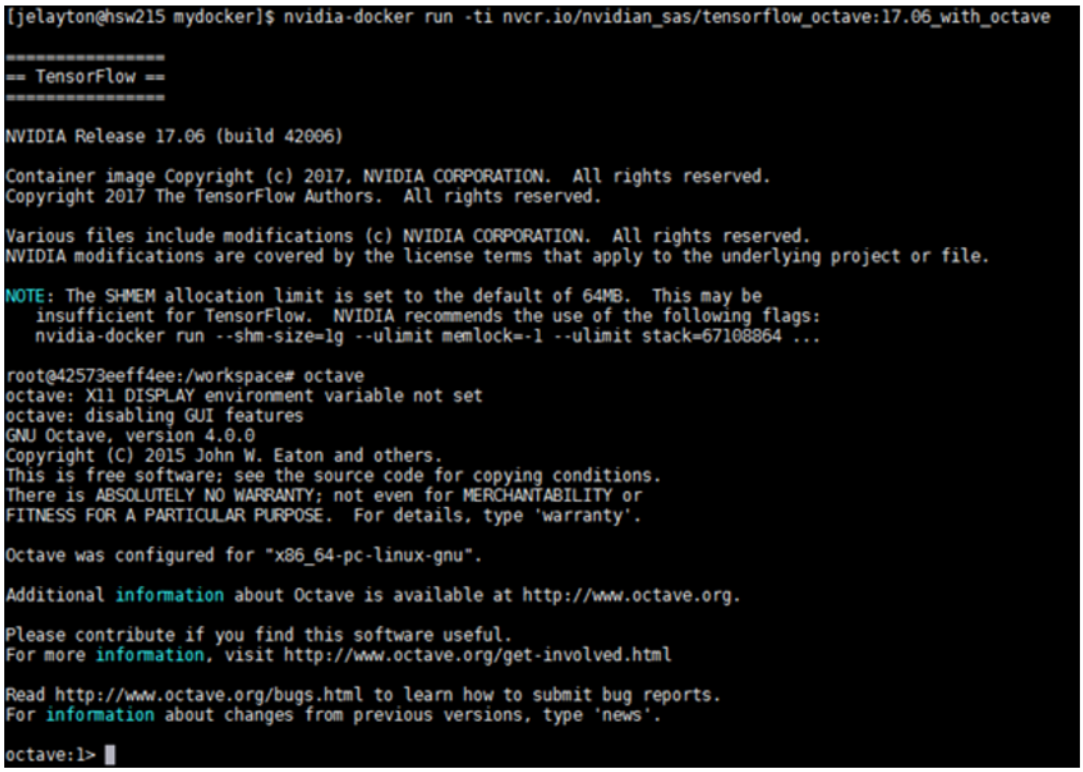

Figure 18 Example of using **nvidia-docker** to pull container

Notice that the **octave** prompt came up so it is installed and functioning correctly within the limits of this testing.

2.5. Using And Mounting File Systems

One of the fundamental aspects of using Docker is mounting file systems inside the Docker container. These file systems can contain input data for the frameworks or even code to run in the container.

Docker containers have their own internal file system that is separate from file systems on the rest of the host.

Important You can copy data into the container file system from outside if you want. However, it's far easier to mount an outside file system into the container.

Mounting outside file systems is done using the **nvidia-docker** command and the **-v** option. For example, the following command mounts two file systems:

```
$ nvidia-docker run --rm -ti ... -v $HOME:$HOME \
  -v /datasets:/digits_data:ro \
  ...
```

Most of the command has been erased except for the volumes. This command mounts the user's home directory from the external file system to the home directory in the

container (**-v $HOME:$HOME**). It also takes the **/datasets** directory from the host and mounts it on **/digits_data** inside the container (**-v /datasets:/digits_data:ro**).

Remember The user has root privileges with Docker, therefore you can mount almost anything from the host system to anywhere in the container.

For this particular command, the volume command takes the form of:

```
-v <External FS Path>:<Container FS Path>(options) \
```

The first part of the option is the path for the external file system. To be sure this works correctly, it's best to use the fully qualified path (FQP). This is also true for the mount point inside the container **<Container FS Path>**.

After the last path, various options can be used in the parenthesis **()**. In the above example, the second file system is mounted read-only (**ro**) inside the container. The various options for the **-v** option are discussed here.

The DGX™ systems (DGX-2, DGX-1, and DGX Station), and the nvidia-docker containers use the Overlay2 storage driver to mount external file systems onto the container file system. Overlay2 is a union-mount file system driver that allows you to combine multiple file systems so that all the content appears to be combined into a single file system. It creates a *union* of the file systems rather than an intersection.

Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, cuFFT, cuSPARSE, DIGITS, DGX, DGX-1, DGX Station, GRID, Jetson, Kepler, NVIDIA GPU Cloud, Maxwell, NCCL, NVLink, Pascal, Tegra, TensorRT, Tesla and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

www.nvidia.com