CONTAINERS

João Paulo Navarro (jpnavarro@nvidia.com)
Solutions Architect

Agenda

- Intro
- Containers Overview
- Managing Images, Containers and Running Containers
- Installing Docker and NVIDIA Docker

Introduction

DIFFERENT ROLES. SAME GOALS.

Driving Productivity and Faster Time-to-Solutions

Data Scientists and Researchers

Eliminate mundane tasks, focus on science and research

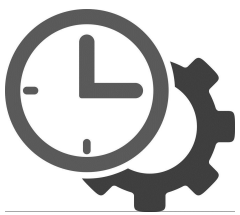
Developers

Speed up development with existing building blocks

Sysadmins

Deploy to production immediately

CHALLENGES UTILIZING AI & HPC SOFTWARE

EXPERTISE	INSTALLATION	OPTIMIZATION	PRODUCTIVITY	MAINTAINENCE
				
Building AI-centric solutions requires expertise	Complex, time consuming, and error-prone	Requires expertise to optimize framework performance	Users limited to older features and lower performance	IT can’t keep up with frequent software upgrades

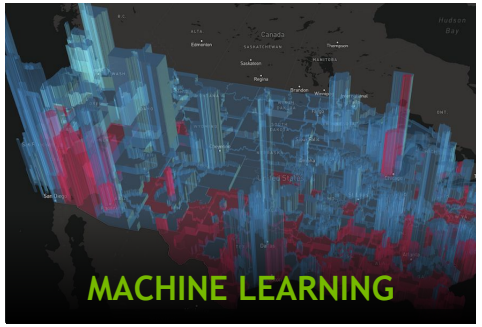
CONTAINERS – SIMPLIFYING AI & HPC WORKFLOWS

EMBEDDING EXPERTISE	FASTER DEPLOYMENTS	OPTIMIZED SOFTWARE	HIGHER PRODUCTIVITY	ZERO MAINTENANCE
				
Deliver greater value, faster	Eliminates installations. Simply Pull & Run the app	Key DL frameworks updated monthly for perf optimization	Better Insights and faster time-to-solution	Empowers users to deploy the latest versions with IT support

GPU-OPTIMIZED SOFTWARE CONTAINERS

Over 50 Containers on NGC – ngc.nvidia.com

TensorFlow | PyTorch | more

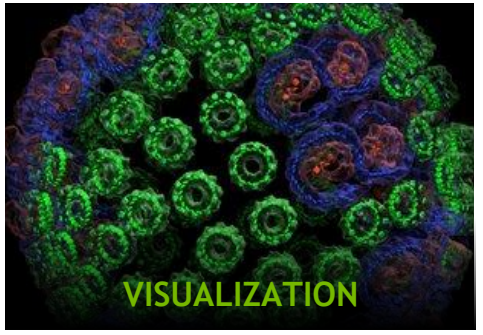


RAPIDS | H2O | more

TensorRT | DeepStream | more

NAMD | GROMACS | more

Parabricks



ParaView | IndeX | more

NVIDIA

NGC CONTAINERS: ACCELERATING WORKFLOWS

WHY CONTAINERS

Simplifies Deployments

- Eliminates complex, time-consuming builds and installs

Get started in minutes

- Simply Pull & Run the app

Portable

- Deploy across various environments, from test to production with minimal changes

WHY NGC CONTAINERS

Optimized for Performance

- Monthly DL container releases offer latest features and superior performance on NVIDIA GPUs

Scalable Performance

- Supports multi-GPU & multi-node systems for scale-up & scale-out environments

Designed for Enterprise & HPC environments

- Supports Docker & Singularity runtimes

Run Anywhere

- Pascal/Volta/Turing-powered NVIDIA DGX, PCs, workstations, and servers
- From Core to the Edge
- On-Prem to Hybrid to Cloud

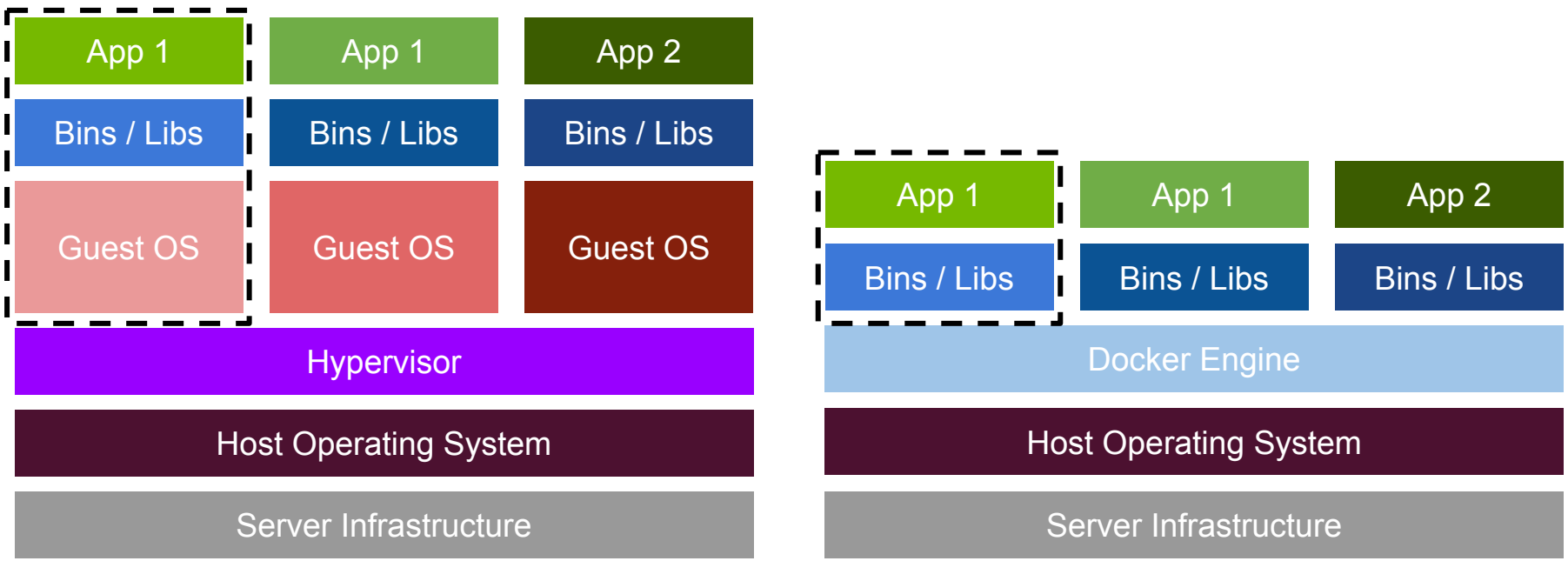
Container Overview

Containers

- Portable and reproducible builds
- Ease of deployment
- Run across heterogeneous CUDA toolkit environments (sharing the host driver)
- Bare Metal Performance
- Facilitate collaboration

Virtual Machine vs. Container

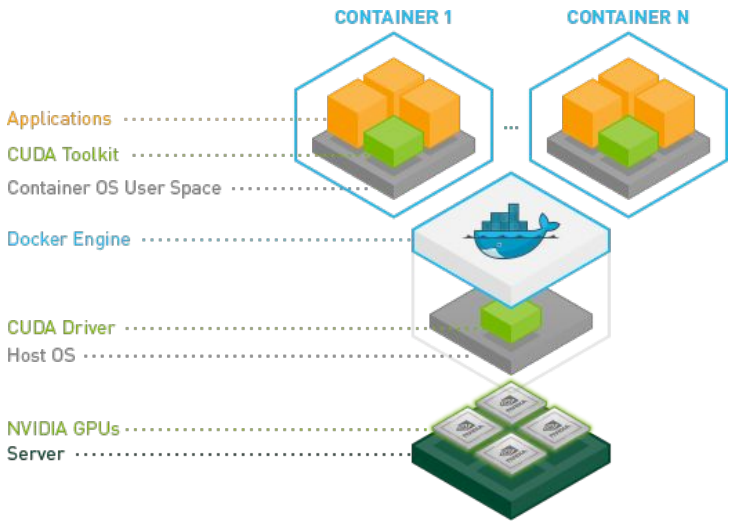
Not so similar



Virtual Machines

Containers

NVIDIA Container Runtime



https://github.com/NVIDIA/nvidia-docker

- ➔ Colloquially called “nvidia-docker”
- ➔ Docker containers are *hardware-agnostic* and *platform-agnostic*
- ➔ NVIDIA GPUs are specialized hardware that require the NVIDIA driver
- ➔ Docker does not natively support NVIDIA GPUs with containers
- ➔ NVIDIA Container Runtime makes the images agnostic of the NVIDIA driver
  - ◆ Required character devices and driver files are mounted when starting the container on the target machine
  - ◆ This makes Docker images portable while still leveraging NVIDIA GPUs

Docker

Definitions

Image
Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.

Container
A container is a runtime instance of a docker image.
A Docker container consists of

- A Docker image
- Execution environment
- A standard set of instructions

https://docs.docker.com/engine/reference/glossary/

Managing Images, Containers and Running Containers

Managing Images and Containers

Common Commands

List Images:

```
docker images
```

Remove an Image:

```
docker rmi imageID
```

```
docker rmi tensorRT
```

Remove all of your images:

The `-a` flag means "all" and the `-q` flag makes the output a list of imageID's.

```
docker rmi $(docker images -a -q)
```

List Containers:

```
docker ps -a
```

Stop a running Container:

```
docker stop containerID
```

Remove a Container:

```
docker rm containerID
```

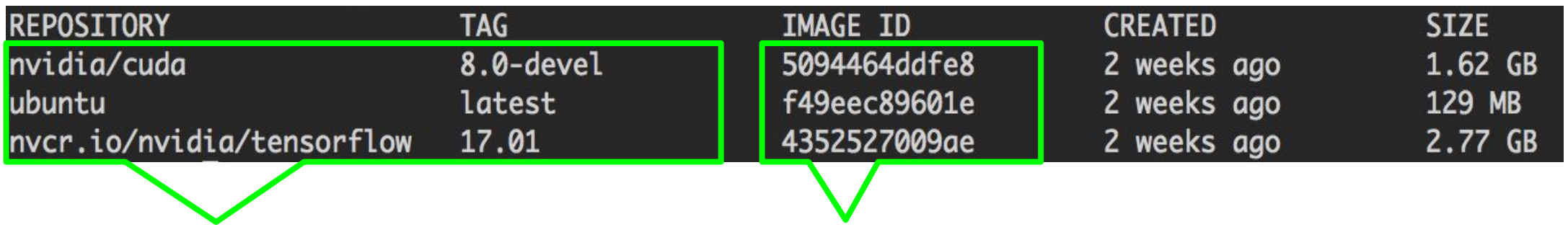
Remove all containers:

Remove all running containers (`-f` will try to force a shutdown of the container if it is running.)

```
docker rm -f $(docker ps -a -q)
```

- Refer to images and containers by their ID hash
- The first few characters of the image/container hash will do

Docker images detail



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nvidia/cuda	8.0-devel	5094464ddfe8	2 weeks ago	1.62 GB
ubuntu	latest	f49eec89601e	2 weeks ago	129 MB
nvcr.io/nvidia/tensorflow	17.01	4352527009ae	2 weeks ago	2.77 GB

Image Name = Repository:Tag

ImageID = Unique Hash

Running Containers

docker run and option

`docker run` Options

- ➔ `--runtime=nvidia` enable GPU capabilities
- ➔ `--rm` remove the container after it exits
- ➔ `-i -t` or `-it` interactive, and connect a "tty"
- ➔ `-d --detach` run in the background
- ➔ `--name` give the container a name
- ➔ `-p 8080:80` port map from host to container
- ➔ `-v ~/data:/data` map storage volume from host to container (bind mount) i.e. bind the `~data` directory in your home directory to `/data` in the container

Starts Tensorflow with ports, volumes, and console (All 1 line):

```
docker run
    --runtime=nvidia
    --rm -it
    --name MyCoolContainer
    -p 8888:80
    -v ~/data:/data
    nvcr.io/nvidia/tensorflow:18.01-py2
    examples/nvcnn.py
```

Navigating the NGC WebUI

NVIDIA GPU Cloud

How do we actually use it?

Our challenge:

- Sign up for a *free* NGC account at www.nvidia.com/ngcsignup
- Login to the WebUI
- Generate an API key for Docker to use

What is an API Key?

(And why do you need one?)

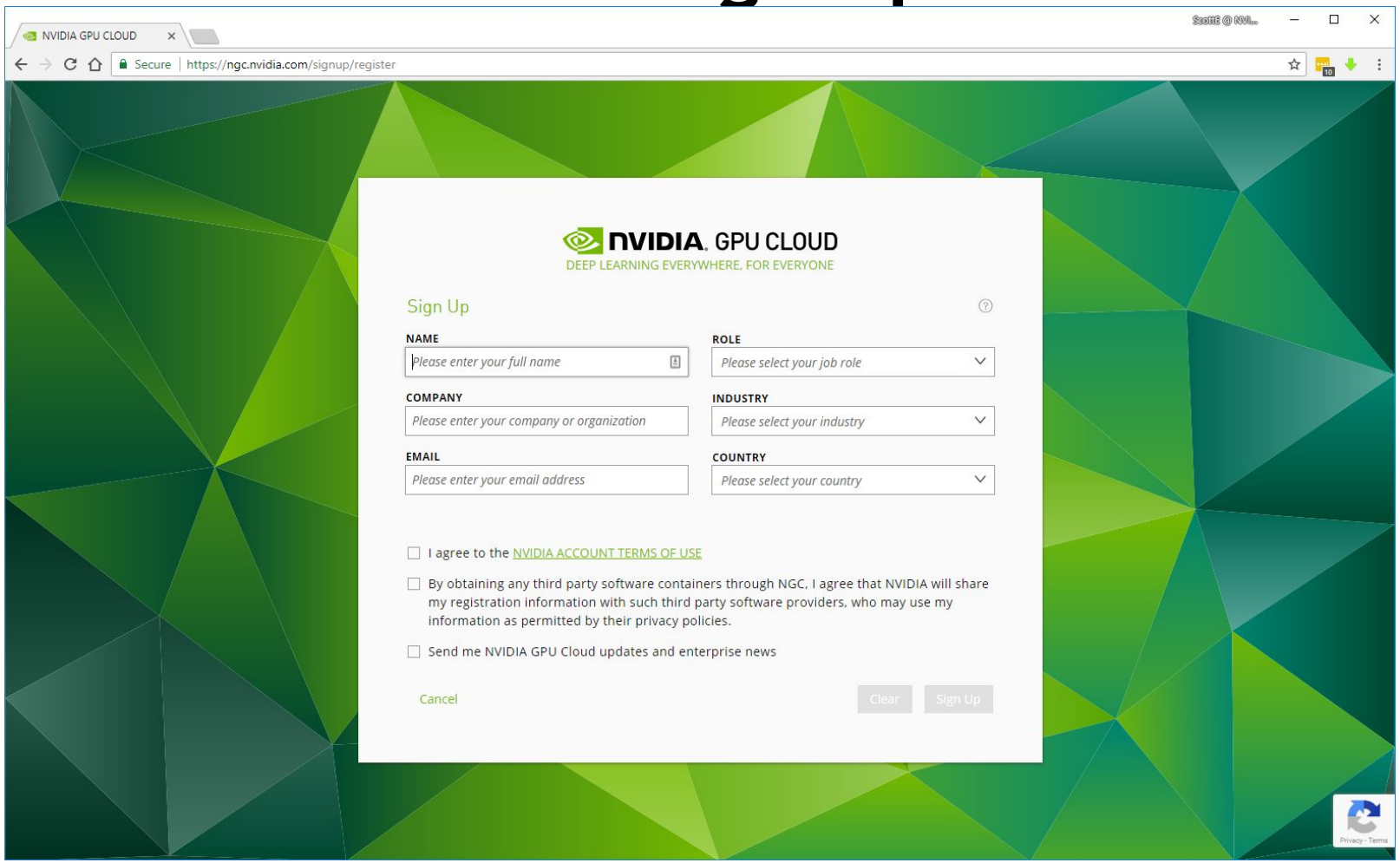
Your API key represents your credentials

- Used for programmatic interaction (e.g., docker, REST API, etc.)
- Uniquely identifies you (think “Username & Password”)
- There can be only one (regenerating your API key invalidates the old one)

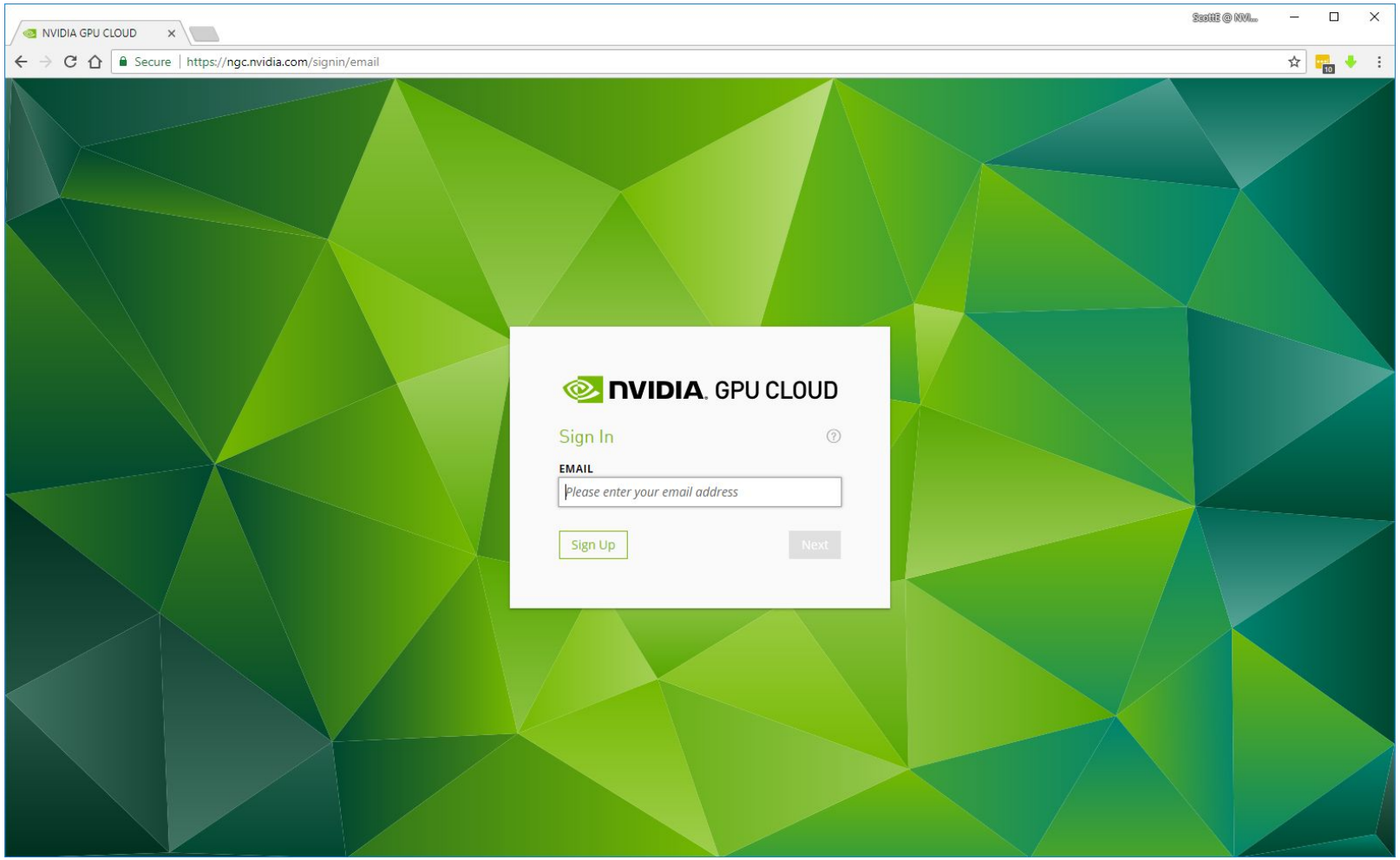
WebUI at ngc.nvidia.com: Use Username & Password

Programmatic interface at nvcr.io: Use API Key

NGC Sign-up


NVIDIA GPU CLOUD
Secure | https://ngc.nvidia.com/signup/register
NVIDIA. GPU CLOUD
DEEP LEARNING EVERYWHERE, FOR EVERYONE
Sign Up
NAME
Please enter your full name
ROLE
Please select your job role
COMPANY
Please enter your company or organization
INDUSTRY
Please select your industry
EMAIL
Please enter your email address
COUNTRY
Please select your country
I agree to the NVIDIA ACCOUNT TERMS OF USE
By obtaining any third party software containers through NGC, I agree that NVIDIA will share my registration information with such third party software providers, who may use my information as permitted by their privacy policies.
Send me NVIDIA GPU Cloud updates and enterprise news
Cancel
Clear
Sign Up

NGC Access

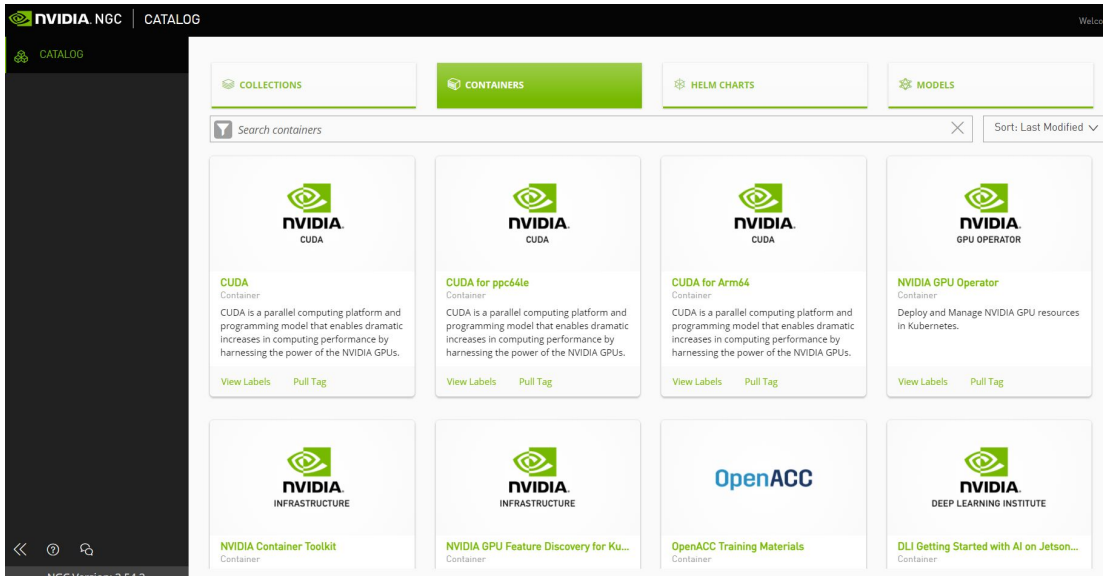

NVIDIA GPU CLOUD
Secure | https://ngc.nvidia.com/signin/email
NVIDIA GPU CLOUD
Sign In
EMAIL
Please enter your email address
Sign Up
Next

NGC WebUI

Where it all begins

When you login…

- Collections
- Containers List
- Instructions and Info
- Image specifics
- Docker pull shortcut

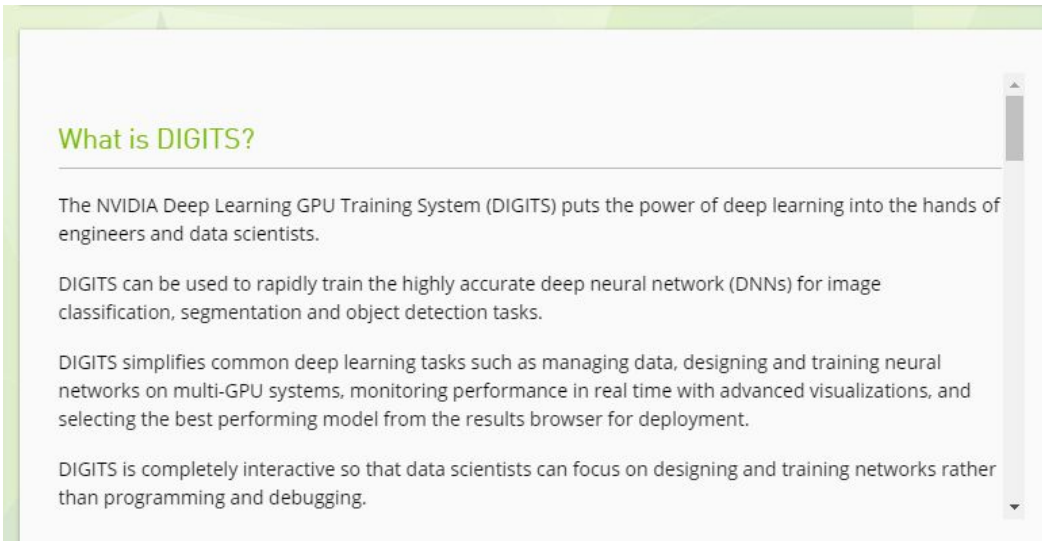


NGC WebUI

Instructions and Information

When you select a container

- Description on what the image contains
- Usually examples on running it
- Often has links for more information and tutorials

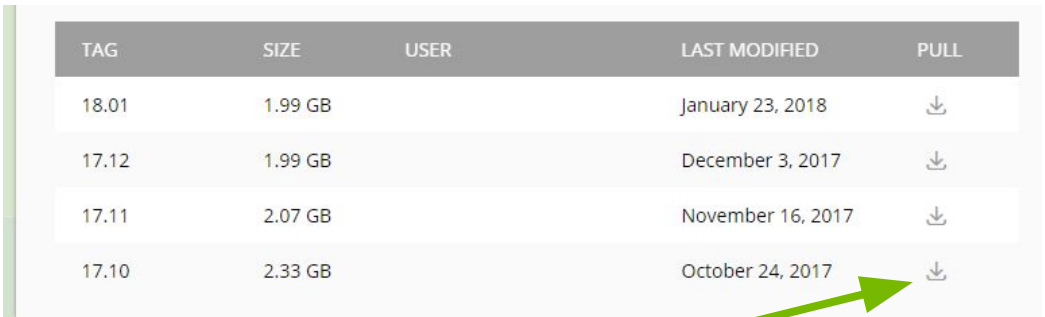


NGC WebUI

Image Specifics

List of images for that container

- Tag (`nvidia/caffe:18.01`)
  - Follows YY.MM format
- Creation date
  - Updated monthly
- Shortcut to copy `docker pull` command to clipboard



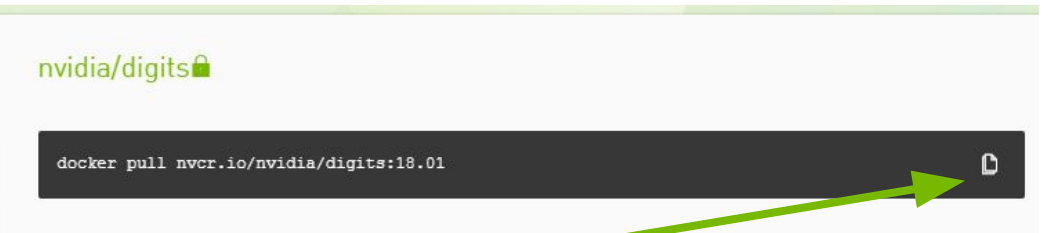
TAG	SIZE	USER	LAST MODIFIED	PULL
18.01	1.99 GB		January 23, 2018	
17.12	1.99 GB		December 3, 2017	
17.11	2.07 GB		November 16, 2017	
17.10	2.33 GB		October 24, 2017	

NGC WebUI

Docker pull shortcut

Shortcut to the latest at the top

- Shows full image name (`nvcr.io/nvidia/digits:18.01`)
- Icon to copy to clipboard
  - Same as in image details

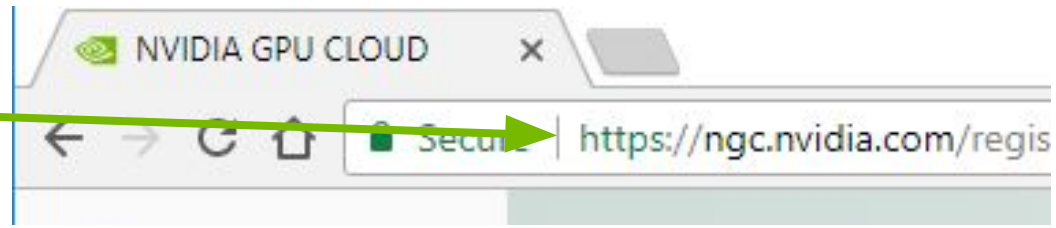


NGC vs. NVCR

Why are there two FQDNs?

ngc.nvidia.com

- NGC = NVIDIA GPU Cloud
- Used for administrative tasks
  - (a.k.a. The WebUI)

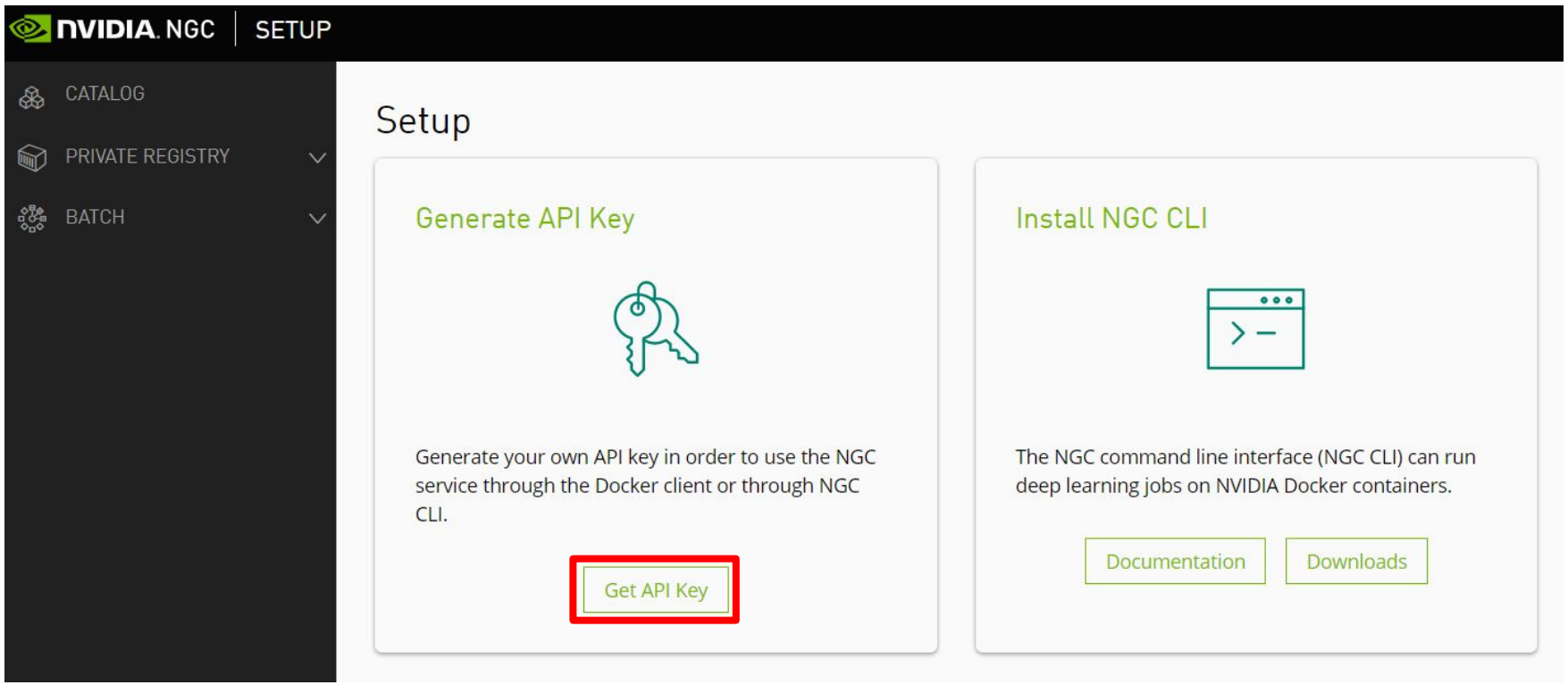



```
docker pull nvcr.io/nvidia/digits:18.01
```

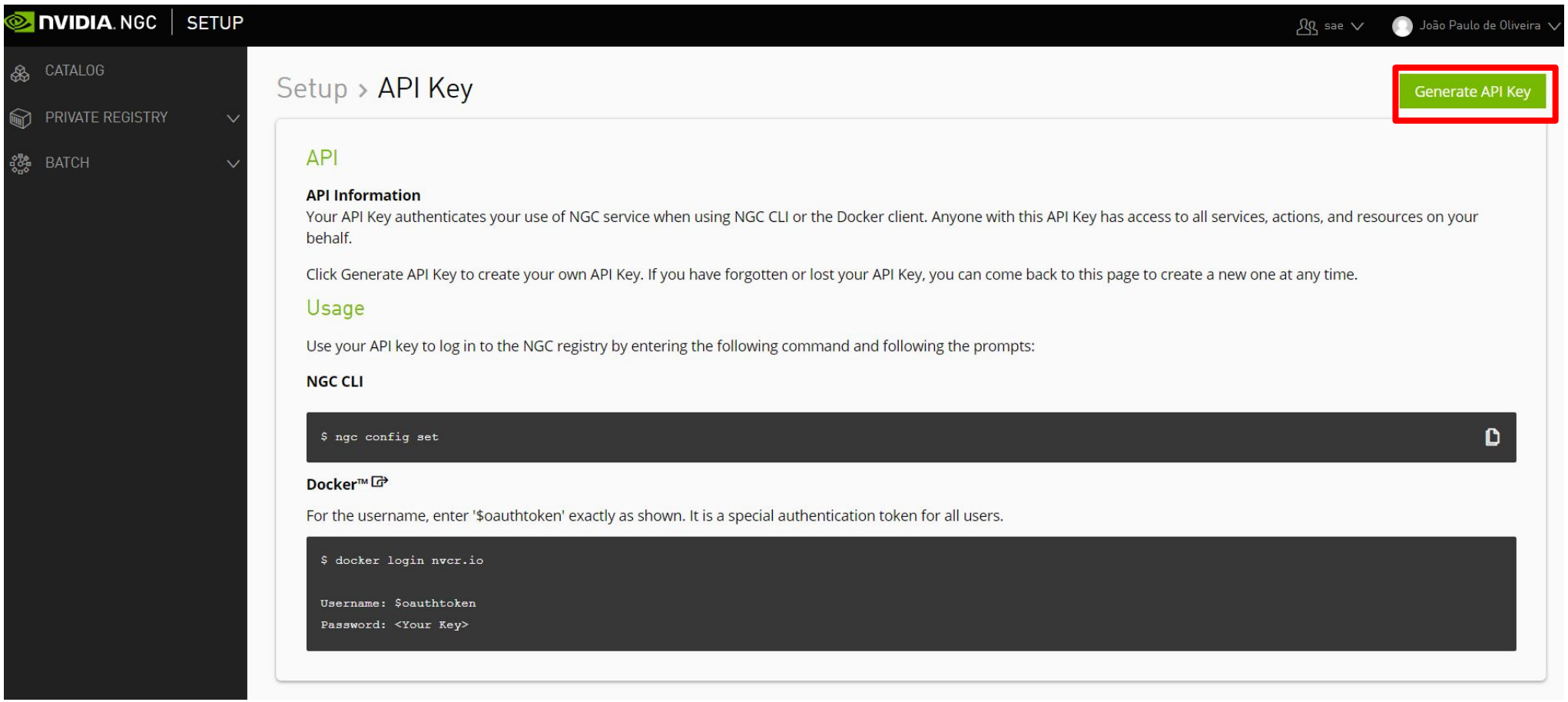
nvcr.io

- NVIDIA Container Repository
- Used for Docker tasks

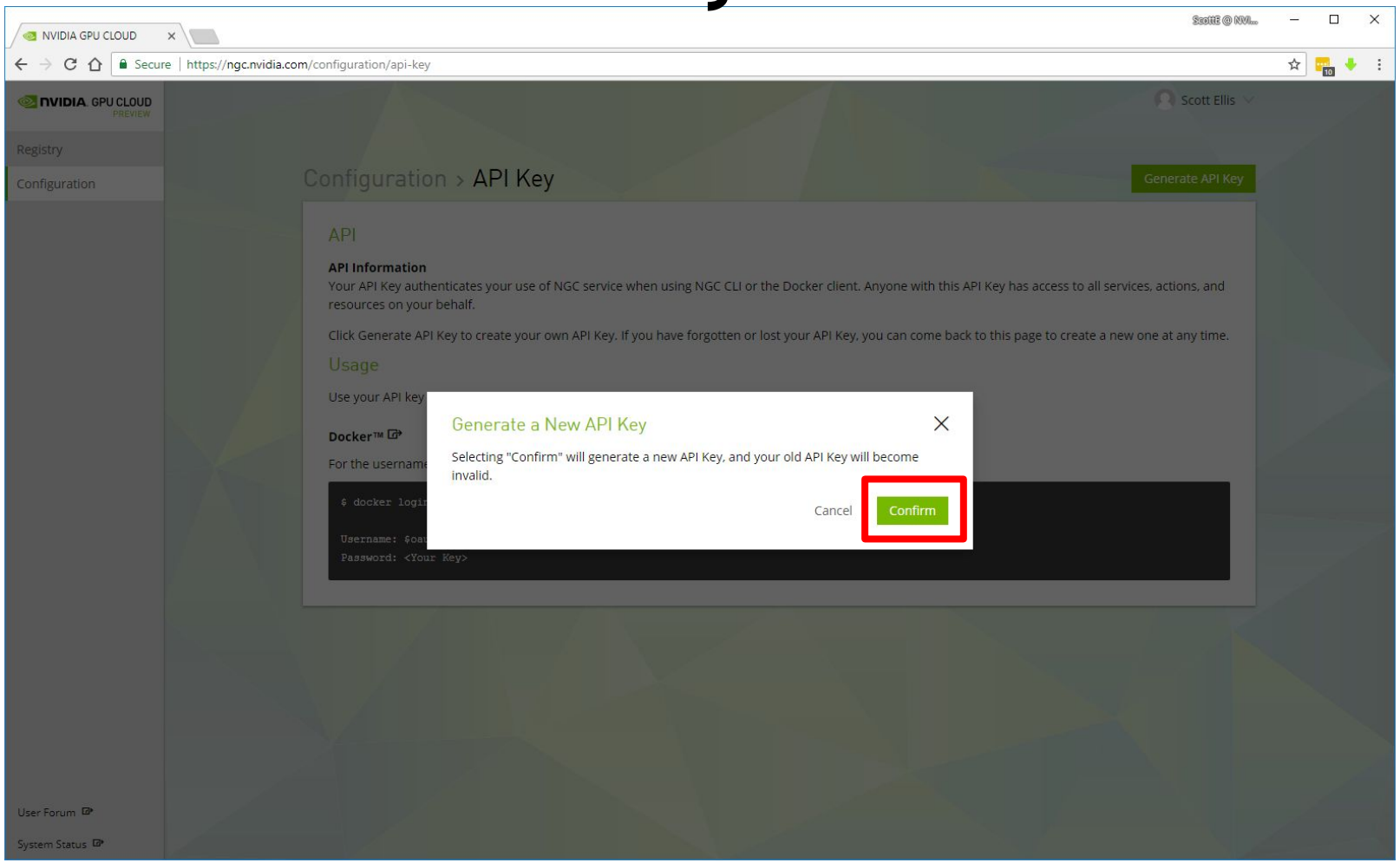
NGC API Key


NVIDIA. NGC
SETUP
CATALOG
PRIVATE REGISTRY
BATCH
Setup
Generate API Key
Generate your own API key in order to use the NGC service through the Docker client or through NGC CLI.
Get API Key
Install NGC CLI
The NGC command line interface (NGC CLI) can run deep learning jobs on NVIDIA Docker containers.
Documentation
Downloads

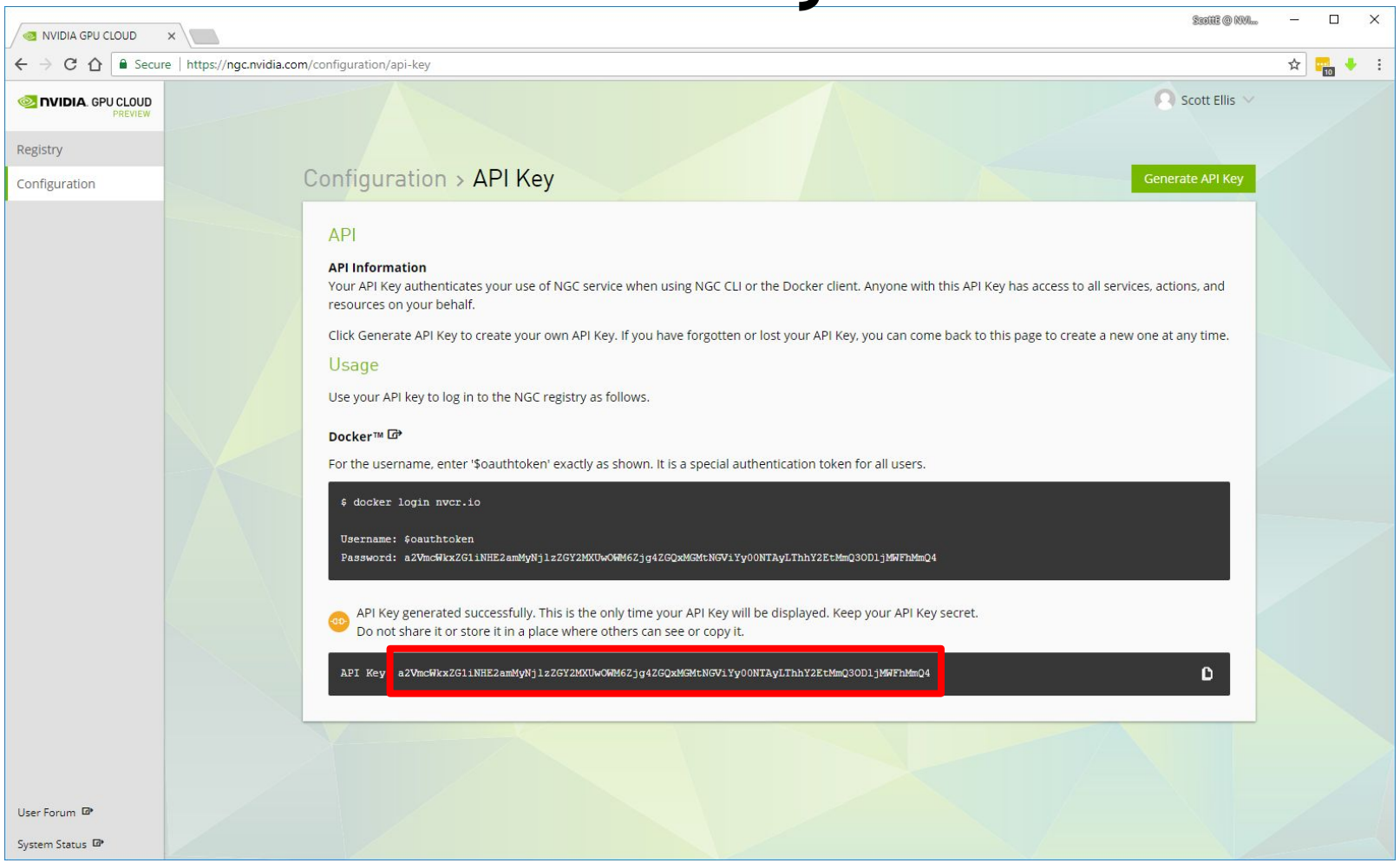
NGC API Key


NVIDIA NGC | SETUP
sae
João Paulo de Oliveira
CATALOG
PRIVATE REGISTRY
BATCH
Setup › API Key
Generate API Key
API
API Information
Your API Key authenticates your use of NGC service when using NGC CLI or the Docker client. Anyone with this API Key has access to all services, actions, and resources on your behalf.
Click Generate API Key to create your own API Key. If you have forgotten or lost your API Key, you can come back to this page to create a new one at any time.
Usage
Use your API key to log in to the NGC registry by entering the following command and following the prompts:
NGC CLI
$ ngc config set
Docker™
For the username, enter '$oauthtoken' exactly as shown. It is a special authentication token for all users.
$ docker login nvcr.io
Username: $oauthtoken
Password: <Your Key>

NGC API Key Generate



NGC API Key Save



EXERCISE

- Use the API key to login into nvcr.io
- Pull the TensorFlow container `nvcr.io/nvidia/tensorflow:18.09-py3`

Run a Container

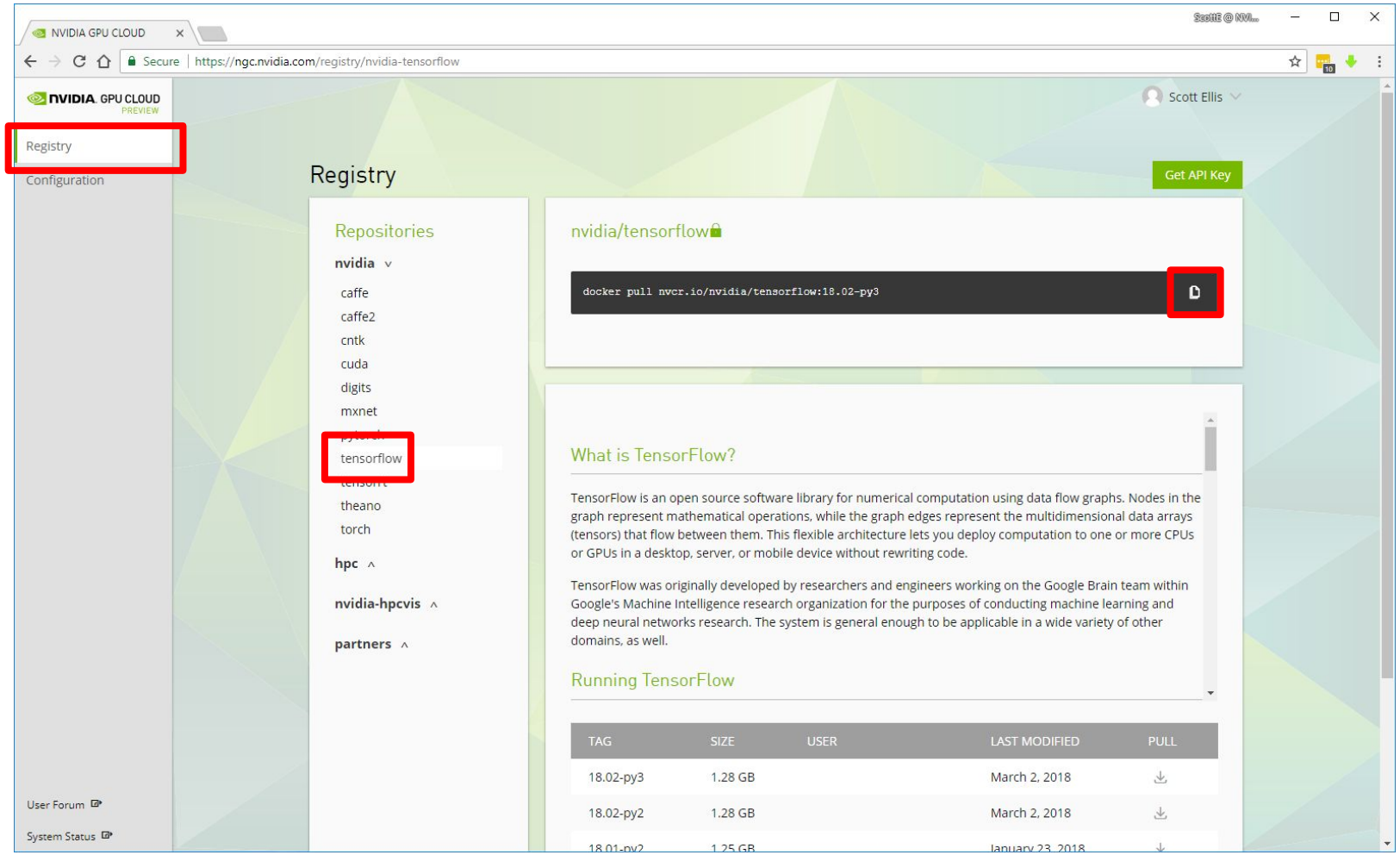
Container Execution

Quick TensorFlow Run

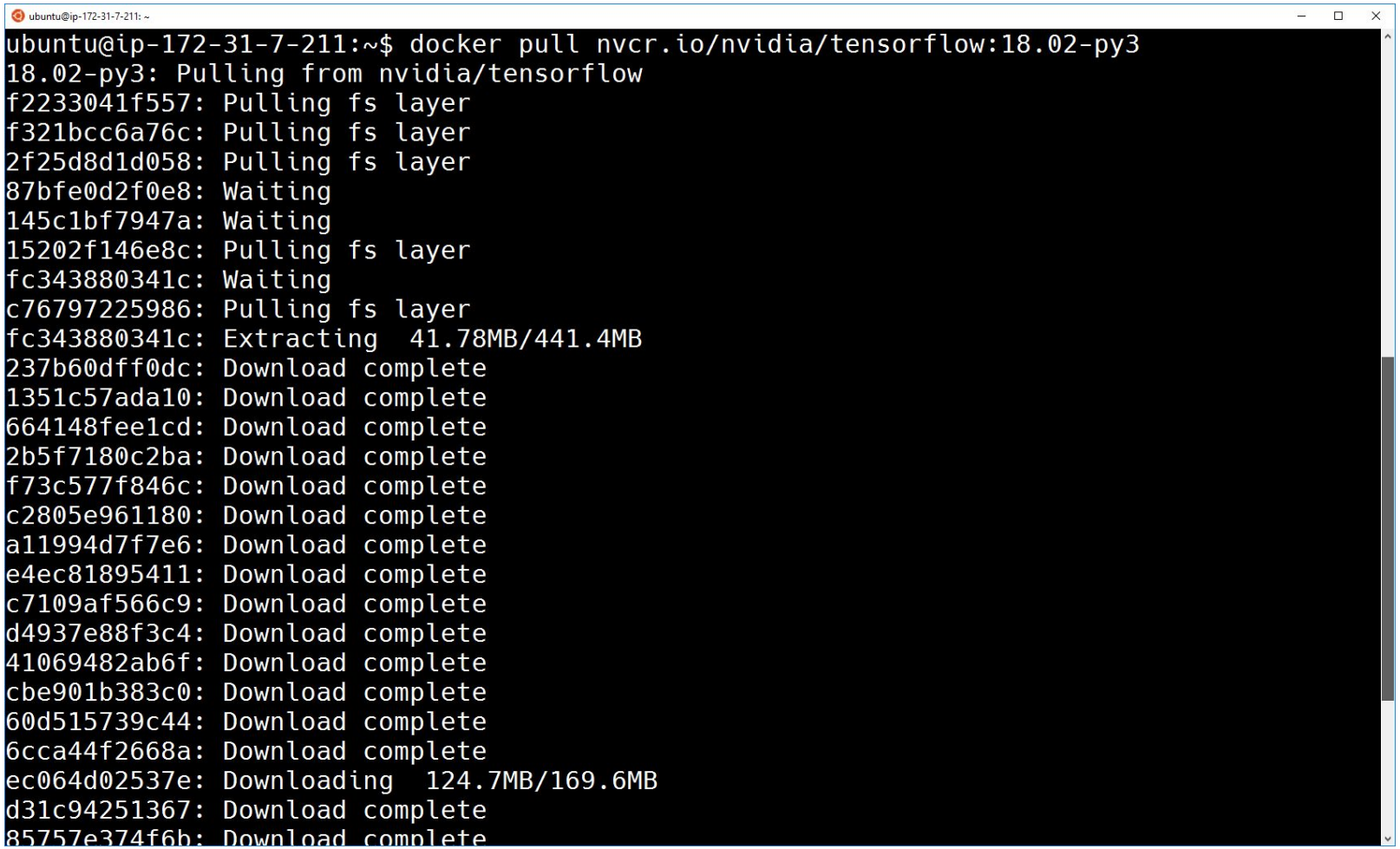
Our challenge:

- Download a TensorFlow container from NGC to local machine
- Run nvcnn.py sample with Resnet-50 with synthetic ImageNet data for 200 epochs

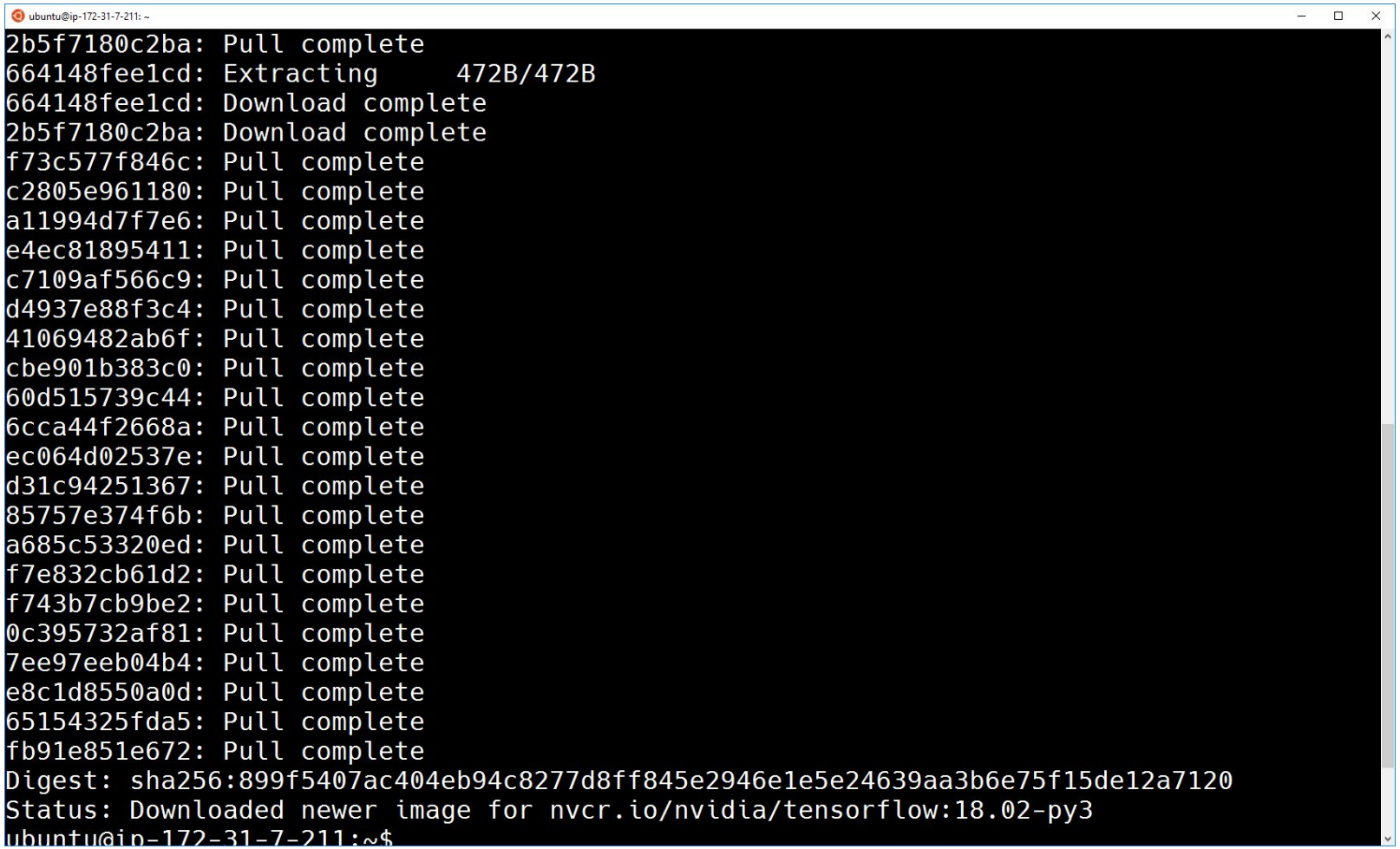
Container Execution



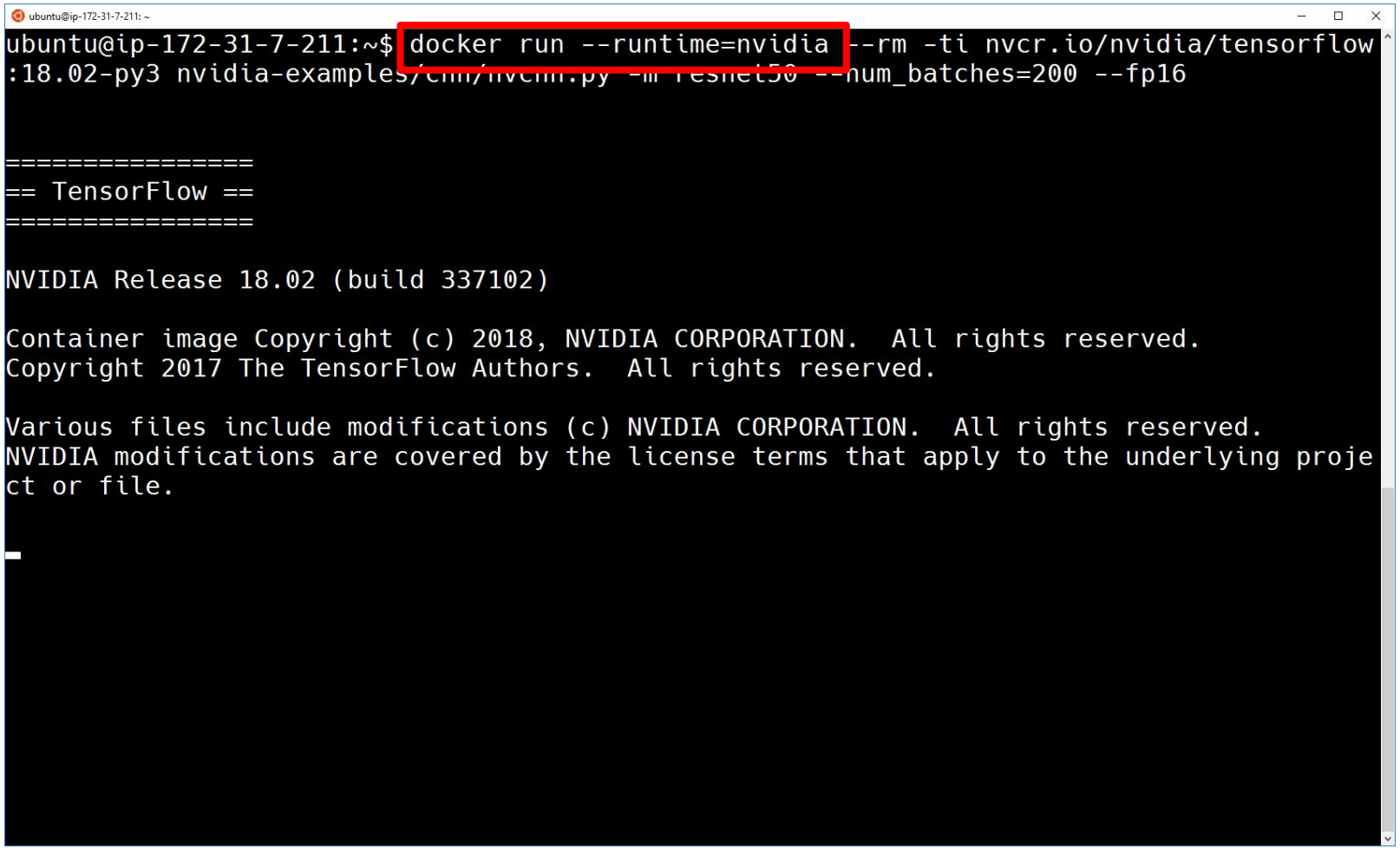
Container Execution



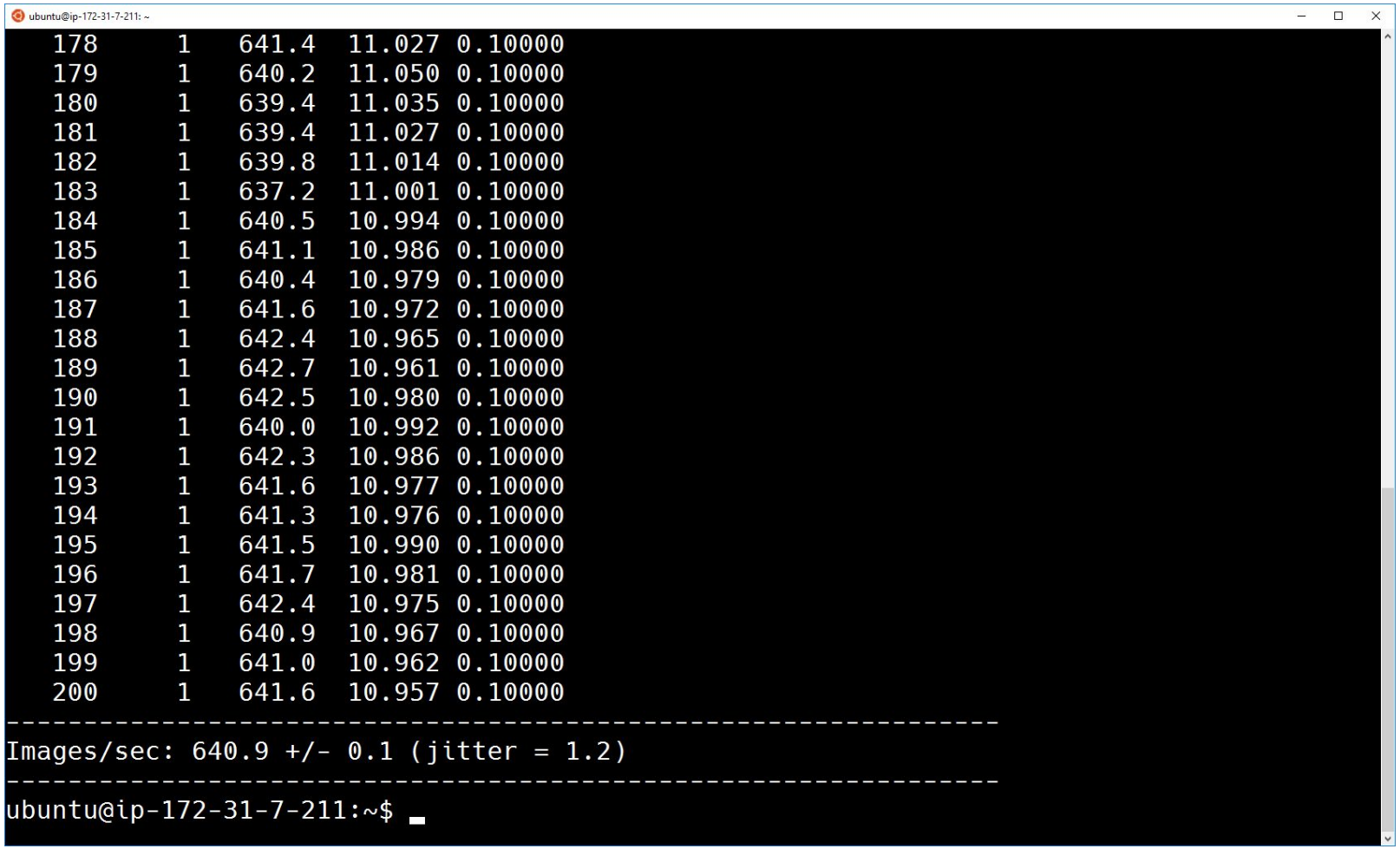
Container Execution



Container Execution



Container Execution



Container Execution

Quick TensorFlow Run

Our challenge:

- Download a TensorFlow container from NGC to local machine
- Run nvcnn.py sample with Resnet-50 with synthetic ImageNet data for 200 epochs
- Hint:

```
# docker run --runtime=nvidia --rm -ti nvcr.io/nvidia/tensorflow:18.04-py3
./nvidia-examples/cnn/nvcnn.py -m resnet50 --num_batches=200 --fp16
```

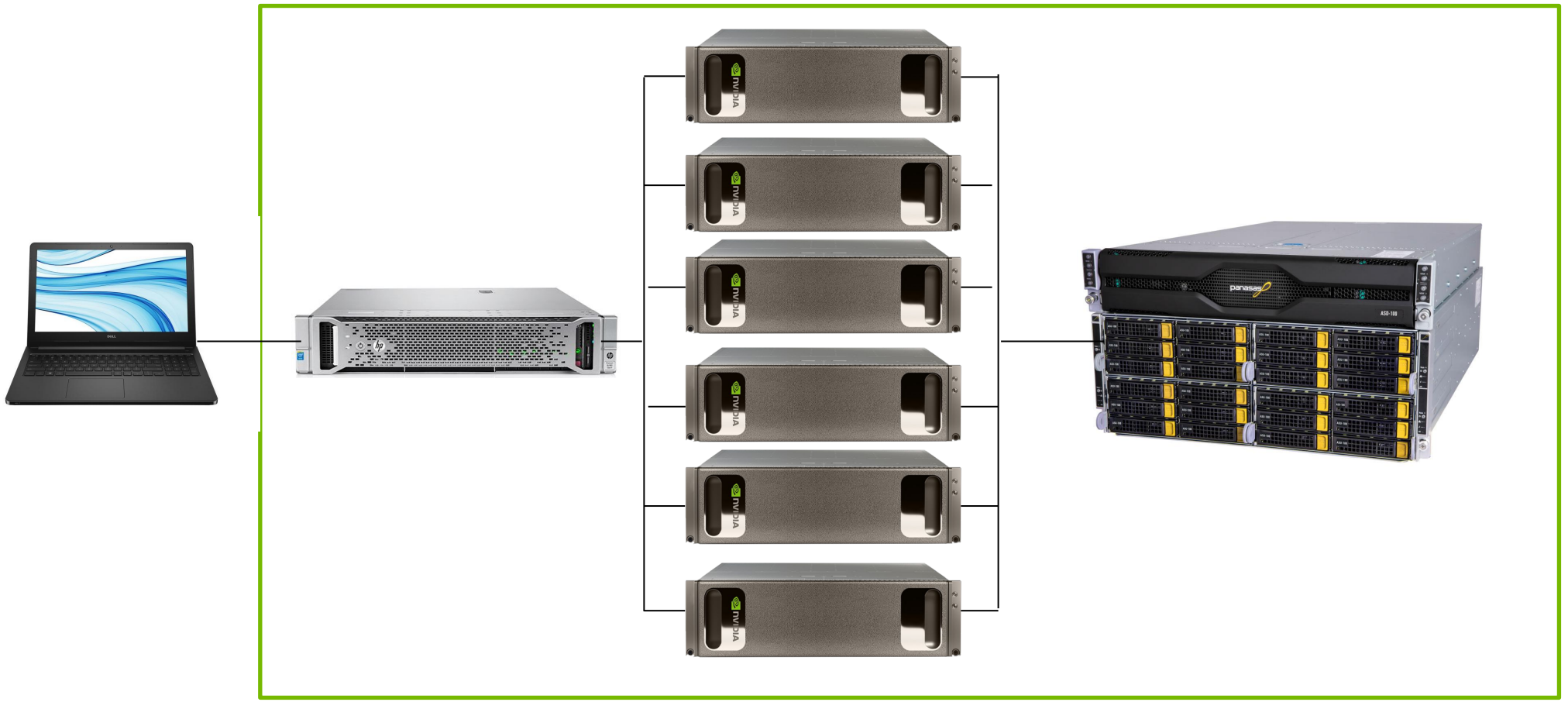
Wait. What Just Happened?

That was too fast.

1. Logged into NGC and created an API key
2. Downloaded the TensorFlow container from NGC
3. Ran nvcnn.py sample with Resnet-50 with synthetic ImageNet data for 200 epochs
4. Profit!

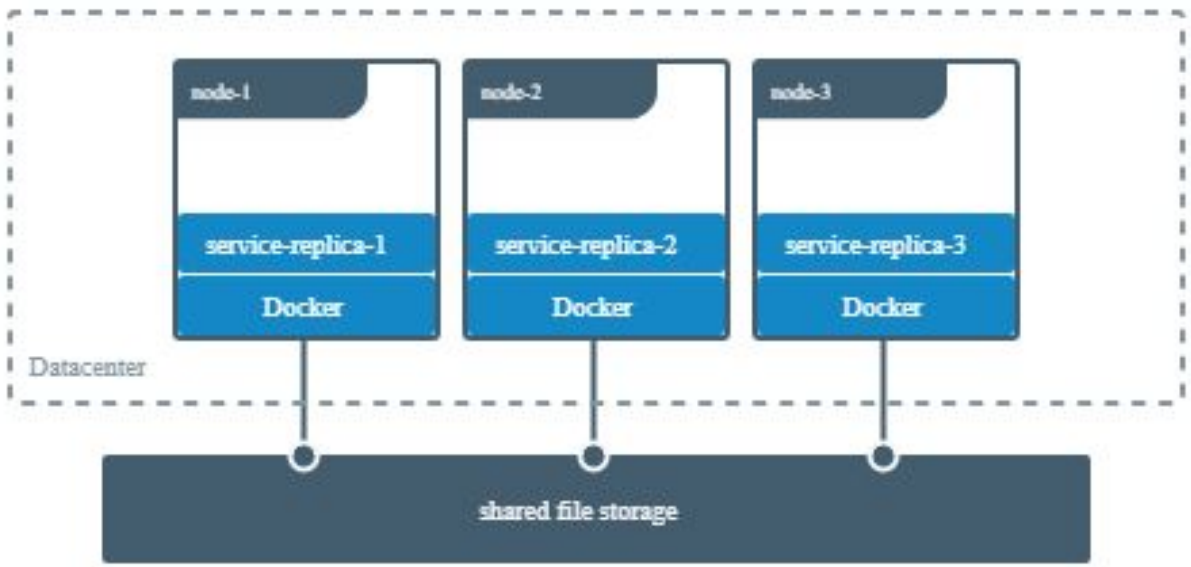
Keeping your Data Persistent: Docker Volumes

HPC Cluster Architecture



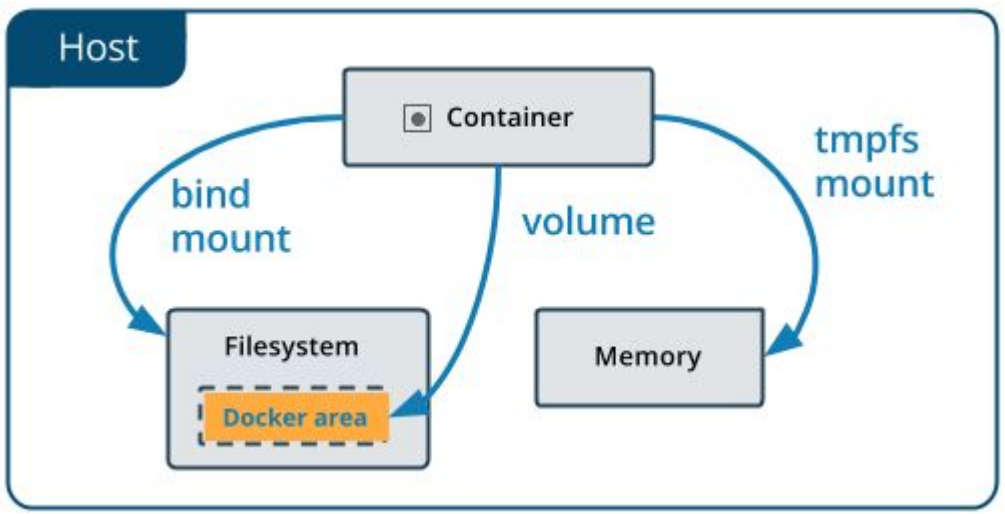
Docker Volume

Persistent Data Across Multiple Containers



Docker Volume

Persistent Data Across Containers Executions




```
docker run \
    --runtime=nvidia \
    --rm -it \
    --name MyCoolContainer \
    -p 8888:80 \
    -v ~/data:/data \
    nvcr.io/nvidia/tensorflow:18.04-py3 \
    bash
```

https://docs.docker.com/storage/volumes/

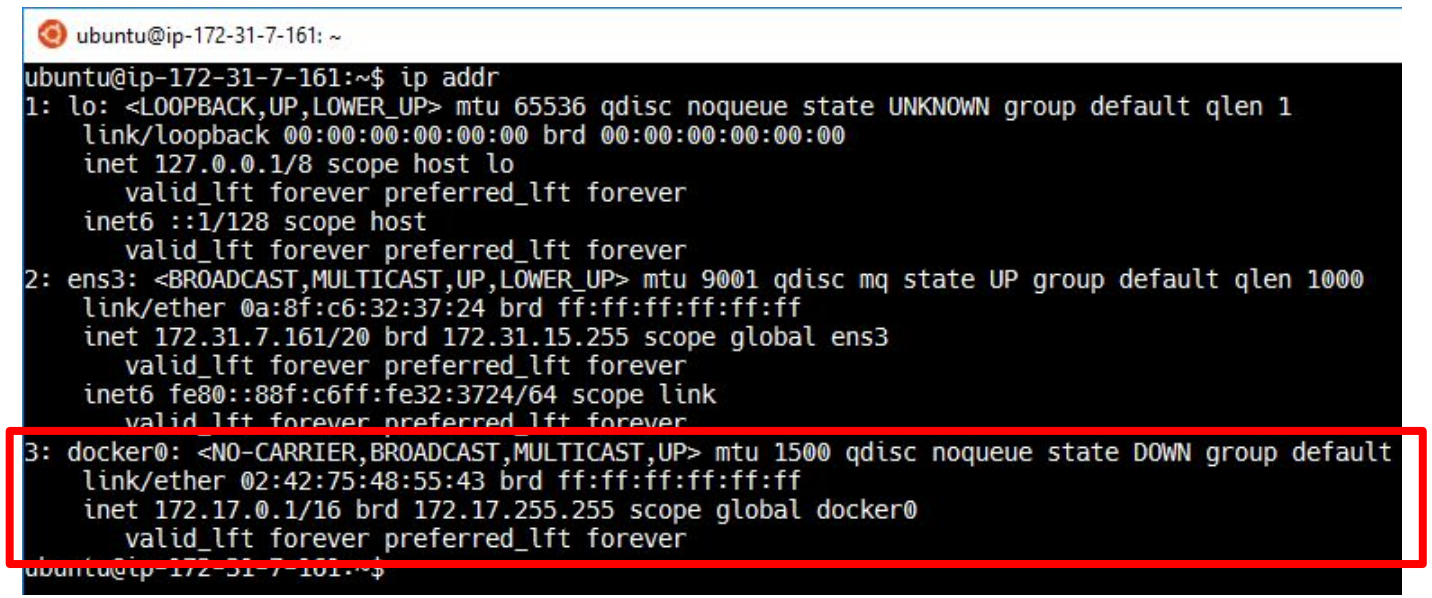
Docker Volume Demo

Exposing Ports

Info: Accessing Container Services

What about things in the container?

Applications in a container are on their own network (‘docker0’ bridge)



Tell Docker you want to use them at runtime (remember `-p` ?)

Challenge: Accessing Container Services

Passing through a TCP port

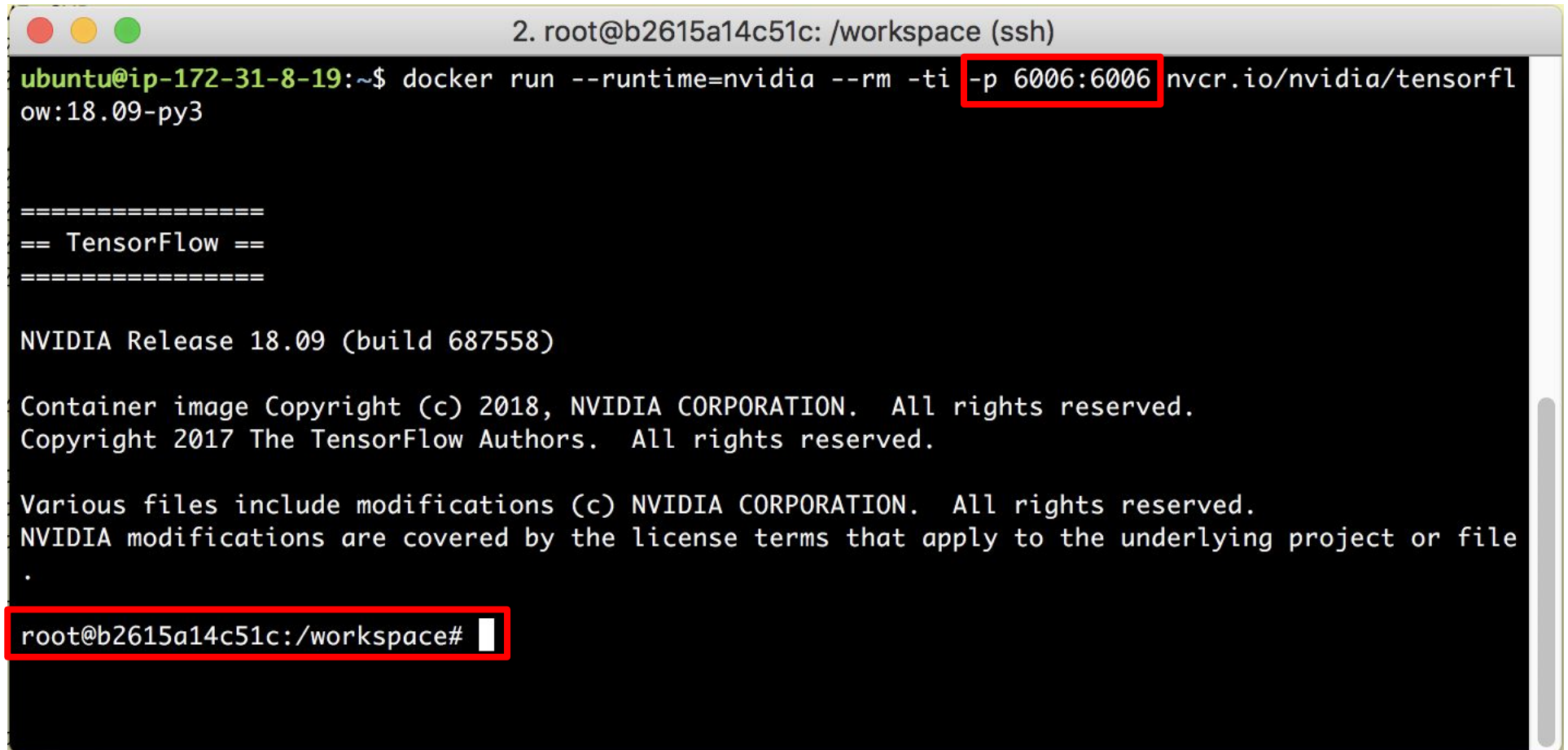
Our challenge:

- Launch the TensorFlow container
  - Run interactively (`-ti`)
  - Expose Tensorboard port 6006 (`-p`)
- Repeat our prior training run
  - Save log data to /tmp
- Run Tensorboard
- Visualize our training run

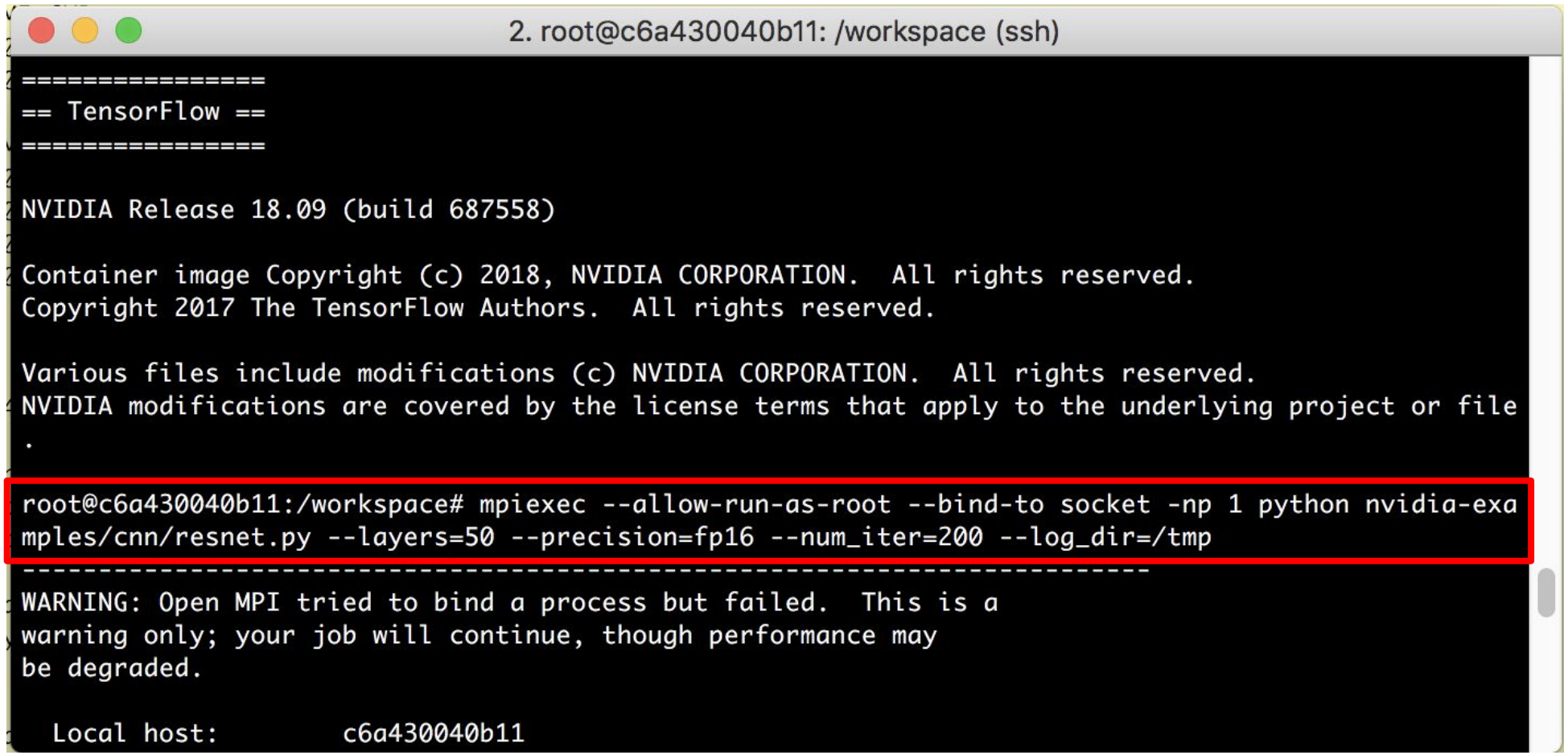
Hint: Remember if you only have 1 GPU in your AMI, you won't be able to flag for 8.

```
# docker run --runtime=nvidia --rm -ti -p 6006:6006 nvcr.io/nvidia/tensorflow:18.09-py3
/workspace# mpiexec --allow-run-as-root --bind-to socket -np 1 python nvidia-examples/cnn/resnet.py
--layers=18 --precision=fp16 --num_iter=200 --log_dir=/tmp --batch_size=64
/workspace# tensorboard --logdir=/tmp
```

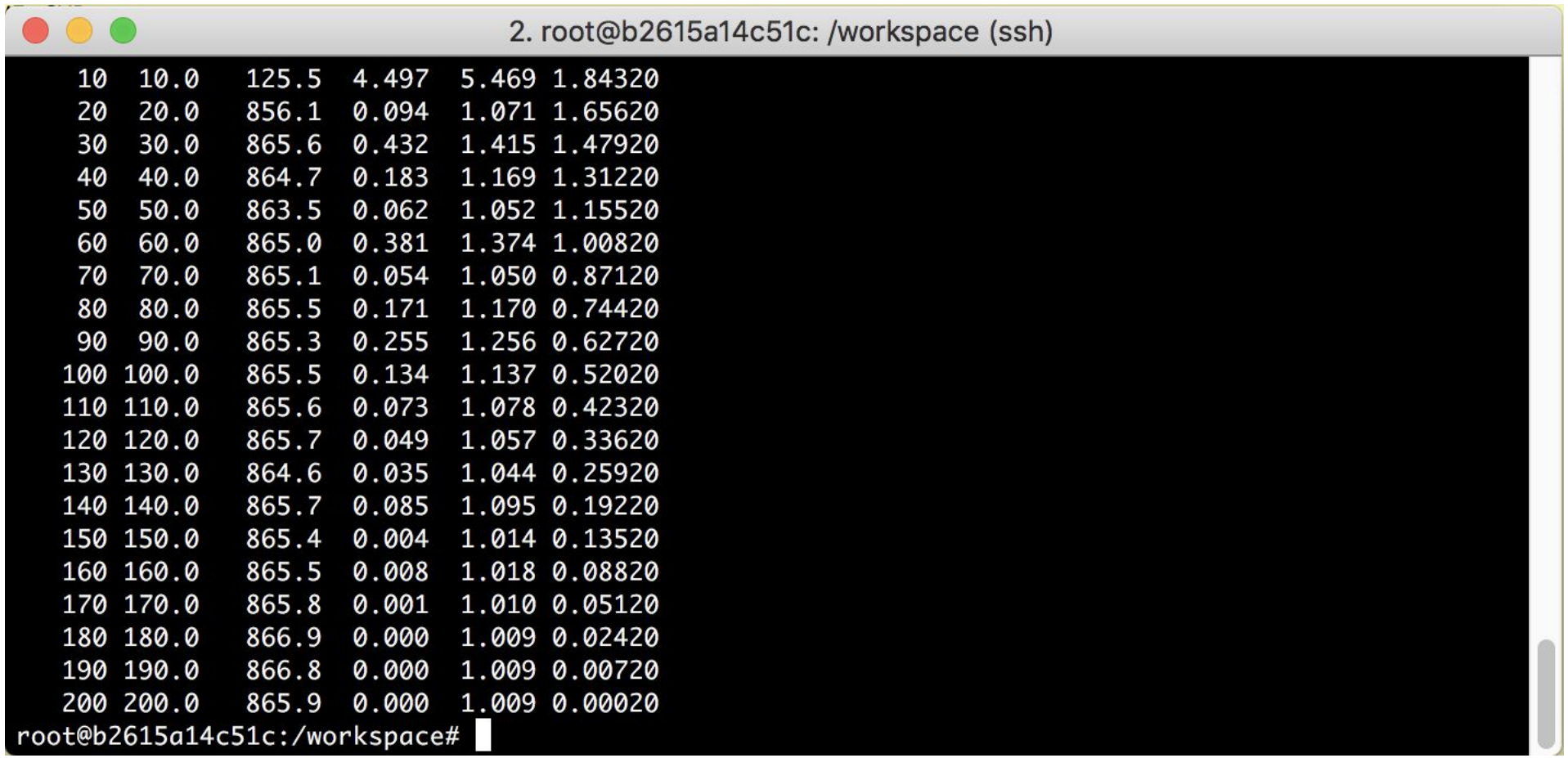
Solution: Accessing Container Services



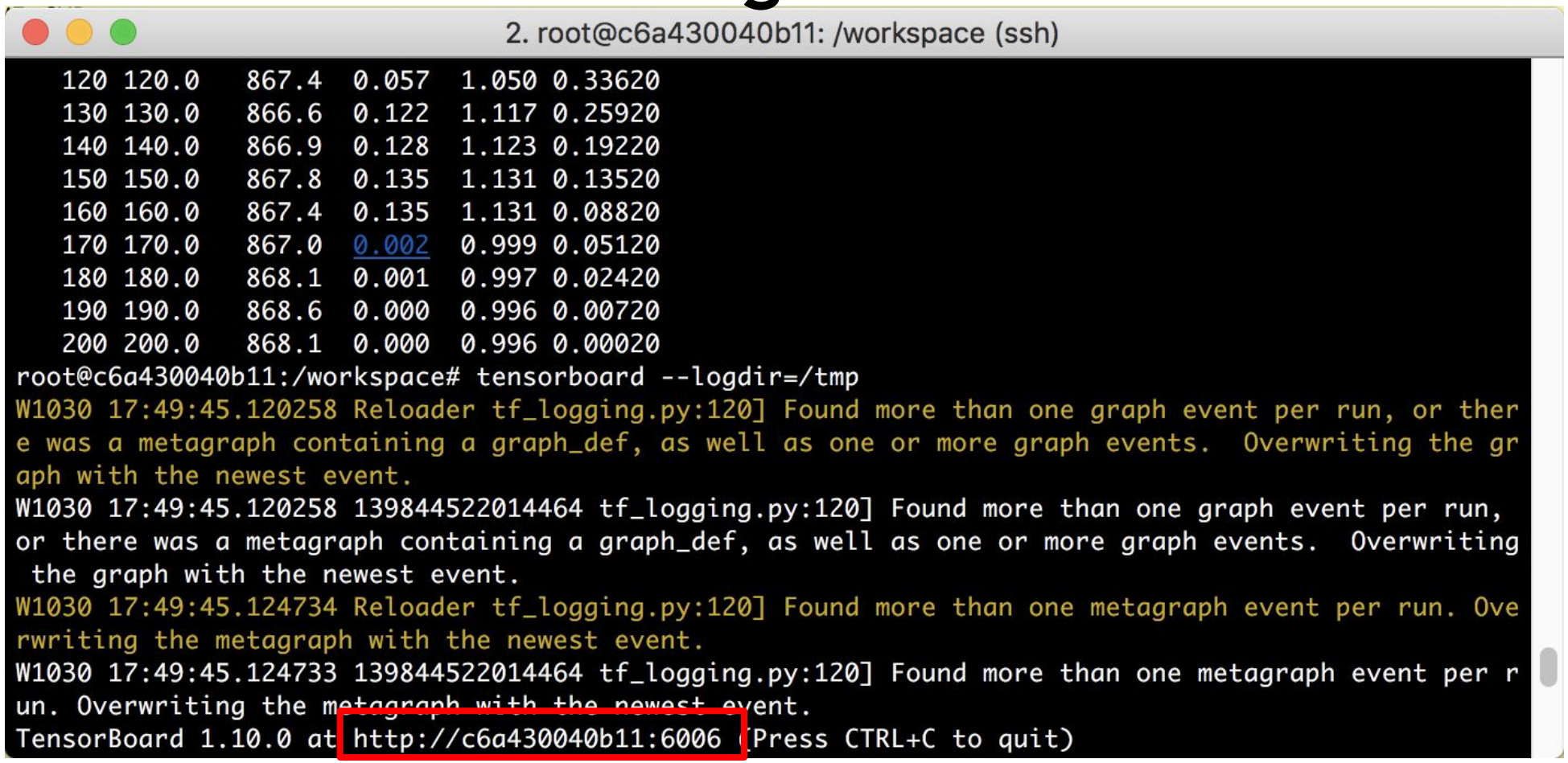
Solution: Accessing Container Services



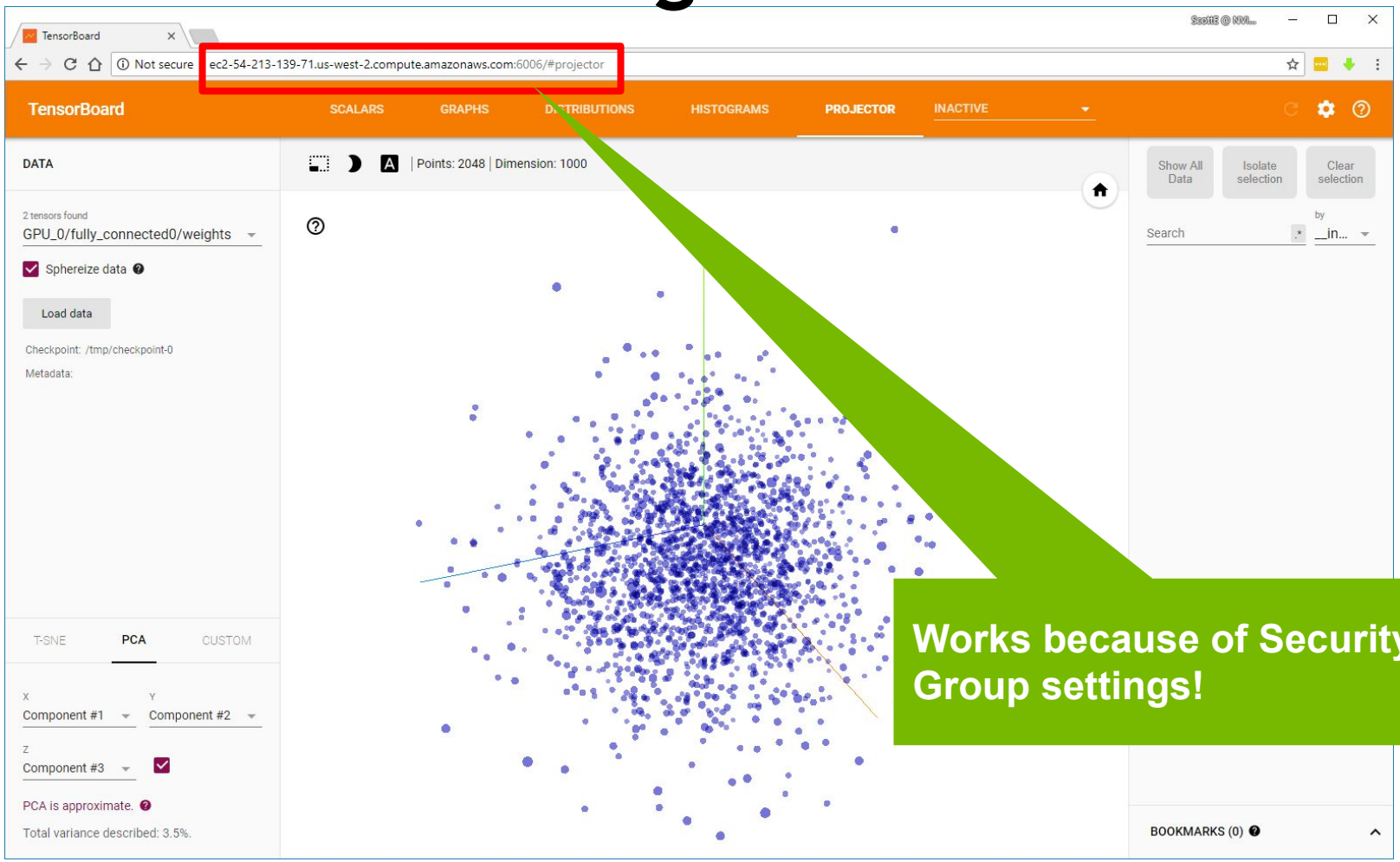
Solution: Accessing Container Services



Solution: Accessing Container Services



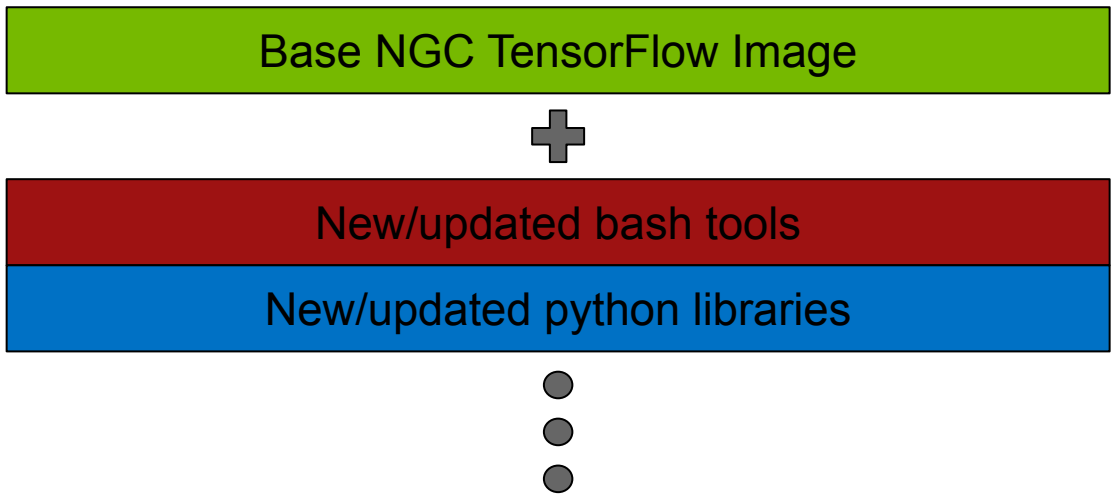
Solution: Accessing Container Services



Extending NGC Images

Info: Extending NGC Images

Layers layers layers



- Often the out-of-the-box image is not enough
- Need extra tools/applications
  - Additional layers on top of base image

- Dockerfile allows for building custom images
  - `docker build` command creates new image from set of instructions

Info: Extending NGC Images

A Dockerfile is a script that contains instructions to custom configure a container from a base image

Here are some common commands:

- FROM is Mandatory as the first instruction. It denotes the base image to be built from. Use a tag to specify the image.
- RUN = Creates a new layer with the output of the specified commands.
- WORKDIR = Directory the command will start it
- CMD = Default command executed when Docker container is started. Use only one CMD instruction in a Dockerfile.

```
1  FROM nvcr.io/nvidia/tensorflow:18.02-py3
2
3  RUN pip install jupyter
4
5  WORKDIR /notebooks
6
7  CMD jupyter notebook --allow-root --ip=0.0.0.0
```

Best practices for writing Dockerfiles

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Challenge: Extending NGC Images

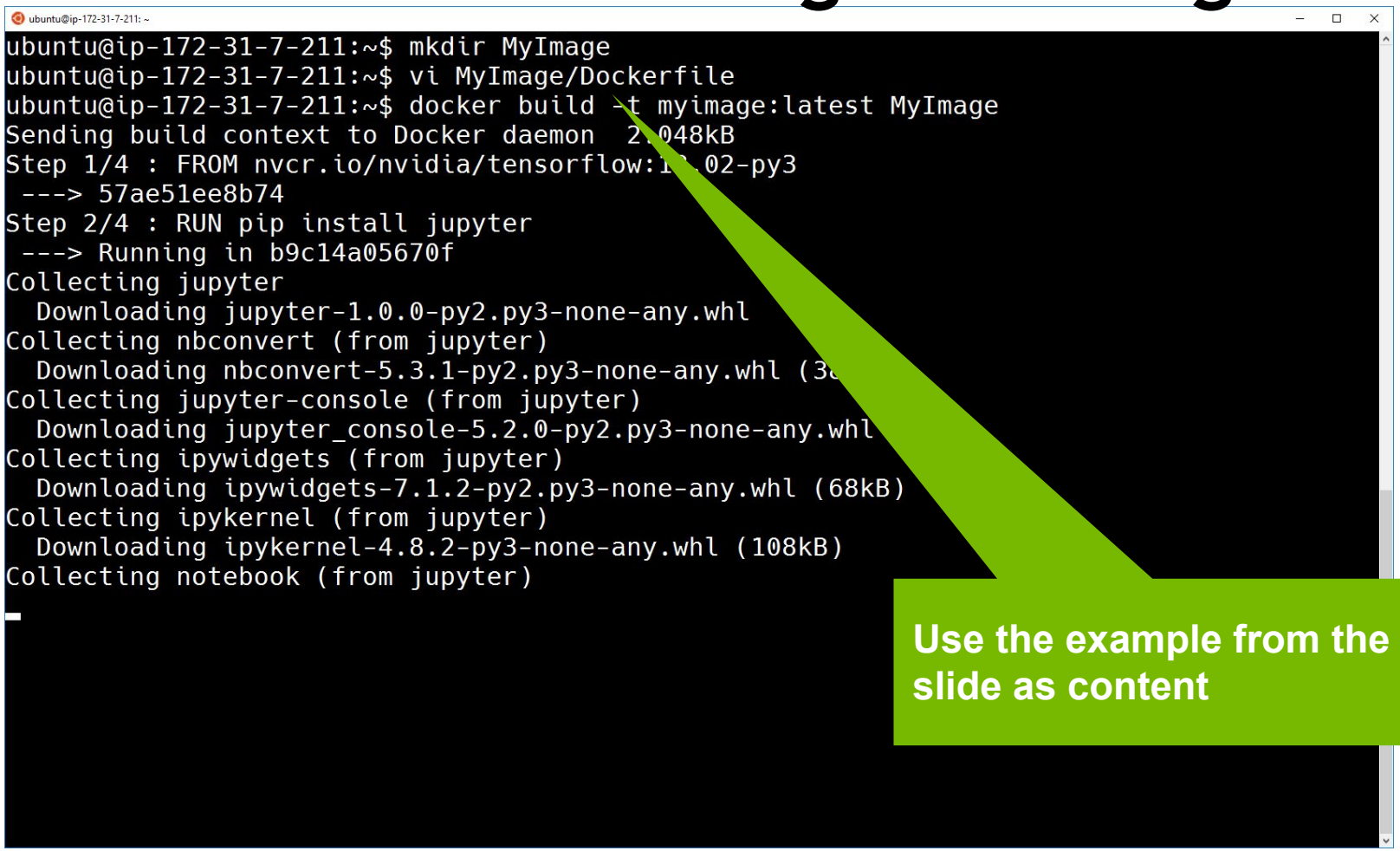
Add Jupyter

Our challenge:

- Add Jupyter to the NVIDIA TensorFlow Image
- Launch jupyter notebook automatically when the container starts
  - By default jupyter listens on port 8888
- Verify it worked!

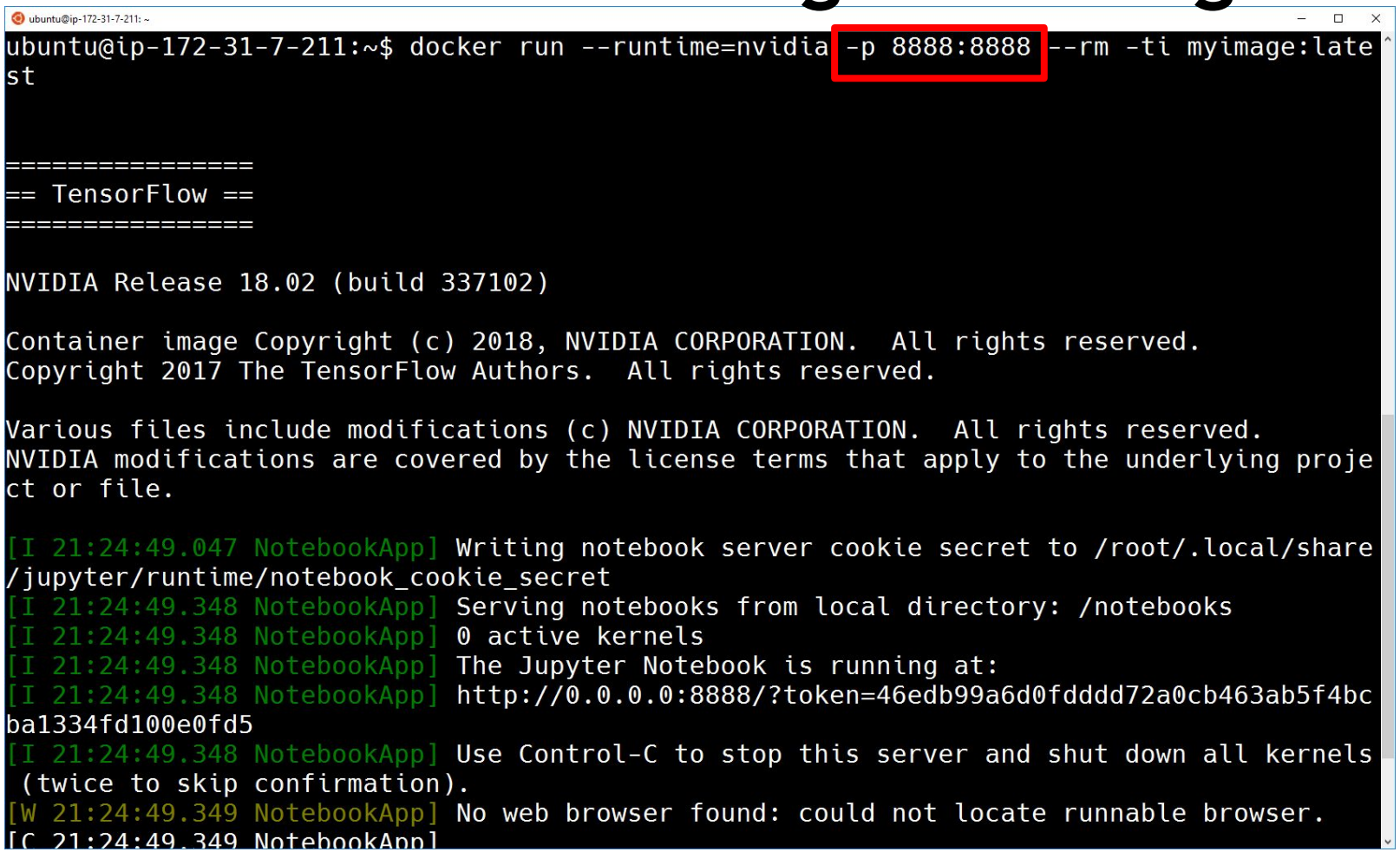
```
# mkdir MyImage
# vi MyImage/Dockerfile

      FROM nvcr.io/nvidia/tensorflow:18.09-py3
      RUN pip install jupyter
      WORKDIR /notebooks
      CMD jupyter notebook --allow-root --ip=0.0.0.0

# docker build -t myimage:latest MyImage
# docker images
# docker run --runtime=nvidia --rm -ti -p 8888:8888 myimage:latest
```

Solution: Extending NGC Images

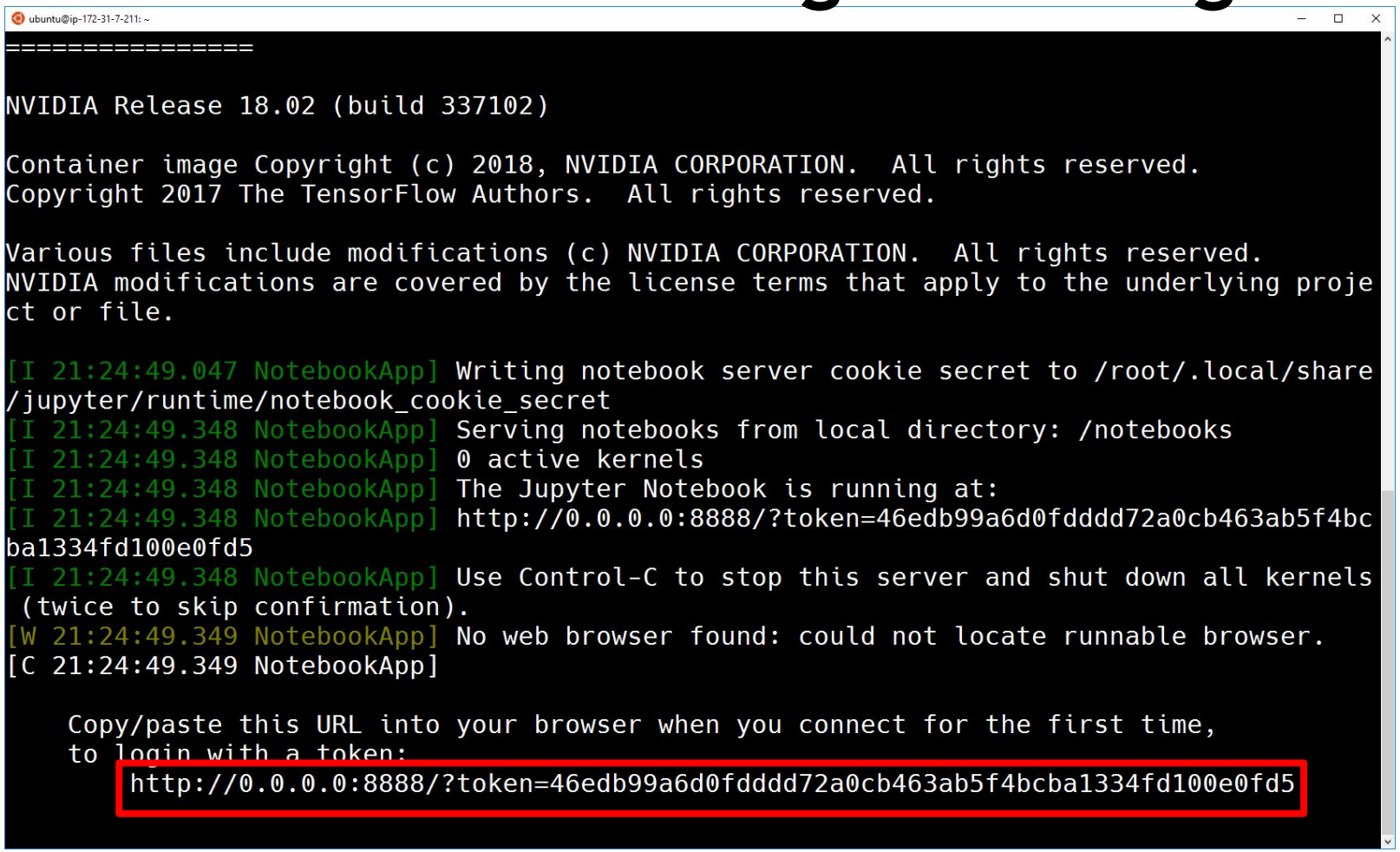


Solution: Extending NGC Images

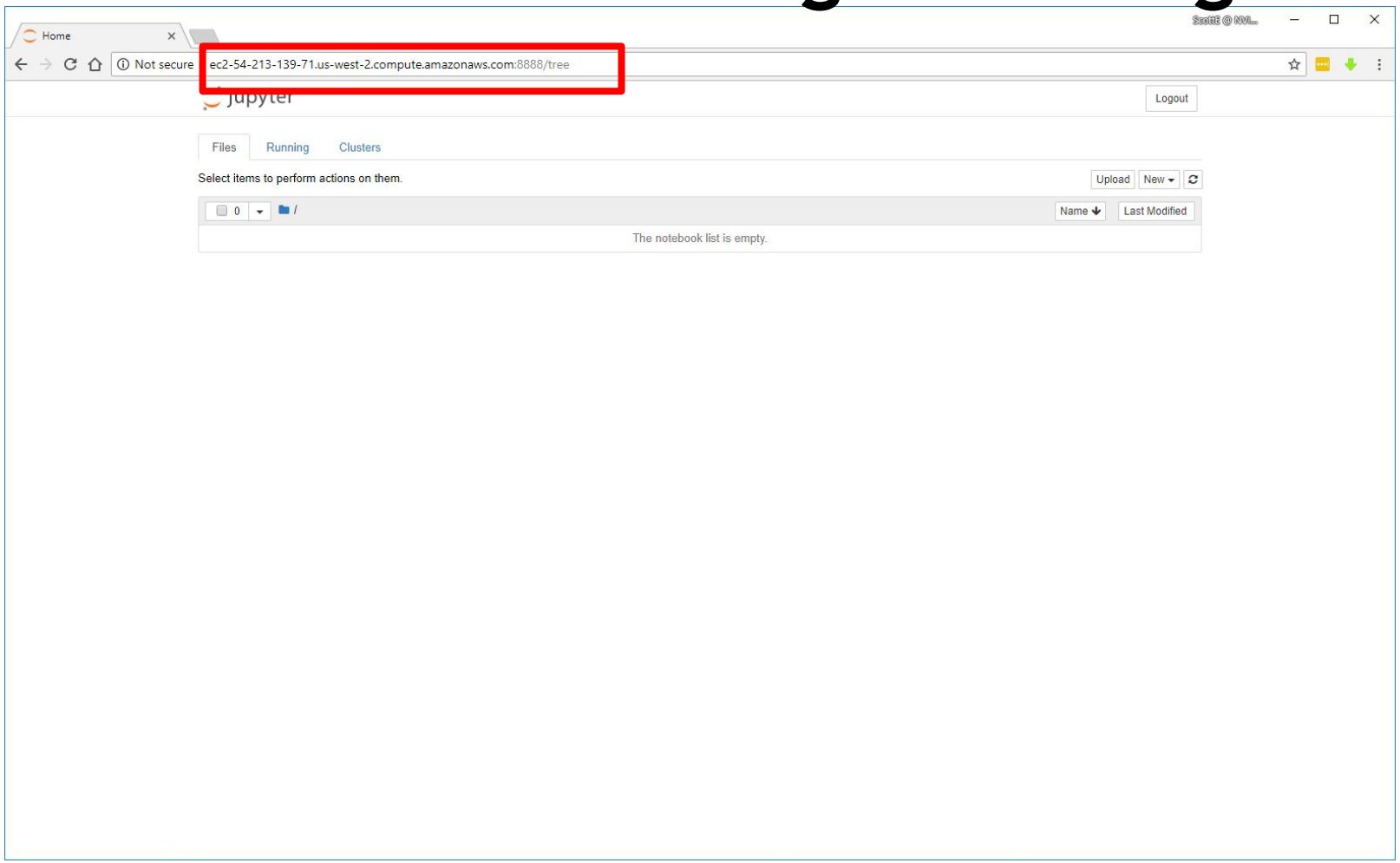
Solution: Extending NGC Images



Solution: Extending NGC Images



Solution: Extending NGC Images



Push an image to a Repository

Summary

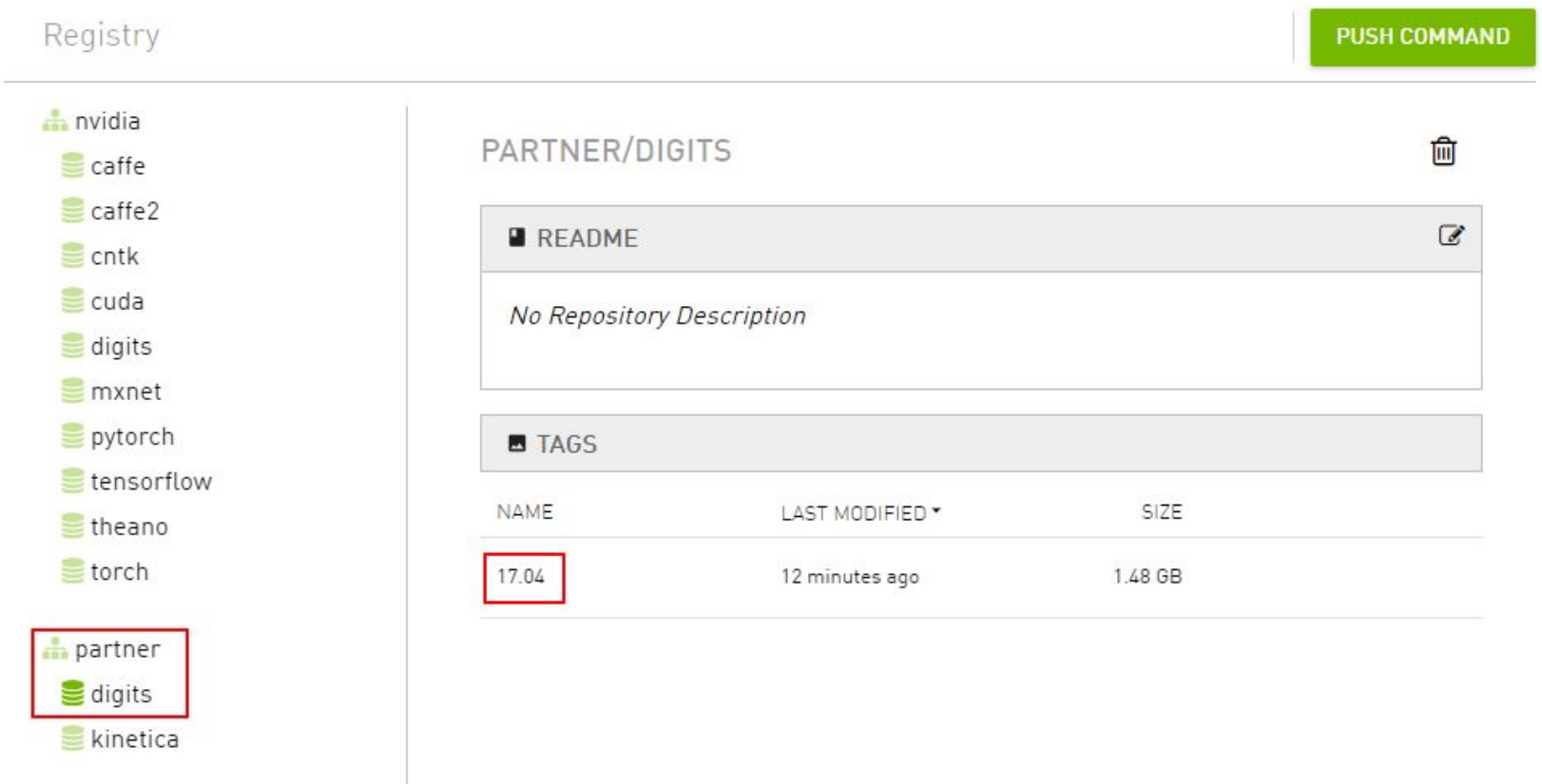
- Push an image or a repository to a registry
- If you are pushing to the NVIDIA DGX container registry an Internet connection is required
- Allows images to be shared between systems

```
$ docker tag nvcr.io/nvidia/digits:17.04
nvcr.io/partner/digits:17.04

$ docker push nvcr.io/partner/digits:17.04
```

Info: Pushing an Image to a Repository

```
/nvidia/cuda     8.0-cudnn6-devel-ubuntu16.04   9a68a1b7ebbc        6 weeks ago         1.616 GB
world            latest                         48b5124b2768        4 months ago        1.84 kB
@dgx-1:~$
@dgx-1:~$ docker tag nvcr.io/nvidia/digits:17.04 nvcr.io/partner/digits:17.04
@dgx-1:~$
@dgx-1:~$ docker images
TORY             TAG                            IMAGE ID            CREATED             SIZE
/nvidia/digits   17.04                          3736f3fe071f        5 weeks ago         4.171 GB
/partner/digits  17.04                          3736f3fe071f        5 weeks ago         4.171 GB
/nvidia/cuda     8.0-cudnn6-devel-ubuntu16.04   9a68a1b7ebbc        6 weeks ago         1.616 GB
world            latest                         48b5124b2768        4 months ago        1.84 kB
@dgx-1:~$
@dgx-1:~$ docker push nvcr.io/partner/digits:17.04
sh refers to a repository [nvcr.io/partner/digits]
975e2d: Pushed
96b8a8: Pushed
3a11af: Pushed
783618: Pushed
372284: Pushed
1ab8fe: Pushed
40342a: Pushed
e31131: Pushed
3a58f3: Pushed
379d53: Pushed
657c24: Pushed
e22efa: Pushed
753c6b: Pushed
6d4f80: Pushed
56827159aa8b: Pushed
440e02c3dcde: Pushed
29660d0e5bb2: Pushed
85782553e37a: Pushed
745f5be9952c: Pushed
17.04: digest: sha256:b5da6bc51bf0da3f414db691cbee8030dfcf1659d9d95308b210a77e95b9df08 size: 15193
dgxuser@dgx-1:~$
```

Info: Pushing an Image to a Repository



GPU Isolation

GPU Isolation

By default, NVIDIA-DOCKER will grant access to all GPUs in the system

Use NV_GPU to assign specific GPUs to the running container

Examples of GPU isolation:

Running nvidia-docker isolating specific GPUs by index

```
NV_GPU=0,1,5 nvidia-docker run --rm -it
nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04
```

Running nvidia-docker isolating specific GPUs by UUID

```
NV_GPU='GPU-836c0c09,GPU-b78a60a' nvidia-docker <docker-options>
<docker-command> <docker-args>
```

Challenge: GPU Isolation

Individual Challenge

- For the two GPU Isolation challenges below run `nvidia-smi` against the image you pulled so the container is removed automatically when the command finishes
- Use nvidia-docker with GPU isolation
  - GPU Isolation - Run nvidia-smi with only two (2) GPU's
  - GPU Isolation - Run nvidia-smi interactively with three (3) GPUs

hint: https://github.com/NVIDIA/nvidia-docker/wiki/GPU-isolation

Solution: GPU Isolation

Summary

GPU Isolation - Run nvidia-smi with two (2) GPU's

```
$ NV_GPU=0,5 nvidia-docker run --rm
nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04 nvidia-smi
```

GPU Isolation - Run nvidia-smi with three (3) GPU's

```
$ NV_GPU=0,1,4 nvidia-docker run --rm -it
nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04
root@<container ID>:/# nvidia-smi
root@<container ID>:/# exit
```

Solution: GPU Isolation

GPU Isolation with two (2) GPUs

```
dgxuser@dvt8:~$ NV_GPU=0,5 nvidia-docker run --rm nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04 nvidia-smi
Fri Jun 23 21:31:06 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.66                 Driver Version: 375.66                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-SXM2...  Off  | 0000:06:00.0     Off |                  Off |
| N/A   31C    P0    32W / 300W |      0MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  Off  | 0000:86:00.0     Off |                  Off |
| N/A   32C    P0    31W / 300W |      0MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
dgxuser@dvt8:~$
```

Solution: GPU Isolation

GPU Isolation interactively with three (3) GPUs

```
dgxuser@dgx-1:~$ NV_GPU=0,1,4 nvidia-docker run --rm -it nvcr.io/nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04
root@22e93990a493:/# nvidia-smi
Mon Apr 17 04:19:15 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.20                 Driver Version: 375.20                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-SXM2...  Off  | 0000:06:00.0     Off |                    0 |
| N/A   32C    P0    30W / 300W |      8MiB / 16308MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  Off  | 0000:07:00.0     Off |                    0 |
| N/A   29C    P0    33W / 300W |      8MiB / 16308MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-SXM2...  Off  | 0000:85:00.0     Off |                    0 |
| N/A   31C    P0    34W / 300W |      8MiB / 16308MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
root@22e93990a493:/# exit
exit
dgxuser@dgx-1:~$
```

CONTAINERS

João Paulo Navarro (jpnavarro@nvidia.com)
Solutions Architect