
CyVerse Documentation

Release 0.1.0

CyVerse

Sep 18, 2020

AstroContainers Workshop

1	Workshop Code of Conduct	3
2	Pre-Workshop Setup	5
3	Agenda	9
4	About CyVerse	11
5	Training session in Docker	13
6	Training session in Singularity	15
7	Introduction to Docker	17
8	Advanced Docker	27
9	Additional Docker Demo's	51
10	Introduction to Singularity	55
11	Advanced Singularity	71
12	Bootng an Atmosphere computer instance for your use!	77
13	Docker related resources	89
14	Singularity related resources	91
15	Other resources	93
16	For instructors!	95
17	Problems? Bugs? Questions?	99



Workshop Code of Conduct

All attendees, speakers, sponsors and volunteers at our workshop are required to follow this code of conduct. Organizers will enforce this code throughout the event. We expect cooperation from all participants to help ensure a safe environment for everyone.

This Code of Conduct is taken from <http://confcodeofconduct.com/>. See <http://www.ashedryden.com/blog/codes-of-conduct-101-faq> for more information on codes of conduct.

FAIR principles

AstroContainers supports FAIR data principles by providing services that help make data Findable, Accessible, Interoperable, and Reusable. Participants will get an introduction to containers and learn how to create and manage containers.

Learning objectives

Participants will learn key containerization concepts for developing reproducible analysis pipelines, with emphasis on container lifecycle management from design to execution and scaling.

The workshop will cover key concepts about containers such as defining the architecture of containers, building images and pushing them to public and private repositories as well as how to scale your analysis from laptop to cloud and to HPC systems using containers.

Who should attend?

Faculty, researchers, postdocs, and graduate students who use and analyze data of all types (genomics, image data, from animals, plants, etc.).

Workshop level

This workshop is focused on beginner-level users with little to no previous container experience.

Intermediate and advanced users who attend will gain a better understanding of and ability with container capabilities and resources, including deploying their own tools and extending these analyses into Cloud and HPC.

Need help?

Couldn't find what you were looking for?

- You can reach the lead instructor, Chan, chikwan, at chanc@email.arizona.edu.

- You can also talk to any of the instructors or TAs if you need immediate help.
- Chat with us and our community on Slack.
- Post an issue on the documentation [issue tracker](#) on GitHub

CHAPTER 2

Pre-Workshop Setup

Please complete the minimum Setup Instructions to prepare for the CyVerse AstroContainers Workshop at the University of Arizona, which will run on May 7th and 8th, 2018.

Prerequisite	Notes	Links
Wi-Fi-enabled laptop	You should be able to use any laptop using Windows/MacOS/Linux. We strongly recommend Firefox or Chrome browser; We do not recommend Microsoft Edge Browser . It is recommended that you have administrative/install permissions on your laptop.	<ul style="list-style-type: none"> • Download FireFox • Download Chrome
CyVerse Account	Please ensure that you have a CyVerse account and have verified your account by completing the verification steps in the email you got when you registered.	Register for your cyverse account at http://user.cyverse.org/ . You can test your account by logging into http://user.cyverse.org/ .
Atmosphere Cloud	The Atmosphere Cloud service is part of CyVerse but it requires explicit activation. Please ensure that you have access to CyVerse Atmosphere Cloud.	The Atmosphere Cloud service should be listed in this webpage once you log in to CyVerse.
Github Account	Please ensure that you have a Github account	Register for your Github account at https://github.com/ .
Docker	Install Docker on your computer	<ul style="list-style-type: none"> • Docker for mac • Docker for windows • Docker for Linux
Dockerhub Account	Please ensure that you have a Dockerhub account	Register for your Dockerhub account at https://hub.docker.com/ .
UA HPC Account	Please ensure that you have a UA HPC account. Faculty and Principal Investigators (PI) can request and authorize their own HPC accounts. Students and Staff requesting HPC Account must be authorized/sponsored by a Faculty member or a PI.	Please follow the instruction here .
Text Editor	Please ensure that you have a Text editor of your choice. Any decent text editor would be sufficient and recommended ones include Sublime2 and Atom	<ul style="list-style-type: none"> • Register for Sublime at https://www.sublimetext.com/. • Register for Atom at https://atom.io/.
Slack for networking	We will be using Slack extensively for communication and networking purposes	Register for Slack at https://slack.com/ .

Optional Download Extras

Listed below are some extra downloads that aren't required for the workshop, but which provide some options for functionalities we will cover.

Tool	Notes	Link
SSH Clients (Windows)	PuTTY allows SSH connection to a remote machine, and is designed for Windows users who do not have a Mac/Linux terminal. MobaXterm is a single Windows application that provides a ton of functions for programmers, webmasters, IT administrators, and anybody is looking to manage system remotely	<ul style="list-style-type: none">• Download PuTTY• Download mobaXterm• Update Windows 10 to use Linux Bash
Cyberduck	Cyberduck is a third-party tool for uploading/downloading data to CyVerse Data Store. Currently, this tool is available for Windows/MacOS only. You will need to download Cyberduck and the connection profile. We will go through configuration and installation at the workshop.	<ul style="list-style-type: none">• Download Cyberduck• Download CyVerse Cyberduck connection profile
iCommands	iCommands are third-party software for command-line connection to the CyVerse Data Store.	Download and installation instructions available at CyVerse Learning Center

CHAPTER 3

Agenda

Below are the schedule and classroom materials for CyVerse AstroContainers Workshop, which will run from May 7th and 8th, 2018.

This workshop runs under a [Code of Conduct](#). Please respect it and be excellent to each other!

Day	Time	Topic/Activity	Notes/Links
05/07/18 (Monday)	09:00am-09:15am	Introduction to CyVerse and Data7 (Nirav Merchant)	Intro slides
	09:15am-10:00am	Container landscape and computation resources (Nirav Merchant)	Intro slides
	10:00am-10:15am	Coffee and snack break with networking	
	10:15am-11:00am	Science code and Science Platform: Docker at LSST (Frossie Economou)	
	11:00am-12:00pm	Introductory Docker (CK Chan)	<ul style="list-style-type: none"> • Docker intro slides • Docker intro demo
	12:00pm-01:00pm	Lunch break on your own	
	01:00pm-2:30pm	Advanced Docker (Upen-dra Devisetty)	Advanced docker
	2:30pm-3:00pm	Coffee break (15 min) and afternoon session planning	
	3:00pm-5:00pm	BYOD/A (Hands-on project)	
05/08/18 (Tuesday)	09:00am-9:15am	General overview of Singularity (Tyson Swetnam)	Singularity Overview Slides Gitpitch slides
	9:15am-10:00am	Singularity setup (Tyson Swetnam)	Singularity Introduction
	10:00am-10:15am	Coffee and snack break with networking	
	10:15am-10:45am	EHT, PCA, and Singularity (Lia Medeiros)	
	10:45am-12:00pm	Introductory Singularity cont.. (Tyson Swetnam)	<ul style="list-style-type: none"> • University of Arizona High Performance Computing • Singularity-Hub
	12:00pm-01:00pm	Lunch break on your own	
	01:00pm-2:30pm	Advanced Singularity (Chris Reidy)	Advanced Singularity
	2:30pm-3:00pm	Coffee break (15 min) and afternoon session planning	
	3:00pm-5:00pm	BYOD/A (Hands-on project)	

CHAPTER 4

About CyVerse

CyVerse Vision: Transforming science through data-driven discovery.

CyVerse Mission: Design, deploy, and expand a national cyberinfrastructure for life sciences research and train scientists in its use. CyVerse provides life scientists with powerful computational infrastructure to handle huge datasets and complex analyses, thus enabling data-driven discovery. Our powerful extensible platforms provide data storage, bioinformatics tools, image analyses, cloud services, APIs, and more.

While originally created with the name iPlant Collaborative to serve U.S. plant science communities, CyVerse cyberinfrastructure is germane to all life sciences disciplines and works equally well on data from plants, animals, or microbes. By democratizing access to supercomputing capabilities, we provide a crucial resource to enable scientists to find solutions for the future. CyVerse is of, by, and for the community, and community-driven needs shape our mission. We rely on your feedback to provide the infrastructure you need most to advance your science, development, and educational agenda.

CyVerse Homepage: <http://www.cyverse.org>

Funding and Citations

CyVerse is funded entirely by the National Science Foundation under Award Numbers DBI-0735191 and DBI-1265383.

Please cite CyVerse appropriately when you make use of our resources, [CyVerse citation policy](#)

Training session in Docker

Trainers (CK Chan and Upendra Devisetty)

In these two sessions, we will explain the various aspects of Docker. Starting with the basics such as starting containers and managing data using volumes. We will then gradually move to advanced topics such as creating your own images, pushing and pulling them from registries.

- [Docker basics/Introduction \(CK\)](#)

This is the introductory session of Docker. The topics include Docker installation, running prebuilt Docker containers, managing data in Docker, and exposing container ports. This session gets you ready to run most of the existing containers.

- [Advanced Docker \(Upendra\)](#)

This is the advanced session of Docker. The topics include building your own Docker images, pushing and pulling Docker them to public and private registries, automated Docker image building from github/bitbucket repositories, Docker compose for building multiple Docker containers, etc.

- [Additional Docker Demo's](#)

We also provide two additional Docker Demo's for the participants to try.

Training session in Singularity

Trainers (Tyson Swetnam and Chris Reidy)

In this session we will show you how to containerize your software/applications using Singularity, push them to Singularityhub and deploy them on cloud and HPC.

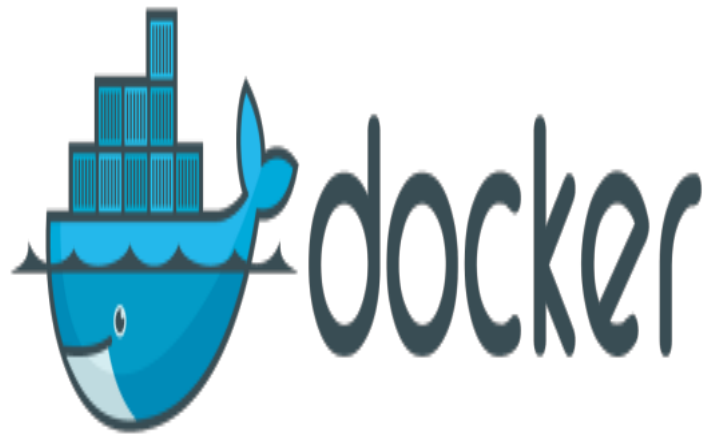
- [Singularity basics/Introduction \(Tyson Swetnam\)](#)

This would be the introductory session for concept of Singularity. The topics include installation Singularity on various platforms, running prebuilt singularity containers, building singularity containers locally etc.

- [Advanced Singularity \(Chris Reidy\)](#)

This is the advanced session for the concept of Singularity. The topics include pushing and pulling Singularity images to and from Singularity hub, converting Docker containers to Singularity containers, mounting data on to Singularity containers etc.

Introduction to Docker



7.1 1. Prerequisites

There are no specific skills needed for this tutorial beyond a basic comfort with the command line and using a text editor. Prior experience in python will be helpful but is not required.

7.2 2. Docker Installation

Getting all the tooling setup on your computer can be a daunting task, but not with Docker. Getting Docker up and running on your favorite OS (Mac/Windows/Linux) is very easy.

The getting started guide on Docker has detailed instructions for setting up Docker on [Mac/Windows/Linux](#).

Note: If you're using Docker for Windows make sure you have [shared your drive](#).

If you're using an older version of Windows or MacOS you may need to use [Docker Machine](#) instead.

All commands work in either bash or Powershell on Windows.

Once you are done installing Docker, test your Docker installation by running the following command to make sure you are using version 1.13 or higher:

```
$ docker --version
Docker version 17.09.0-ce, build afdb6d4
```

When run without `--version` you should see a whole bunch of lines showing the different options available with docker. Alternatively you can test your installation by running the following:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7
Status: Downloaded newer image for hello-world:latest

Hello from Docker.
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
.....
```

Note: Depending on how you've installed Docker on your system, you might see a `permission denied` error after running the above command. If you're on Linux, you may need to prefix your Docker commands with `sudo`. Alternatively to run docker command without `sudo`, you need to add your user (who has root privileges) to docker group. For this run:

Create the docker group:

```
$ sudo groupadd docker
```

Add your user to the docker group:

```
$ sudo usermod -aG docker $USER
```

Log out and log back in so that your group membership is re-evaluated

7.3 3. Running Docker containers

Now that you have everything setup, it's time to get our hands dirty. In this section, you are going to run a container from [Alpine Linux](#) (a lightweight linux distribution) image on your system and get a taste of the `docker run` command.

But wait, what exactly is a container and image?

Containers - Running instances of Docker images - containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS.

Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run `docker inspect hello-world` and `docker history hello-world`. In the demo above, you could have used the `docker pull` command to download the `hello-world` image. However when you executed the command `docker run hello-world`, it also did a `docker pull` behind the scenes to download the `hello-world` image with `latest` tag (we will learn more about tags little later).

Now that we know what a container and image is, let's run the following command in our terminal:

```
$ docker run alpine ls -l
total 52
drwxr-xr-x    2 root    root          4096 Dec 26  2016 bin
drwxr-xr-x    5 root    root           340 Jan 28  09:52 dev
drwxr-xr-x   14 root    root          4096 Jan 28  09:52 etc
drwxr-xr-x    2 root    root          4096 Dec 26  2016 home
drwxr-xr-x    5 root    root          4096 Dec 26  2016 lib
drwxr-xr-x    5 root    root          4096 Dec 26  2016 media
.....
```

Similar to `docker run hello-world` command in the demo above, `docker run alpine ls -l` command fetches the `alpine:latest` image from the Docker registry first, saves it in our system and then runs a container from that saved image.

When you run `docker run alpine`, you provided a command `ls -l`, so Docker started the command specified and you saw the listing

You can use the `docker images` command to see a list of all images on your system

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
alpine              latest             c51f86c28340       4 weeks ago        109 MB
hello-world         latest             690ed74de00f       5 months ago        960 B
```

Let's try something more exciting.

```
$ docker run alpine echo "Hello world"
Hello world
```

OK, that's some actual output. In this case, the Docker client dutifully ran the `echo` command in our `alpine` container and then exited it. If you've noticed, all of that happened pretty quickly. Imagine booting up a virtual machine, running a command and then killing it. Now you know why they say containers are fast!

Try another command.

```
$ docker run alpine sh
```

Wait, nothing happened! Is that a bug? Well, no. These interactive shells will exit after running any scripted commands such as `sh`, unless they are run in an interactive terminal - so for this example to not exit, you need to `docker run -it alpine sh`. You are now inside the container shell and you can try out a few commands like `ls -l`, `uname -a` and others.

Before doing that, now it's time to see the `docker ps` command which shows you all containers that are currently running.

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
↳ STATUS	PORTS	NAMES	

Since no containers are running, you see a blank line. Let's try a more useful variant: `docker ps -a`

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
↳ STATUS	PORTS	NAMES	
36171a5da744	alpine	"/bin/sh"	5 minutes ago
↳ Exited (0) 2 minutes ago		fervent_newton	
a6a9d46d0b2f	alpine	"echo 'hello from alp"	6 minutes ago
↳ Exited (0) 6 minutes ago		lonely_kilby	
ff0a5c3750b9	alpine	"ls -l"	8 minutes ago
↳ Exited (0) 8 minutes ago		elated_ramanujan	
c317d0a9e3d2	hello-world	"/hello"	34 seconds ago
↳ Exited (0) 12 minutes ago		stupefied_mcclintock	

What you see above is a list of all containers that you ran. Notice that the STATUS column shows that these containers exited a few minutes ago.

If you want to run scripted commands such as `sh`, they should be run in an interactive terminal. In addition, interactive terminal allows you to run more than one command in a container. Let's try that now:

```
$ docker run -it alpine sh
/ # ls
bin      dev      etc      home     lib      media    mnt      proc     root     run      sbin     srv
↳ sys    tmp      usr      var
/ # uname -a
Linux de4bbc3eeaec 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 Linux
```

Running the `run` command with the `-it` flags attaches us to an interactive `tty` in the container. Now you can run as many commands in the container as you want. Take some time to run your favorite commands.

Exit out of the container by giving the `exit` command.

```
/ # exit
```

Note: If you type `exit` your **container** will exit and is no longer active. To check that, try the following:

```
$ docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
↳ STATUS	PORTS	NAMES	
de4bbc3eeaec	alpine	"/bin/sh"	3 minutes ago
↳ Exited (0) About a minute ago		pensive_leavitt	

If you want to keep the container active, then you can use keys `Ctrl-p`, `Ctrl-q`. To make sure that it is not exited run the same `docker ps -a` command again:

```
$ docker ps -l
```

CONTAINER ID	IMAGE	PORTS	COMMAND	NAMES	CREATED
↪ STATUS					
0db38ea51a48	alpine		"sh"		3 minutes ago
↪ Up 3 minutes				elastic_lewin	

Now if you want to get back into that container, then you can type `docker attach <container id>`. This way you can save your container:

```
$ docker attach 0db38ea51a48
```

7.4 4. Managing data in Docker

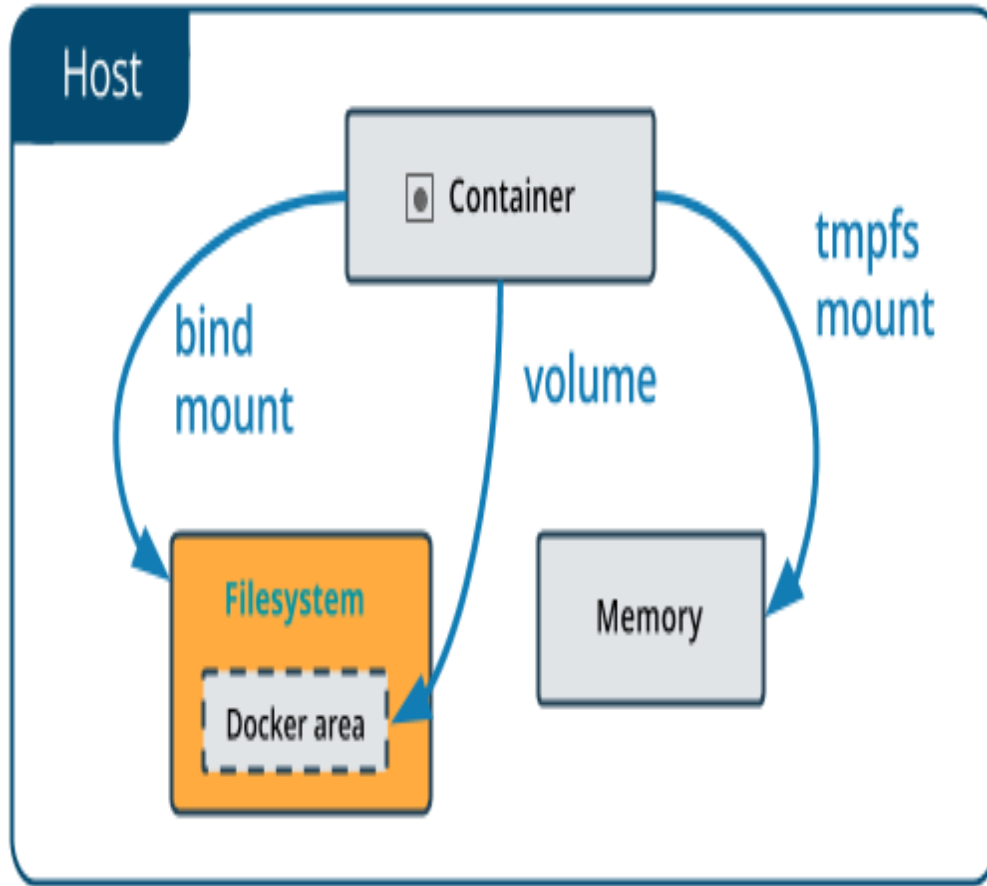
From the above examples, we learned that a running Docker container is an isolated environment created from an Docker image. This means, although it is possible to store data within the “writable layer” of a container, there are some limitations:

- The data doesn’t persist when that container is no longer running, and it can be difficult to get the data out of the container if another process needs it.
- A container’s writable layer is tightly coupled to the host machine where the container is running. You can’t easily move the data somewhere else.

Docker offers three different ways to mount data into a container from the Docker host: **volumes**, **bind mounts**, or **tmpfs volumes**. For simplicity, we will only discuss bind mounts here, even though volumes is the more powerful and usable option for most use cases.

7.4.1 4.1 Bind mounts

Bind mounts: When you use a bind mount, a file or directory on the host machine is mounted into a container.



Warning: One side effect of using bind mounts, for better or for worse, is that you can change the host filesystem via processes running in a container, including creating, modifying, or deleting important system files or directories. This is a powerful ability which can have security implications, including impacting non-Docker processes on the host system.

If you use `--mount` to bind-mount a file or directory that does not yet exist on the Docker host, Docker does not automatically create it for you, but generates an error.

4.1.1 Start a container with a bind mount

Let's clone a git repository to obtain our data sets:

```
$ git clone git@github.com: AstroContainers/2018-05-examples.git
```

We can then `cd` into the HOPS work directory, and mount it to `/root` as we launch the `eventhorizontelescope/hops` container:

```
$ cd 2018-05-examples/hops
$ ls
1234
$ docker run -it --rm --name hops -v $PWD:/root eventhorizontelescope/hops
Setup HOPS v3.19 with HOPS_ROOT=/root for x86_64-3.19
```

You will start at the `/root` work directory and the host data 1234 is available in it:

```
$ pwd
/root
$ ls
1234
```

You can open another terminal and use `docker inspect hops | grep -A9 Mounts` to verify that the bind mount was created correctly. Look for the “Mounts” section

```
$ docker inspect hops | grep -A9 Mounts
"Mounts": [
  {
    "Type": "bind",
    "Source": "/Users/ckchan/2018-05-examples/hops",
    "Destination": "/root",
    "Mode": "",
    "RW": true,
    "Propagation": "rprivate"
  }
],
```

This shows that the mount is a bind mount, it shows the correct source and target, it shows that the mount is read-write, and that the propagation is set to rprivate.

7.4.2 Use case 1: Processing VLBI data with HOPS in Docker

HOPS stands for the Haystack Observatory Postprocessing System. It is a standard tool used in Very-long-baseline interferometry (VLBI) to perform data analysis. HOPS has a long history and it depends on legacy libraries. This makes it difficult to compile HOPS on modern Unix/Linux systems. Nevertheless, the Docker, you **have** already launched a HOPS environment that you can analysis VLBI data!

The most basic step in analysis VLBI is called “fringe fitting”, which we will perform in the running HOPS container by

```
$ ls 1234/No0055/
3C279.zxxerd L..zxxerd LL..zxxerd LW..zxxerd W..zxxerd WW..zxxerd
$ fourfit 1234
fourfit: Warning: No valid data for this pass for pol 2
fourfit: Warning: No valid data for this pass for pol 3
$ ls 1234/No0055/
3C279.zxxerd LL..zxxerd LL.B.2.zxxerd LW.B.3.zxxerd W..zxxerd WW.B.5.zxxerd
L..zxxerd LL.B.1.zxxerd LW..zxxerd LW.B.4.zxxerd WW..zxxerd
```

`fourfit` reads in the correlated data and create the so called “fringe files”. The warnings are normal because there are missing polarizations in the data. In order to see the result of the fringe fitting, you can use `fplot`:

```
$ fplot -d %04d.ps 1234
$ ls
0000.ps 0001.ps 0002.ps 0003.ps 0004.ps 1234
```

You just created 4 fringe plots which contain all important information of the VLBI experiment! Now you can exit your HOPS container and open them on your host machine.

7.5 5. Exposing container ports

Mounting a host directory is one way to make a container connect with the outside work. Another possible is through network by exposing a port.

7.5.1 Use case 2: Processing Galaxy Simulation with Jupyter in Docker

In this second use case, we will use Docker to run a “ready to go” Jupyter notebook in a container. We will expose the port 8888 from the container to the localhost so that you can connect to the notebook.

Inside the 2018-05-examples git repository that you downloaded earlier, there is a sample Galaxy simulation:

```
$ pwd
/Users/ckchan/2018-05-examples/hops
$ cd ../galaxy/
$ pwd
/Users/ckchan/2018-05-examples/galaxy

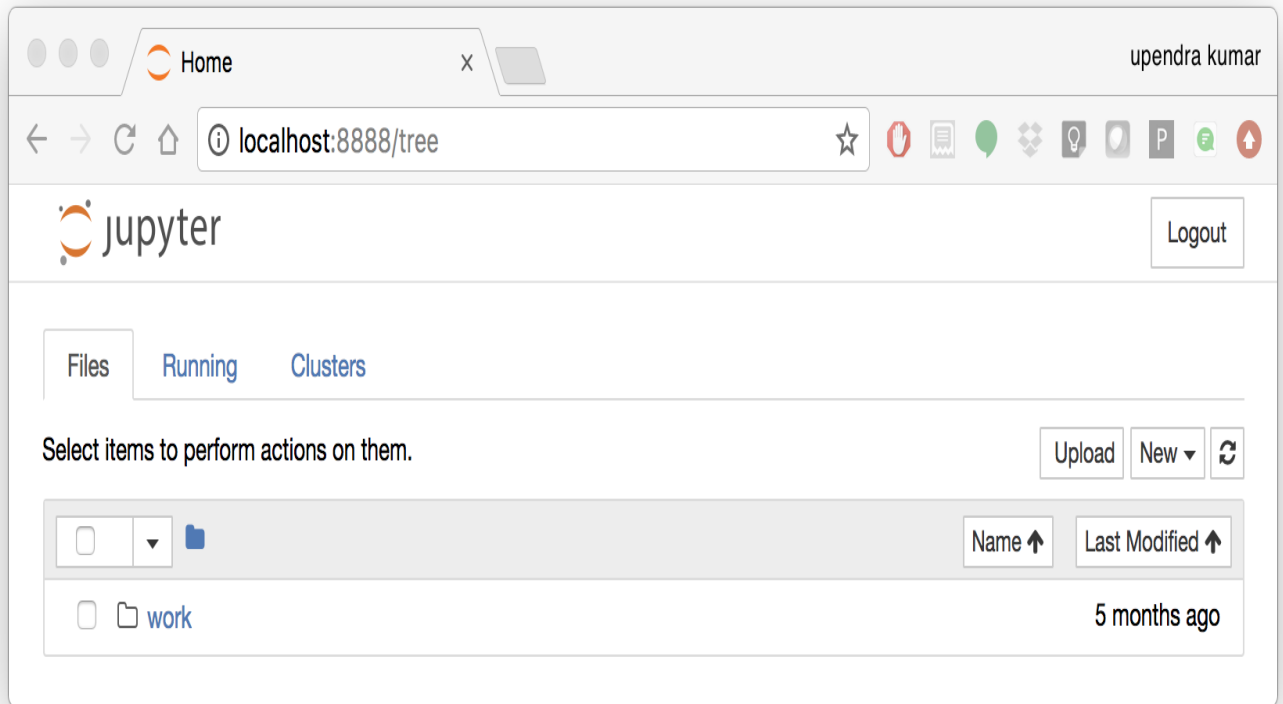
# Specify the uid of the jovyan user. Useful to mount host volumes with specific_
↪file ownership. For this option to take effect, you must run the container with --
↪user root

$ docker run -it --rm -v $PWD:/home/jovyan/work -p 8888:8888 -e NB_UID=$(id -u) --
↪user root astrocontainers/jupyter
Set username to: jovyan
usermod: no changes
Set jovyan UID to: 1329
Executing the command: jupyter notebook
[I 23:36:09.446 NotebookApp] Writing notebook server cookie secret to /home/
↪jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[W 23:36:09.686 NotebookApp] WARNING: The notebook server is listening on all IP_
↪addresses and not using encryption. This is not recommended.
[I 23:36:09.722 NotebookApp] JupyterLab beta preview extension loaded from /opt/
↪conda/lib/python3.6/site-packages/jupyterlab
[I 23:36:09.722 NotebookApp] JupyterLab application directory is /opt/conda/share/
↪jupyterlab
[I 23:36:09.730 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 23:36:09.730 NotebookApp] 0 active kernels
[I 23:36:09.730 NotebookApp] The Jupyter Notebook is running at:
[I 23:36:09.730 NotebookApp] http://[all ip addresses on your system]:8888/?
↪token=a81dbeec92b286df393bb484fdf53efffab410fd64ec8702
[I 23:36:09.730 NotebookApp] Use Control-C to stop this server and shut down all_
↪kernels (twice to skip confirmation).
[C 23:36:09.731 NotebookApp]
Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=dfb50de6c1da091fd62336ac52cdb88de5fe339eb0faf478
```

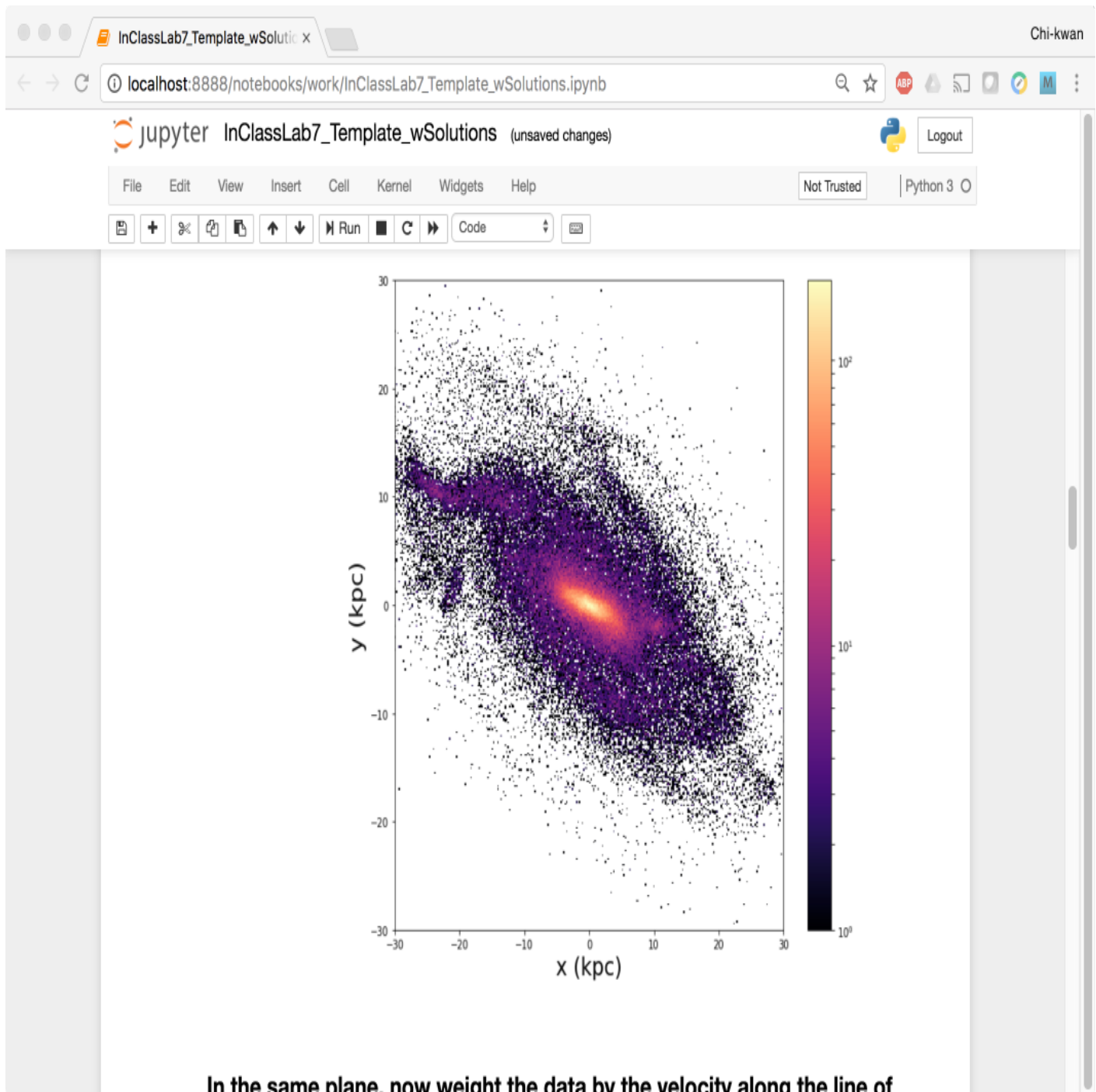
The last line is a URL that we need to copy and paste into our browser to access our new Jupyter Notebook:

```
http://localhost:8888/?token=dfb50de6c1da091fd62336ac52cdb88de5fe339eb0faf478
```

Warning: Do not copy and paste the above URL in your browser as this URL is specific to my environment and it doesn't work for you.



Once you've done that you should be greeted by your very own containerised Jupyter service! Now open `work/InClassLab7_Template_wSolutions.ipynb` and try analysis a Galaxy simulation!



To shut down the container once you're done working, simply hit `Ctrl-C` in the terminal/command prompt. Your work will all be saved on your actual machine in the path we set in our Docker compose file. And there you have it—a quick and easy way to start using Jupyter notebooks with the magic of Docker.

Advanced Docker

Now that we are relatively comfortable with Docker basics, let's look at some of the advanced Docker topics such as building your own Docker image (manual and automatic) using Dockerfile, porting the Docker image to repositories (public and private), and finally deploy containers into cloud and other infrastructures, etc.

8.1 1. Building Docker images

As our HOPS example showed in the last session, one area where Docker shines is when you need to use a command line utility that has a large number of dependencies.

In this session, let's dive deeper into what Docker images are. Later on we will build our own image and use that image to run an application locally.

8.1.1 1.1 Docker images

Docker images are the basis of containers. In the previous example, you pulled the `hello-world` image from the registry and asked the Docker client to run a container based on that image. To see the list of images that are available locally on your system, run the `docker images` command.

```
$ docker images
REPOSITORY      TAG           IMAGE ID      CREATED        SIZE
astroml         latest       8054c898a213  27 seconds ago 934MB
debian          stretch     8626492fec3   7 days ago    101MB
hello-world     latest      690ed74de00f  5 months ago  960 B
alpine          latest      3fd9065eaf02  3 months ago  4.15MB
.....
```

Above is a list of images that I've pulled from the registry and those I've created myself (we'll shortly see how). You will have a different list of images on your machine. The **TAG** refers to a particular snapshot of the image and the **ID** is the corresponding unique identifier for that image.

For simplicity, you can think of an image akin to a git repository - images can be committed with changes and have multiple versions. When you do not provide a specific version number, the client defaults to latest.

For example you could pull a specific version of ubuntu image as follows:

```
$ docker pull ubuntu:16.04
```

If you do not specify the version number of the image, as mentioned, the Docker client will default to a version named latest.

So for example, the `docker pull` command given below will pull an image named `ubuntu:latest`

```
$ docker pull ubuntu
```

To get a new Docker image you can either get it from a registry (such as the Docker hub) or create your own. There are hundreds of thousands of images available on Docker hub. You can also search for images directly from the command line using `docker search`.

```
$ docker search ubuntu
NAME                                STARS          OFFICIAL    DESCRIPTION                                     AUTOMATED
↳ ubuntu                            7310           [OK]        Ubuntu is a Debian-based_
↳ Linux operating sys...            163           [OK]        Ubuntu with openssh-server_
↳ dorowu/ubuntu-desktop-lxde-vnc    131            [OK]        Dockerized SSH service,_
↳ and NoVNC                          90            [OK]        Ubuntu 14.04 LTS with_
↳ rastasheep/ubuntu-sshd             81            [OK]        Upstart is an event-based_
↳ built on top of offi...            43            [OK]        NeuroDebian provides_
↳ ansible/ansible                    35            [OK]        debootstrap --
↳ ubuntu-upstart                     26            [OK]        ubuntu-16-nginx-php-
↳ replacement for th...              22            [OK]        Simple always updated Ubuntu_
↳ neurodebian                       11            [OK]        Simple Ubuntu docker images_
↳ neuroscience research s...         9            [OK]        Ubuntu is a Debian-based_
↳ ubuntu-debootstrap                 7            [OK]        Ubuntu is a Debian-based_
↳ variant=minbase --components=m... 5            [OK]        ubuntu-16-nginx-php-5.6-wordpress-4
↳ landlinternet/ubuntu-16-nginx-php-phpmyadmin-mysql-5 2
↳ phpmyadmin-mysql-5                1            [OK]        ubuntu-16-nginx
↳ nuagebec/ubuntu                   0            [OK]        A quick freshening-up of the_
↳ docker images w...                 0            [OK]        ubuntu with smartentry_
↳ tutum/ubuntu                      0            [OK]
↳ with SSH access                   0            [OK]
↳ ppc64le/ubuntu                    0            [OK]
↳ Linux operating sys...             0            [OK]
↳ i386/ubuntu                       0            [OK]
↳ Linux operating sys...             0            [OK]
↳ landlinternet/ubuntu-16-apache-php-7.0 0
↳ eclipse/ubuntu_jdk8               0            [OK]
↳ curl, nmap, mc, ...               0            [OK]
↳ darksheer/ubuntu                  0            [OK]
↳ hourly                           0            [OK]
↳ codenvy/ubuntu_jdk8               0            [OK]
↳ curl, nmap, mc, ...               0            [OK]
↳ landlinternet/ubuntu-16-nginx-php-5.6-wordpress-4 0
↳ wordpress-4                       0            [OK]
↳ landlinternet/ubuntu-16-nginx     0            [OK]
↳ pivotaldata/ubuntu                0            [OK]
↳ base Ubuntu doc...                0            [OK]
↳ smartentry/ubuntu                 0            [OK]
```

(continues on next page)

(continued from previous page)

pivotaldata/ubuntu-gpdb-dev	Ubuntu images for GPDB
↪development 0	
landlinternet/ubuntu-16-healthcheck	ubuntu-16-healthcheck
↪0	[OK]
thatsamguy/ubuntu-build-image	Docker webapp build images
↪based on Ubuntu 0	
ossobv/ubuntu	Custom ubuntu image from
↪scratch (based on o... 0	
landlinternet/ubuntu-16-sshd	ubuntu-16-sshd
↪0	[OK]

An important distinction with regard to images is between base images and child images and official images and user images (Both of which can be base images or child images.).

Important: **Base images** are images that have no parent images, usually images with an OS like ubuntu, alpine or debian.

Child images are images that build on base images and add additional functionality.

Official images are Docker sanctioned images. Docker, Inc. sponsors a dedicated team that is responsible for reviewing and publishing all Official Repositories content. This team works in collaboration with upstream software maintainers, security experts, and the broader Docker community. These are not prefixed by an organization or user name. In the list of images above, the python, node, alpine and nginx images are official (base) images. To find out more about them, check out the Official Images Documentation.

User images are images created and shared by users like you. They build on base images and add additional functionality. Typically these are formatted as `user/image-name`. The user value in the image name is your Dockerhub user or organization name.

8.1.2 1.2 Building custom Docker images

1.2.1 Using docker commit (not recommended)

As we saw in the Docker introduction, the general Docker workflow is:

- start a container based on an image in a known state
- add things to the filesystem, such as packages, codebases, libraries, files, or anything else
- commit the changes as layers to make a new image

Let's follow this workflow to build a custom image. Instead of *alpine* this time we will use *ubuntu* linux image to install some interesting packages

As before first either pull the *ubuntu* docker image or you can just `docker run -it ubuntu` to pull and run the container interactively

```
$ docker run -it ubuntu:16.04
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu
Digest: sha256:9ee3b83bcaa383e5e3b657f042f4034c92cdd50c03f73166c145c9ceaea9ba7c
Status: Downloaded newer image for ubuntu:16.04
root@7f989e4174aa:/#
```

Let's install two packages *fortune*, *cowsay*, *lolcat* inside the container. But before that it's always a good idea to update the packages that are already existing in the ubuntu.

```
root@7f989e4174aa:/# apt-get update
root@7f989e4174aa:/# apt-get install -y fortune cowsay lolcat
```

Now exit the container and run *docker ps -a* to check to see if the status of the container (which is exit in this case)

```
root@7f989e4174aa:/# exit
```

Go ahead and commit the changes and create a new image.

```
docker commit -m "Installed fortune cowsay lolcat" $(docker ps -lq) ubuntu/
↳ fortunecowsaylolcat
sha256:77ae42b823e60c2a350228d892aacda337e1e01c19c3ae72da104f7f4a77f83f
```

Congratulations. You created your first Docker image. Check to see your docker image in the list of images using *docker images*. Let's run a container using that newly created docker image

```
$ docker run ubuntu/fortunecowsaylolcat /usr/games/cowsay "Hi"

  ____
< Hi >
  ----
      \   ^__^
         (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

and another one

```
$ docker run ubuntu/fortunecowsay /usr/games/fortune
It's all in the mind, ya know.
```

Pretty cool isn't it..

Exercise: Can you figure out a way to combine these two commands in this order *fortune*, *cowsay* and *lolcat* to print what cowsay of the fortune output?

Hint: Use pipe and use interactive terminal

1.2.2 Using Dockerfile (recommended)

As you noticed by now that this method of making images is not reproducible. For example if you share this image with someone (we will see how it is done later), then they wouldn't know what is installed in this image. Ofcourse you can provide them with your notes but still it's not reproducible. Rather than just running commands and installing commands using *apt-get install*, we'll put our instructions in a special file called the Dockerfile

What exactly is a Dockerfile?

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image. Using *docker build* users can create an automated build that executes several command-line instructions in succession. Let's create a Dockerfile for the above image

Open up a text editor of your choice and type in the following commands and save it as *Dockerfile*

Tip: You can name your Dockerfile as anything but according to best practices it is recommended to name it as *Dockerfile* for reasons we will see later

```
FROM ubuntu:16.04
MAINTAINER Upendra Devisetty <upendra@cyverse.org>
LABEL version="1.0" description="This Dockerfile is for building fortune cowsay_
↳lolcat ubuntu image"
RUN apt-get update
RUN apt-get install -y fortune cowsay lolcat

ENV PATH=/usr/games/:$PATH
CMD fortune | cowsay | lolcat
```

That's it. Now building the Docker image using *docker build* command as below. The *docker build* command is quite simple - it takes an optional tag name with the *-t* flag, and the location of the directory containing the Dockerfile - the *.* indicates the current directory:

```
docker build -t ubuntu/fortunecowsaylolcat2 .
Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM ubuntu:16.04
---> c9d990395902
Step 2/5 : MAINTAINER Upendra Devisetty <upendra@cyverse.org>
---> Running in a365c28eb283
Removing intermediate container a365c28eb283
---> 91d18ff89d44
Step 3/5 : LABEL Description "This Dockerfile is for building fortune coway ubuntu_
↳image"
---> Running in d24ff4a347fa
Removing intermediate container d24ff4a347fa
---> 73daa1277fea
Step 4/5 : RUN apt-get update
---> Running in eed1e2fe25de
.....
.....
Successfully built ffe89a681d5c
Successfully tagged ubuntu/fortunecowsaylolcat2:latest
```

Great! We successfully built a Docker image using Dockerfile. Let's test it out by launching a container using *docker run*.

```
$ docker run --rm ubuntu/fortunecowsaylolcat2:1.0
```

```
 / It was all so different before \
 \ everything changed.             /
-----
      \      ^__^
       (oo)\_____)
          (_____) )  \\\
              ||----w |
              ||     ||
```

Superb! So you have build a Docker image using Dockerfile. See how easy it is and it is also reproducible since you know how it is built. In addition, you can version control (using git or others) this Dockerfile.

Before we go further, let's look at what those commands in Dockerfile mean

FROM

This instruction is used to set the base image for subsequent instructions. It is mandatory to set this in the first line of a Dockerfile. You can use it any number of times though.

MAINTAINER

This is a non-executable instruction used to indicate the author of the Dockerfile.

LABEL

You can assign metadata in the form of key-value pairs to the image using this instruction. It is important to notice that each LABEL instruction creates a new layer in the image, so it is best to use as few LABEL instructions as possible

RUN

This instruction lets you execute a command on top of an existing layer and create a new layer with the results of command execution

CMD

This defines the commands that will run on the Image at start-up. Unlike a **RUN**, this does not create a new layer for the Image, but simply runs the command. There can only be one CMD per a Dockerfile/Image. If you need to run multiple commands, the best way to do that is to have the CMD run a script. CMD requires that you tell it where to run the command, unlike RUN.

ENV

This defines Environmental variables (one or more) in the Docker image

WORKDIR

The WORKDIR directive is used to set where the command defined with CMD is to be executed.

ENTRYPOINT

This argument sets the concrete default application that is used every time a container is created using the image. For example, if you have installed a specific application inside an image and you will use this image to only run that application, you can state it with ENTRYPOINT and whenever a container is created from that image, your application will be the target

Use case 1: Building a astroML Docker image

This is a minimal Docker image using *astroML* as an example. `plot_spectrum_sum_of_norms.py` is an example script from astroML. It is modified to run better in a container environment.

```
$ mkdir astroML && cd astroML

$ wget https://de.cyverse.org/dl/d/2CE93196-0F91-414F-B532-0AC8D3AE032E/plot_spectrum_
↪sum_of_norms.py

$ vi Dockerfile
FROM debian:stretch
MAINTAINER Upendra Devisetty <upendra@cyverse.org>
LABEL version="1.0" description="This image is for astroML"

# The base image is minimal with very few software packages. We
# update apt and install python-pip with recommended packages here.
RUN apt-get -qq update &&\
    apt-get install -y python-pip &&\
    apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
# Next, we install astroML's dependencies and astroML itself using pip.
RUN pip install numpy scipy scikit-learn matplotlib astropy &&\
    pip install astroML astroML_addons

# Change the default work directory to "/root" inside the container.
WORKDIR /root

# Install Jupyter and other visualization packages
```

(continues on next page)

(continued from previous page)

```

RUN pip install jupyter ipywidgets &&\
    jupyter nbextension enable --py --sys-prefix widgetsnbextension

COPY plot_spectrum_sum_of_norms.py /usr/bin
COPY run.sh /usr/bin
RUN chmod +x /usr/bin/plot_spectrum_sum_of_norms.py
RUN chmod +x /usr/bin/run.sh
ENTRYPOINT ["run.sh"]

$ vi run.sh
#!/bin/bash
plot_spectrum_sum_of_norms.py $1

```

Let's build the image from the Dockerfile now

```
$ docker build -t debian/astroml:1.0 .
```

Now run the built image to execute first

```
$ docker run --rm -v ${PWD}:/root debian/astroml:1.0 test.pdf
```

The result is the pdf - *test.pdf*

Now remove the the two ouputs and run it with overriding the ENTRYPOINT

```

$ rm -r astroML_data test.pdf

$ docker run --rm -it -p 8888:8888 -v ${PWD}:/root --entrypoint jupyter-notebook_
↳debian/astroml:1.0 --allow-root --ip='*' --no-browser
[W 18:11:00.622 NotebookApp] WARNING: The notebook server is listening on all IP_
↳addresses and not using encryption. This is not recommended.
[I 18:11:00.629 NotebookApp] Serving notebooks from local directory: /root
[I 18:11:00.629 NotebookApp] 0 active kernels
[I 18:11:00.629 NotebookApp] The Jupyter Notebook is running at:
[I 18:11:00.629 NotebookApp] http://[all ip addresses on your system]:8888/?
↳token=a2a9027494420d6151824ad23d930b91a37eeb44597454ac
[I 18:11:00.630 NotebookApp] Use Control-C to stop this server and shut down all_
↳kernels (twice to skip confirmation).
[C 18:11:00.633 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://localhost:8888/?token=a2a9027494420d6151824ad23d930b91a37eeb44597454ac

```

- **Exercise:**

Create a new Jupyter notebook for the *plot_spectrum_sum_of_norms.py* script and execute it to make sure that all the steps are working fine..

8.2 2. Docker registries

To demonstrate the portability of what we just created, let's upload our built Docker image and run it somewhere else (CyVerse Atmosphere cloud or Discovery Environment). After all, you'll need to learn how to push to registries when you want to deploy containers to production.

Important: So what exactly is a registry?

A registry is a collection of repositories, and a repository is a collection of images—sort of like a GitHub repository, except the code is already built. An account on a registry can create many repositories. The docker CLI uses Docker’s public registry by default. You can even set up your own private registry using Docker Trusted Registry

There are several things you can do with Docker registries:

- Pushing images
- Finding images
- Pulling images
- Sharing images

8.2.1 2.1 Public repositories

Some example of public registries include [Docker cloud](#), [Docker hub](#) and [quay.io](#) etc.,

2.1.1 Log in with your Docker ID

Now that you’ve created and tested your image, you can push it to Docker cloud or Docker hub.

Note: If you don’t have a Docker account, sign up for one at [Docker cloud](#) or [Docker hub](#). Make note of your username. There are several advantages of registering to Dockerhub which we will see later on in the session

First you have to login to your Docker hub account. To do that:

```
$ docker login -u <dockerhub username>
Password:
```

Enter you Password when prompted.

2.1.2 Tag the image

The notation for associating a local image with a repository on a registry is `username/repository:tag`. The tag is optional, but recommended, since it is the mechanism that registries use to give Docker images a version. Give the repository and tag meaningful names for the context, such as `get-started:part2`. This will put the image in the `get-started` repository and tag it as `part2`.

Note: By default the docker image gets a `latest` tag if you don’t provide one. Thought convenient, it is not recommended for reproducibility purposes.

Now, put it all together to tag the image. Run `docker tag image` with your username, repository, and tag names so that the image will upload to your desired destination. For our docker image since we already have our Dockerhub username we will just add tag which in this case is `1.0`

```
$ docker tag debian/astroml:1.0 <dockerhub username>/astroml:1.0
```

2.1.3 Publish the image

Upload your tagged image to the Dockerhub repository

```
$ docker push <dockerhub username>/astroml:1.0
```

Once complete, the results of this upload are publicly available. If you log in to Docker Hub, you will see the new image there, with its pull command.



Congrats! You just made your first Docker image and shared it with the world!

2.1.4 Pull and run the image from the remote repository

Let's try to run the image from the remote repository on Cloud server by logging into CyVerse Atmosphere, [launching an instance](#)

First install Docker on Atmosphere using from here <https://docs.docker.com/install/linux/docker-ce/ubuntu> or alternatively you can use `ezd` command which is a short-cut command for installing Docker on Atmosphere

```
$ ezd
```

Now run the following command to run the docker image from Dockerhub

```
$ docker run --rm -v ${PWD}:/root upendradevisetty/astroml:1.0
```

Note: You don't have to run `docker pull` since if the image isn't available locally on the machine, Docker will pull it from the repository.

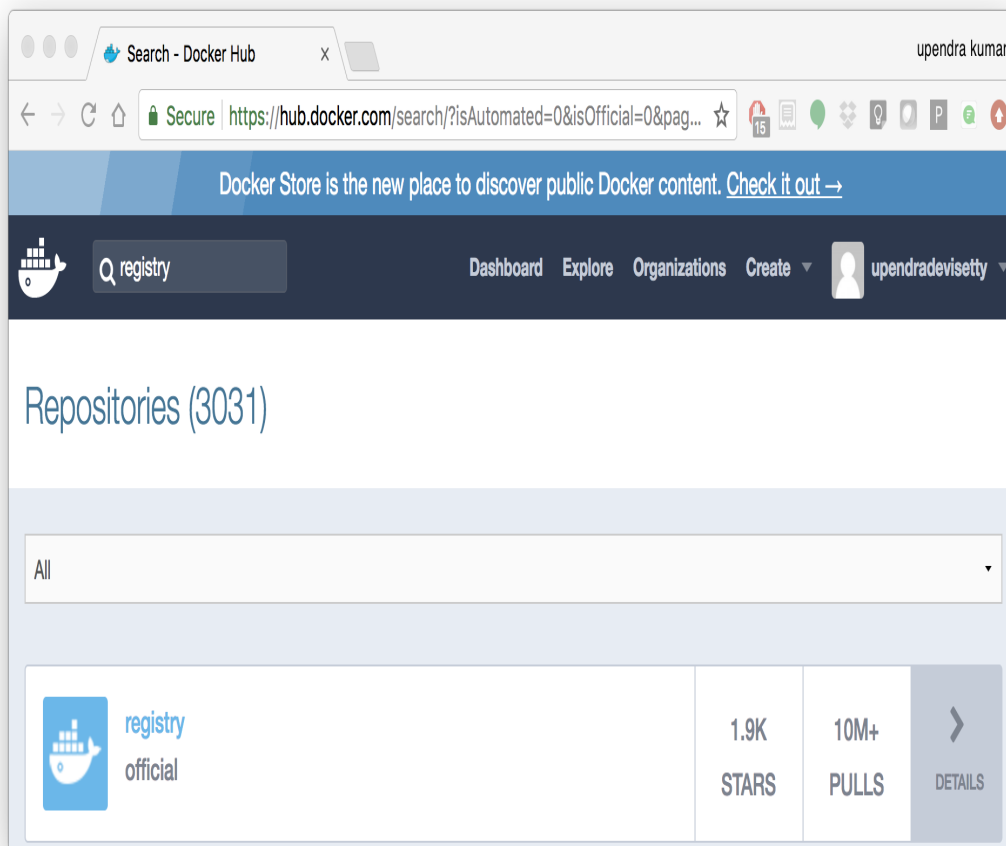
8.2.2 Private repositories

In an earlier part, we had looked at the Docker Hub, which is a public registry that is hosted by Docker. While the Dockerhub plays an important role in giving public visibility to your Docker images and for you to utilize quality Docker images put up by others, there is a clear need to setup your own private registry too for your team/organization. For example, CyVerse has its own private registry which will be used to push the Docker images.

2.2.1 Pull down the Registry Image

You might have guessed by now that the registry must be available as a Docker image from the Docker Hub and it should be as simple as pulling the image down and running that. You are correct!

A Dockerhub search on the keyword `registry` brings up the following image as the top result:



Run a container from registry Dockerhub image

```
$ docker run -d -p 5000:5000 --name registry registry:2
```

Run `docker ps -l` to check the recent container from this Docker image

```
$ docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
↪STATUS	PORTS	NAMES	
6e44a0459373	registry:2	"/entrypoint.sh /e..."	11 seconds ago
↪Up 10 seconds	0.0.0.0:5000->5000/tcp	registry	

2.2.2 Tag the image that you want to push

Next step is to tag your image under the registry namespace and push it there

```
$ REGISTRY=localhost:5000

$ docker tag upendradevisetty/astroml:1.0 $REGISTRY/${whoami}/astroml:1.0
```

2.2.2 Publish the image into the local registry

Finally push the image to the local registry

```
$ docker push $REGISTRY/${whoami}/astroml:1.0
The push refers to repository [localhost:5000/upendra_35/astroml]
dbcl33154d04: Pushed
2efc0f8eb69d: Pushed
2ec163aac8ff: Pushed
0f3a12fef684: Pushed
1.0: digest: sha256:eba0beb8f735a8d32b74bed0e0194c0b04e6d15608c21736f819ca8ee06f83c5
↪size: 1167
```

2.2.3 Pull and run the image from the local repository

You can also pull the image from the local repository similar to how you pull it from Dockerhub and run a container from it

```
$ docker run --rm -p 8888:8888 -v ${PWD}:/root $REGISTRY/${whoami}/astroml:1.0
[I 17:06:53.813 NotebookApp] Writing notebook server cookie secret to /root/.local/
↪share/jupyter/runtime/notebook_cookie_secret
[W 17:06:54.094 NotebookApp] WARNING: The notebook server is listening on all IP
↪addresses and not using encryption. This is not recommended.
[I 17:06:54.106 NotebookApp] Serving notebooks from local directory: /root
[I 17:06:54.106 NotebookApp] 0 active kernels
[I 17:06:54.106 NotebookApp] The Jupyter Notebook is running at:
[I 17:06:54.106 NotebookApp] http://[all ip addresses on your system]:8888/?
↪token=b0ff6191b65f65a7bdc185597b4168e4f9755d06363dde62
[I 17:06:54.107 NotebookApp] Use Control-C to stop this server and shut down all
↪kernels (twice to skip confirmation).
[C 17:06:54.110 NotebookApp]
```

Copy/paste this URL into your browser when you connect **for** the first time,

(continues on next page)

(continued from previous page)

```
to login with a token:  
http://localhost:8888/?token=b0ff6191b65f65a7bdc185597b4168e4f9755d06363dde62
```

8.3 3. Automated Docker image building from github

An automated build is a Docker image build that is triggered by a code change in a GitHub or Bitbucket repository. By linking a remote code repository to a Dockerhub automated build repository, you can build a new Docker image every time a code change is pushed to your code repository.

A build context is a Dockerfile and any files at a specific location. For an automated build, the build context is a repository containing a Dockerfile.

Automated Builds have several advantages:

- Images built in this way are built exactly as specified.
- The Dockerfile is available to anyone with access to your Docker Hub repository.
- Your repository is kept up-to-date with code changes automatically.
- Automated Builds are supported for both public and private repositories on both GitHub and Bitbucket.

8.3.1 3.1 Prerequisites

To use automated builds, you first must have an account on [Docker Hub](#) and on the hosted repository provider ([GitHub](#) or [Bitbucket](#)). While Dockerhub supports linking both GitHub and Bitbucket repositories, here we will use a GitHub repository. If you don't already have one, make sure you have a GitHub account. A basic account is free

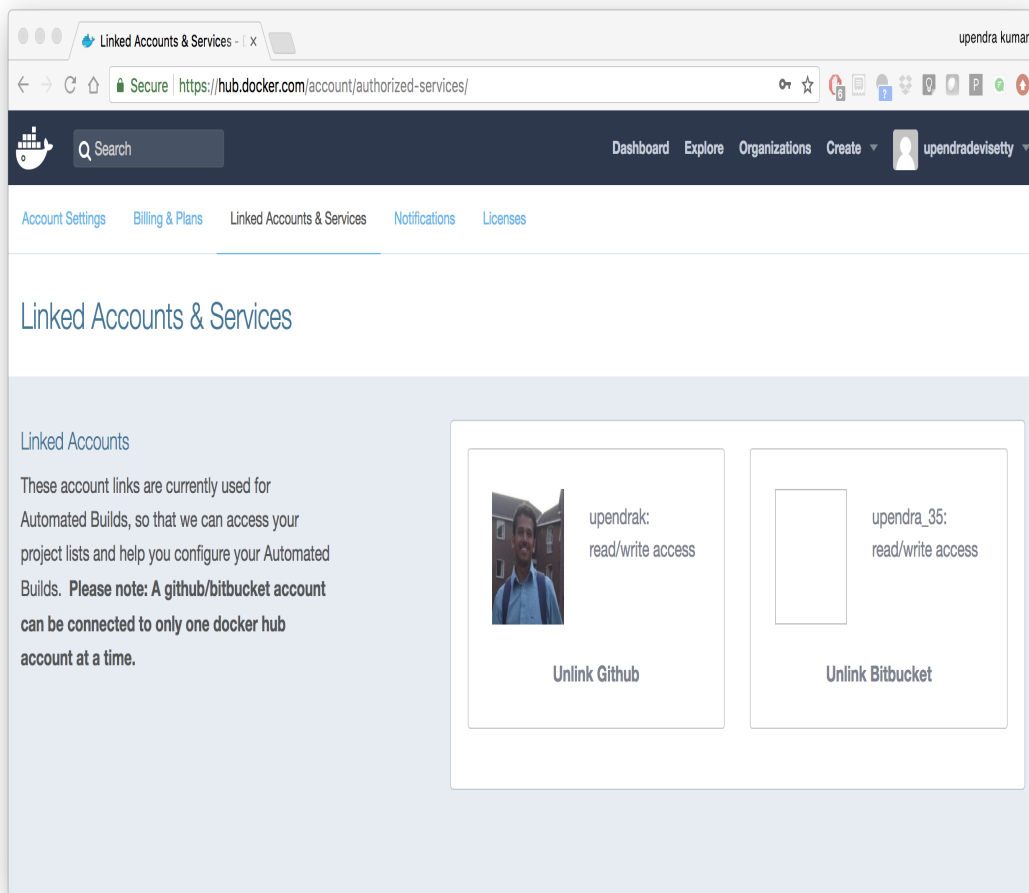
Note:

- If you have previously linked your Github or Bitbucket account, you must have chosen the Public and Private connection type. To view your current connection settings, log in to Docker Hub and choose Profile > Settings > Linked Accounts & Services.
 - Building Windows containers is not supported.
-

8.3.2 3.2 Link your Docker Hub account to GitHub

1. Log into Docker Hub.
2. Navigate to [Profile > Settings > Linked Accounts & Services](#).
3. Click the [Link GitHub](#). The system prompts you to choose between **Public and Private** and **Limited Access**. The **Public** and **Private** connection type is required if you want to use the Automated Builds.
4. Press [Select](#) under **Public and Private** connection type. If you are not logged into GitHub, the system prompts you to enter GitHub credentials before prompting you to grant access. After you grant access to your code repository, the system returns you to Docker Hub and the link is complete.

After you grant access to your code repository, the system returns you to Docker Hub and the link is complete. For example, github linked hosted repository looks like this:



8.3.3 3.3 Create a new automated build

Automated build repositories rely on the integration with your github code repository to build.

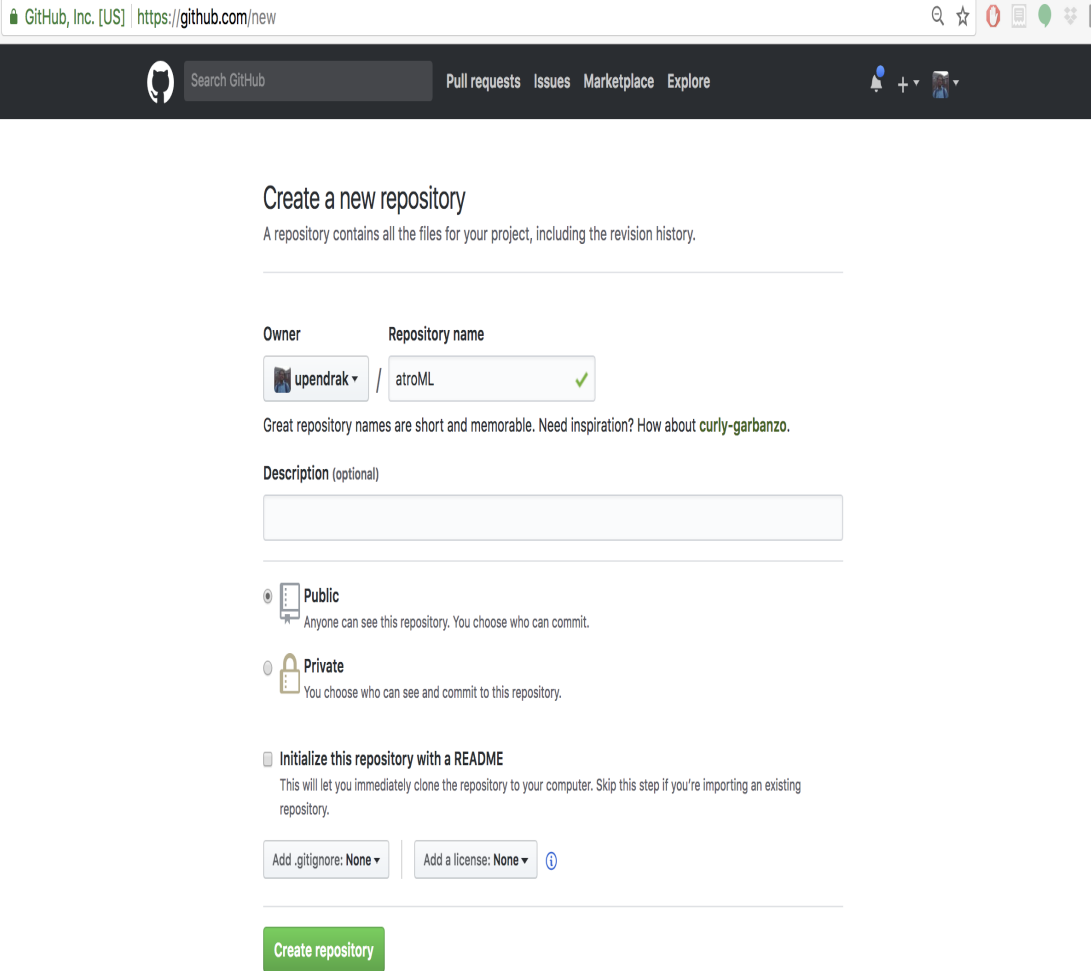
Let's create an automatic build for our `astroML` using the instructions below:

1. Initialize git repository for the `flask-app` directory

```
$ git init
Initialized empty Git repository in /Users/upendra_35/Downloads/docker_workshop/
↪astroML/.git/

$ git add Dockerfile run.sh plot_spectrum_sum_of_norms.py && git commit -m"Add files,
↪and folders"
[master (root-commit) 3d85ec9] Add files and folders
 3 files changed, 83 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 plot_spectrum_sum_of_norms.py
 create mode 100644 run.sh
```

2. Create a new repository on github by navigating to this url - <https://github.com/new>




GitHub, Inc. [US] | <https://github.com/new>

Search GitHub Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

 upendrak / atroML ✓

Great repository names are short and memorable. Need inspiration? How about curly-garbanzo.

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** Add a license: **None** ⓘ

Create repository

3. Push the repository to github

...or push an existing repository from the command line

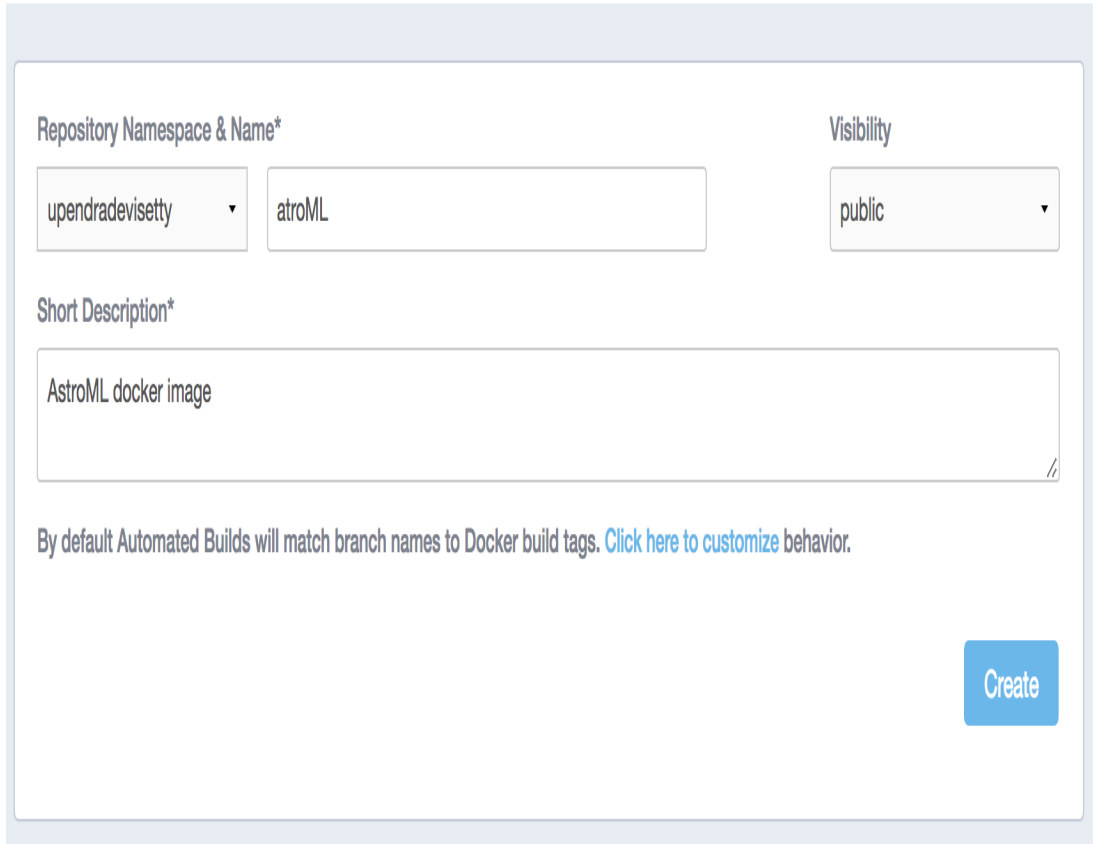
```
git remote add origin https://github.com/upendrak/atroML.git  
git push -u origin master
```

```
$ git remote add origin https://github.com/upendrak/atroML.git  
$ git push -u origin master
```

4. Select Create > Create Automated Build from Docker Hub.

- The system prompts you with a list of User/Organizations and code repositories.
- For now select your GitHub account from the User/Organizations list on the left. The list of repositories change.
- Pick the project to build. In this case `atroML`. Type in “AstroML docker image” in the Short Description box.
- If you have a long list of repos, use the filter box above the list to restrict the list. After you select the project, the system displays the Create Automated Build dialog.

Create Automated Build



The screenshot shows a web form titled "Create Automated Build". The form is contained within a light blue border. It has the following fields and elements:

- Repository Namespace & Name***: This section contains two input fields. The first is a dropdown menu with "upendradevisetty" selected and a downward arrow. The second is a text input field containing "atroML".
- Visibility**: A dropdown menu with "public" selected and a downward arrow.
- Short Description***: A large text area containing the text "AstroML docker image". A small "//" icon is visible at the bottom right of the text area.
- Help Text**: Below the text area, there is a line of text: "By default Automated Builds will match branch names to Docker build tags. [Click here to customize behavior.](#)"
- Create Button**: A blue button with the text "Create" in white, located at the bottom right of the form.

Note: The dialog assumes some defaults which you can customize. By default, Docker builds images for each branch in your repository. It assumes the Dockerfile lives at the root of your source. When it builds an image, Docker tags it with the branch name.

5. Customize the automated build by pressing the `Click here to customize behavior` link.

By default Automated Builds will match branch names to Docker build tags. [Click here to customize behavior.](#)

Customize Autobuild Tags

Your image will build automatically when your source repository is pushed based on the following rules. [Revert to default settings](#)

Push Type	Name	Dockerfile Location	Docker Tag	
Tag	1.0	/	1.0	+
Branch	All branches except master	/	Same as branch	-

Create

Specify which code branches or tags to build from. You can build by a code branch or by an image tag. You can enter a specific value or use a regex to select multiple values. To see examples of regex, press the Show More link on the right of the page.

- Leave Push Type as Branch as is.
- Leave the Dockerfile location as is.
- Recall the file is in the root of your code repository.
- Specify 1.0 for the Tag Name.

6. Click **Create**.

Important: During the build process, Docker copies the contents of your Dockerfile to Docker Hub. The Docker community (for public repositories) or approved team members/orgs (for private repositories) can then view the Dockerfile on your repository page.

The build process looks for a README.md in the same directory as your Dockerfile. If you have a README.md file in your repository, it is used in the repository as the full description. If you change the full description after a build, it's overwritten the next time the Automated Build runs. To make changes, modify the README.md in your Git repository.

Warning: You can only trigger one build at a time and no more than one every five minutes. If you already have a build pending, or if you recently submitted a build request, Docker ignores new requests.

It can take a few minutes for your automated build job to be created. When the system is finished, it places you in the detail page for your Automated Build repository.

7. Manually Trigger a Build

Before you trigger an automated build by pushing to your GitHub `astroML` repo, you'll trigger a manual build. Triggering a manual build ensures everything is working correctly.

From your automated build page choose `Build Settings`

Press `Trigger` button and finally click `Save Changes`.

Note: Docker builds everything listed whenever a push is made to the code repository. If you specify a particular branch or tag, you can manually build that image by pressing the `Trigger`. If you use a regular expression syntax (regex) to define your build branch or tag, Docker does not give you the option to manually build.

PUBLIC | AUTOMATED BUILD

upendradevisetty/atroML ☆


Last pushed: never

Repo Info Tags Dockerfile Build Details Build Settings Collaborators Webhooks Settings Try in PWD

Build Settings

☒ When active, builds will happen automatically on pushes.

The build rules below specify how to build your source into Docker images. The name can be a string or a regex. The Docker Tag name may contain variables. We currently support `{sourcerefs}`, which refers to the source branch/tag name. [Show more](#)



Source Repository
upendrak/atroML

Type	Name	Dockerfile Location	Docker Tag Name		
Branch ▾	master	/	1.0	+	Trigger
Branch ▾	All branches except master	/	Same as branch	-	

Save Changes

8. Review the build results

The Build Details page shows a log of your build systems:

Navigate to the `Build Details` page.

Wait until your image build is done.

You may have to manually refresh the page and your build may take several minutes to complete.

PUBLIC | AUTOMATED BUILD

upendradevisetty/atroml ☆

Last pushed: 6 minutes ago

[Repo Info](#) [Tags](#) [Dockerfile](#) [Build Details](#) [Build Settings](#) [Collaborators](#) [Webhooks](#) [Settings](#)



Status	Actions	Tag	Created	Last Updated	Source Repository
✓ Success		1.0	18 minutes ago	6 minutes ago	upendrak/atroml

8.4 4. Docker Compose for multi container apps

Docker Compose is a tool for defining and running your multi-container Docker applications.

Main advantages of Docker compose include:

- Your applications can be defined in a YAML file where all the options that you used in `docker run` are now defined (Reproducibility).
- It allows you to manage your application as a single entity rather than dealing with individual containers (Simplicity).

Let's now create a simple web app with Docker Compose using Flask (which you already seen before) and Redis. We will end up with a Flask container and a Redis container all on one host.

Note: The code for the above compose example is available [here](#)

1. You'll need a directory for your project on your host machine:

```
$ mkdir compose_flask && cd compose_flask
```

2. Add the following to *requirements.txt* inside *compose_flask* directory:

```
flask
redis
```

3. Copy and paste the following code into a new file called *app.py* inside *compose_flask* directory:

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
    return 'This Compose/Flask demo has been viewed %s time(s).' % redis.get('hits')

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

4. Create a Dockerfile with the following code inside *compose_flask* directory:

```
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py
```

5. Add the following code to a new file, *docker-compose.yml*, in your project directory:

```
version: '2'
services:
  web:
    restart: always
    build: .
    ports:
      - "8888:5000"
    volumes:
      - ./code
    depends_on:
      - redis
  redis:
    restart: always
    image: redis
```

A brief explanation of *docker-compose.yml* is as below:

- `restart: always` means that it will restart whenever it fails.
- We define two services, **web** and **redis**.
- The web service builds from the Dockerfile in the current directory.
- Forwards the container's exposed port (5000) to port 8888 on the host.

- Mounts the project directory on the host to /code inside the container (allowing you to modify the code without having to rebuild the image).
- depends_on links the web service to the Redis service.
- The redis service uses the latest Redis image from Docker Hub.

Note: Docker for Mac and Docker Toolbox already include Compose along with other Docker apps, so Mac users do not need to install Compose separately. Docker for Windows and Docker Toolbox already include Compose along with other Docker apps, so most Windows users do not need to install Compose separately.

For Linux users

```
sudo curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-
↪compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

5. Build and Run with docker-compose up -d command

```
$ docker-compose up -d

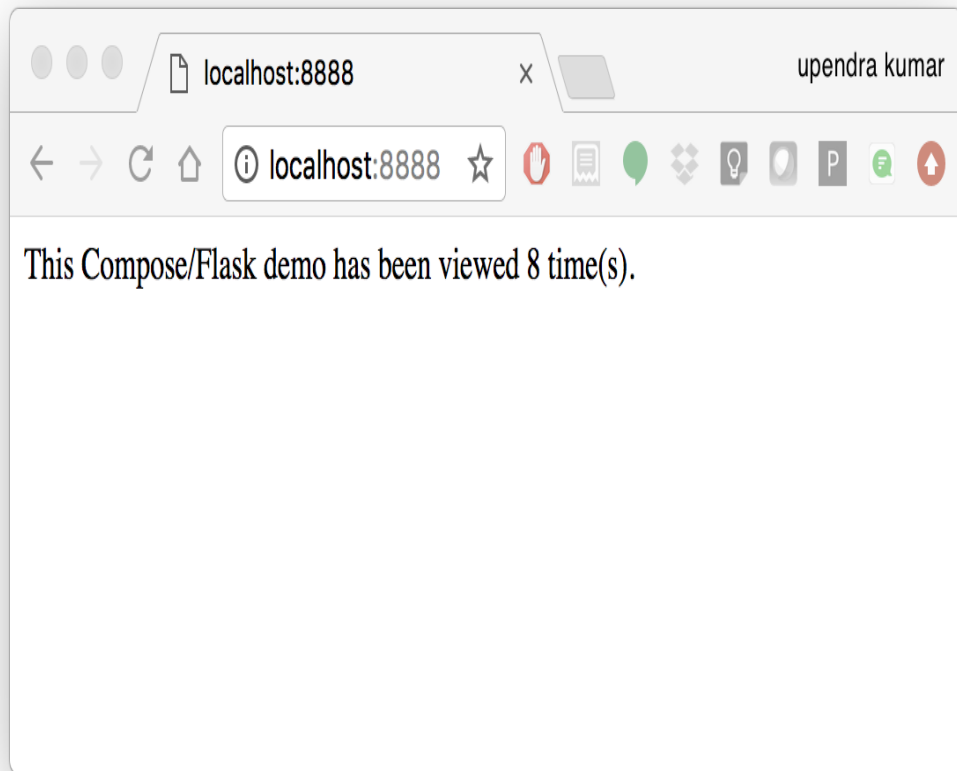
Building web
Step 1/5 : FROM python:2.7
2.7: Pulling from library/python
f49cf87b52c1: Already exists
7b491c575b06: Already exists
b313b08bab3b: Already exists
51d6678c3f0e: Already exists
09f35bd58db2: Already exists
f7e0c30e74c6: Pull complete
c308c099d654: Pull complete
339478b61728: Pull complete
Digest: sha256:8cb593cb9cd1834429f0b4953a25617a8457e2c79b3e111c0f70bffd21acc467
Status: Downloaded newer image for python:2.7
---> 9e92c8430ba0
Step 2/5 : ADD . /code
---> 746bcecf3c9
Step 3/5 : WORKDIR /code
---> c4cf3d6cb147
Removing intermediate container 84d850371a36
Step 4/5 : RUN pip install -r requirements.txt
---> Running in d74c2elcfbf7
Collecting flask (from -r requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting redis (from -r requirements.txt (line 2))
  Downloading redis-2.10.6-py2.py3-none-any.whl (64kB)
Collecting itsdangerous>=0.21 (from flask->-r requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.4 (from flask->-r requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting Werkzeug>=0.7 (from flask->-r requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting click>=2.0 (from flask->-r requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->flask->-r requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
```

(continues on next page)

(continued from previous page)

```
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
  Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/fc/a8/66/
↪24d655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
  Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/88/a7/30/
↪e39a54a87bcbe25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, ↪
↪flask, redis
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 flask-0.
↪12.2 itsdangerous-0.24 redis-2.10.6
---> 5cc574ff32ed
Removing intermediate container d74c2e1c9bf7
Step 5/5 : CMD python app.py
---> Running in 3ddb7040e8be
---> e911b8e8979f
Removing intermediate container 3ddb7040e8be
Successfully built e911b8e8979f
Successfully tagged composeflask_web:latest
```

And that's it! You should be able to see the Flask application running on `http://localhost:8888` or `<ipaddress>:8888`



Additional Docker Demo's

9.1 1. Portainer

Portainer is an open-source lightweight management UI which allows you to easily manage your Docker hosts or Swarm cluster.

- Simple to use: It has never been so easy to manage Docker. Portainer provides a detailed overview of Docker and allows you to manage containers, images, networks and volumes. It is also really easy to deploy, you are just one Docker command away from running Portainer anywhere.
- Made for Docker: Portainer is meant to be plugged on top of the Docker API. It has support for the latest versions of Docker, Docker Swarm and Swarm mode.

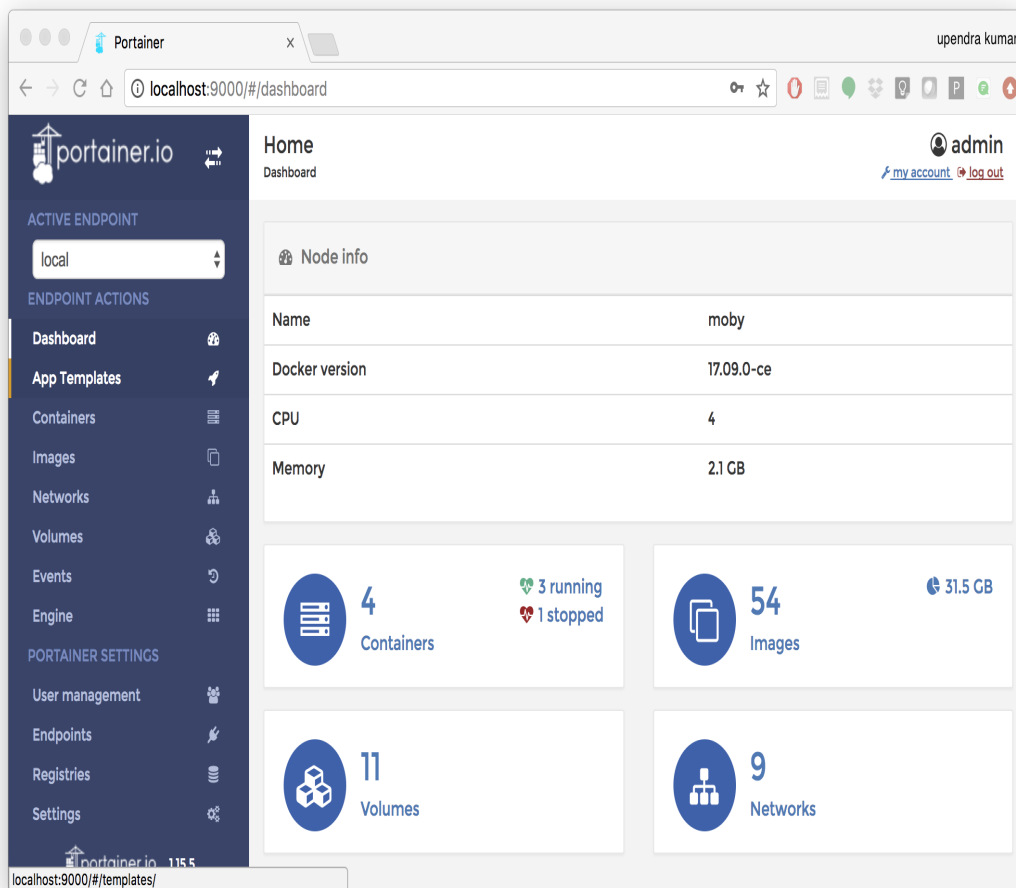
9.1.1 7.1 Installation

Use the following Docker commands to deploy Portainer. Now the second line of command should be familiar to you by now. We will talk about first line of command in the Advanced Docker session.

```
$ docker volume create portainer_data  
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v  
→portainer_data:/data portainer/portainer
```

- If you are on mac, you'll just need to access the port 9000 (<http://localhost:9000>) of the Docker engine where portainer is running using username `admin` and password `tryportainer`
- If you are running Docker on Atmosphere/Jetstream or on any other cloud, you can open `ipaddress:9000`. For my case this is `http://128.196.142.26:9000`

Note: The `-v /var/run/docker.sock:/var/run/docker.sock` option can be used in mac/linux environments only.



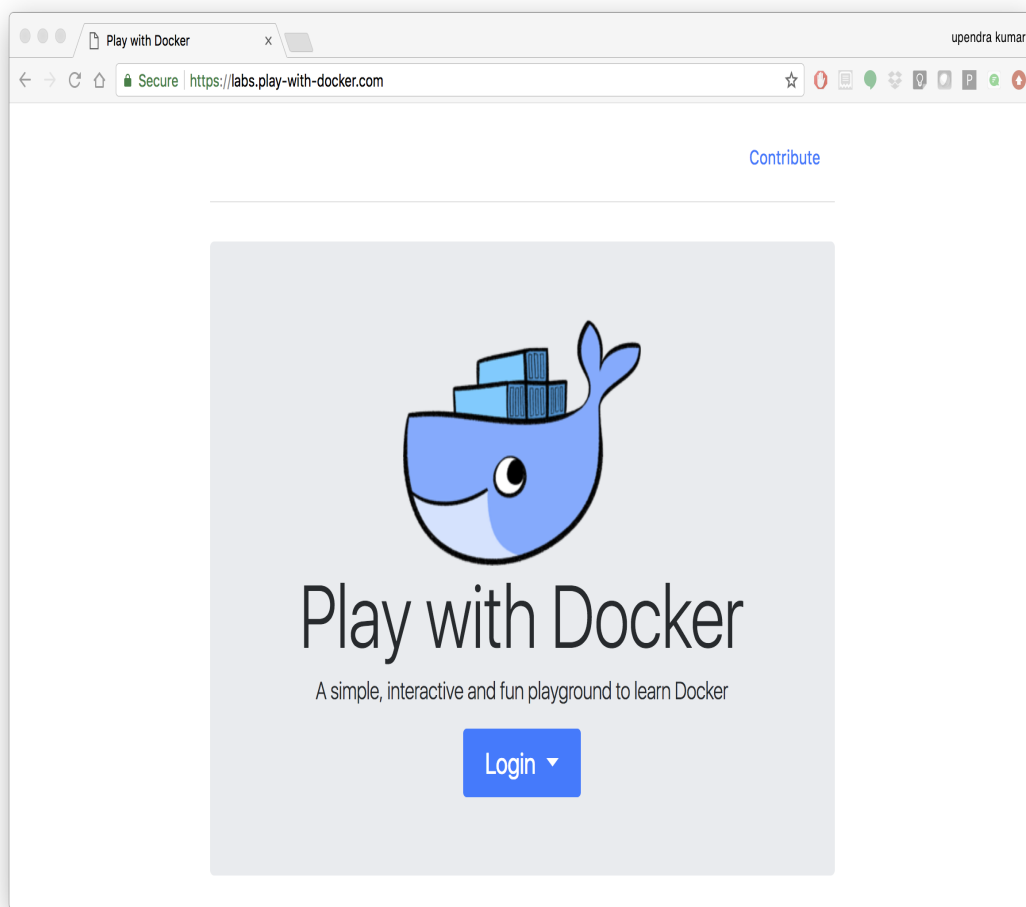
9.2 2 Play-with-docker (PWD)

PWD is a Docker playground which allows users to run Docker commands in a matter of seconds. It gives the experience of having a free Alpine Linux Virtual Machine in browser, where you can build and run Docker containers and even create clusters in **Docker Swarm Mode**. Under the hood, Docker-in-Docker (DinD) is used to give the effect of multiple VMs/PCs. In addition to the playground, PWD also includes a training site composed of a large set of Docker labs and quizzes from beginner to advanced level available at training.play-with-docker.com.

9.2.1 2.1 Installation

You don't have to install anything to use PWD. Just open <https://labs.play-with-docker.com/> and start using PWD

Note: You can use your Dockerhub credentials to log-in to PWD



Introduction to Singularity



10.1 1. Prerequisites

There are no specific skills needed for this tutorial beyond a basic comfort with the command line and using a text editor. Prior experience developing web applications could be helpful but is not required.

Note:

Important: Docker and Singularity are [friends](#) but they have distinct differences.

For more information see: [Singularity Related Resources](#)

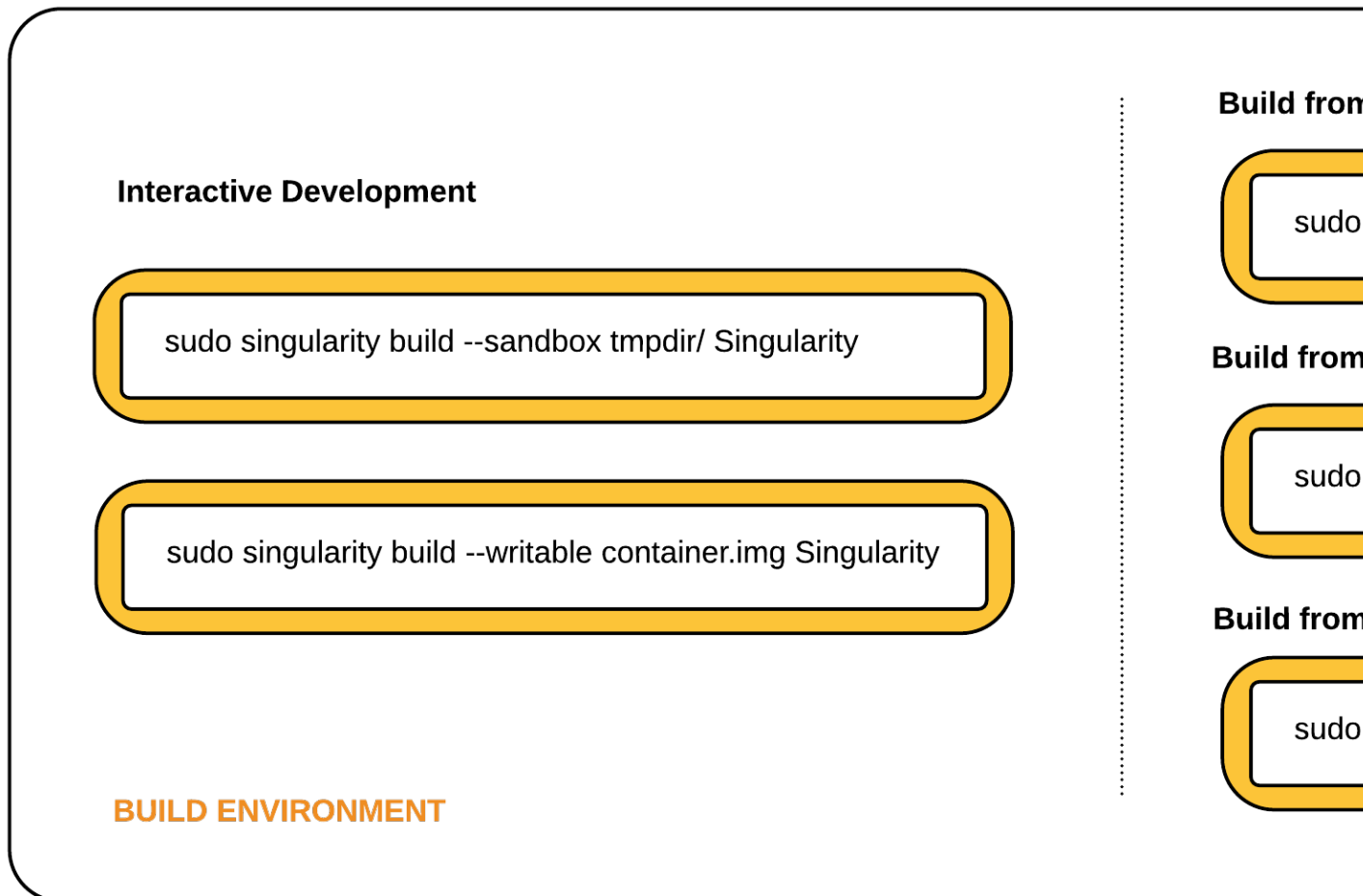
Docker:

- Inside a Docker container the user has escalated privileges, effectively making them root on the host system. This is not supported by most administrators of High Performance Computing (HPC) centers.

Singularity:

- Works on HPC systems
 - Same user inside and outside the container
 - User only has root privileges if elevated with *sudo*
 - Run (and modify!) existing Docker containers
-

Singularity uses a ‘flow’ whereby you can (1) create and modify images on your dev system, (2) build containers using recipes or pulling from repositories, and (3) execute containers on production systems.



10.2 2. Singularity installation

Local installations of Singularity are useful for testing out new containers, before pushing them to Singularity-Hub.

If you plan to use Singularity on HPC you will need a local installation of Singularity to build your own containers. HPC admins are *very* unlikely to grant you `sudo` privileges needed to build your own containers there.

10.2.1 Exercise 1 (15-20 mins)

10.2.2 2.1 Setting up your Laptop

To Install Singularity on your laptop or desktop PC follow the instructions from Singularity:

- [Windows](#)
- [Linux](#)
- [Mac](#)

Note: on Mac OS X the dependency [VagrantBox](#) is required to run Singularity

10.2.3 2.2 HPC

If you are interested in working on the UA HPC, you can load Singularity as a module. Many HPC systems are running Environment Modules with the simple command `module`. You can check to see what is available on an HPC with the command:

```
$ module avail
```

If Singularity is installed:

```
$ module load singularity
```

If you want to run Singularity on a different HPC, check their documentation. Else, you may need to contact the systems administrator and request they install the latest version of [Singularity](#).

10.2.4 2.3 XSEDE Jetstream / CyVerse Atmosphere Clouds

CyVerse staff have deployed an Ansible playbooks called `ez` installation which includes [Singularity](#) that only requires you to type a short line of code.

Start a featured instance on Atmosphere or Jetstream.

Type in the following:

```
$ ezs

* Updating ez singularity and installing singularity (this may take a few minutes, ☕
↳coffee break!)
Cloning into '/opt/cyverse-ez-singularity'...
remote: Counting objects: 11, done.
remote: Total 11 (delta 0), reused 0 (delta 0), pack-reused 11
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
```

10.2.5 2.4 Check Installation

Singularity should now be installed on your laptop or VM, or loaded on the HPC, you can check the installation with:

```
$ singularity pull shub://vsoch/hello-world
Progress |=====| 100.0%
Done. Container is at: /tmp/vsoch-hello-world-master.simg

$ singularity run vsoch-hello-world-master.simg
RaawwWWWWRRRR!!
```

View the Singularity help:

```
$ singularity --help

USAGE: singularity [global options...] <command> [command options...] ...

GLOBAL OPTIONS:
  -d|--debug      Print debugging information
  -h|--help       Display usage summary
  -s|--silent     Only print errors
  -q|--quiet      Suppress all normal output
                  --version      Show application version
  -v|--verbose    Increase verbosity +1
  -x|--sh-debug   Print shell wrapper debugging information

GENERAL COMMANDS:
  help           Show additional help for a command or container
  selftest       Run some self tests for singularity install

CONTAINER USAGE COMMANDS:
  exec           Execute a command within container
  run            Launch a runscrip[t] within container
  shell          Run a Bourne shell within container
  test           Launch a testscrip[t] within container

CONTAINER MANAGEMENT COMMANDS:
  apps           List available apps within a container
  bootstrap      *Deprecated* use build instead
  build          Build a new Singularity container
  check          Perform container lint checks
  inspect        Display container's metadata
  mount          Mount a Singularity container image
  pull           Pull a Singularity/Docker container to $PWD

COMMAND GROUPS:
  image          Container image command group
  instance       Persistent instance command group

CONTAINER USAGE OPTIONS:
  see singularity help <command>

For any additional help or support visit the Singularity
website: http://singularity.lbl.gov/
```

10.3 3. Downloading Singularity containers

The easiest way to use a Singularity container is to *pull* an existing container from one of the Container Registries maintained by the Singularity group.

10.3.1 Exercise 2 (~10 mins)

10.3.2 3.1: Pulling a Container from Singularity Hub

You can use the *pull* command to download pre-built images from a number of Container Registries, here we'll be focusing on the [Singularity-Hub](#) or [DockerHub](#).

Container Registries:

- *shub* - images hosted on Singularity Hub
- *docker* - images hosted on Docker Hub
- *localimage* - images saved on your machine
- *yum* - yum based systems such as CentOS and Scientific Linux
- *debootstrap* - apt based systems such as Debian and Ubuntu
- *arch* - Arch Linux
- *busybox* - BusyBox
- *zypper* - zypper based systems such as Suse and OpenSuse

In this example I am pulling a base Ubuntu container from Singularity-Hub:

```
$ singularity pull shub://singularityhub/ubuntu
```

You can rename the container using the *--name* flag:

```
$ singularity pull --name ubuntu_test.simg shub://singularityhub/ubuntu
```

After your image has finished downloading it should be in the present working directory, unless you specified to download it somewhere else.

```
$ singularity pull --name ubuntu_test.simg shub://singularityhub/ubuntu
Progress |=====| 100.0%
Done. Container is at: /home/***/ubuntu_test.simg
$ singularity run ubuntu_test.simg
This is what happens when you run the container...
$ singularity shell ubuntu_test.simg
Singularity: Invoking an interactive shell within container...

Singularity ubuntu_test.simg:~> cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04 LTS"
NAME="Ubuntu"
VERSION="14.04, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04 LTS"
```

(continues on next page)

(continued from previous page)

```

VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
Singularity ubuntu_test.simg:~>

```

10.3.3 3.2: Pulling containers from Docker Hub

This example pulls a container from DockerHub

Build to your container by pulling an image from Docker:

```

$ singularity pull docker://ubuntu:16.04
WARNING: pull for Docker Hub is not guaranteed to produce the
WARNING: same image on repeated pull. Use Singularity Registry
WARNING: (shub://) to pull exactly equivalent images.
Docker image path: index.docker.io/library/ubuntu:16.04
Cache folder set to /home/.../.singularity/docker
[5/5] |=====| 100.0%
Importing: base Singularity environment
Importing: /home/.../.singularity/docker/
↪sha256:1be7f2b886e89a58e59c4e685fcc5905a26ddef3201f290b96f1eff7d778e122.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:6fbc4a21b806838b63b774b338c6ad19d696a9e655f50b4e358cc4006c3baa79.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:c71a6f8e13782fed125f2247931c3eb20cc0e6428a5d79edb546f1f1405f0e49.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:4be3072e5a37392e32f632bb234c0b461ff5675ab7e362afad6359fbd36884af.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:06c6d2f5970057aef3aef6da60f0fde280db1c077f0cd88ca33ec1a70a9c7b58.tar.gz
Importing: /home/.../.singularity/metadata/
↪sha256:c6a9ef4b9995d615851d7786fbc2fe72f72321bee1a87d66919b881a0336525a.tar.gz
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: ./ubuntu-16.04.simg
Cleaning up...
Done. Container is at: ./ubuntu-16.04.simg

```

Note, there are some Warning messages concerning the build from Docker.

The example below does the same as above, but renames the image.

```

$ singularity pull --name ubuntu_docker.simg docker://ubuntu
Importing: /home/.../.singularity/docker/
↪sha256:c71a6f8e13782fed125f2247931c3eb20cc0e6428a5d79edb546f1f1405f0e49.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:4be3072e5a37392e32f632bb234c0b461ff5675ab7e362afad6359fbd36884af.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:06c6d2f5970057aef3aef6da60f0fde280db1c077f0cd88ca33ec1a70a9c7b58.tar.gz
Importing: /home/.../.singularity/metadata/
↪sha256:c6a9ef4b9995d615851d7786fbc2fe72f72321bee1a87d66919b881a0336525a.tar.gz
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: ./ubuntu_docker.simg

```

(continues on next page)

(continued from previous page)

```
Cleaning up...
Done. Container is at: ./ubuntu_docker.simg
```

When we run this particular Docker container, without any runtime arguments notice that it does not return any notifications, and the Bash prompt does not change the prompt.

```
$ singularity run ubuntu_docker.simg
$ cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.3 LTS"
NAME="Ubuntu"
VERSION="16.04.3 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.3 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
```

Whoa, we're inside the container!?! This is the OS on the VM I tested this on:

```
$ exit
exit
$ cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.1 LTS"
NAME="Ubuntu"
VERSION="16.04.1 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.1 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
```

By exiting, I can check the OS again to make sure that everything is back. Here we are back in the container using the shell invocation:

```
$ singularity shell ubuntu_docker.simg
Singularity: Invoking an interactive shell within container...

Singularity ubuntu_docker.simg:~> cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.3 LTS"
```

(continues on next page)

(continued from previous page)

```
NAME="Ubuntu"
VERSION="16.04.3 LTS (Xenial Xerus) "
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.3 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
Singularity ubuntu_docker.simg:~>
```

When starting a container, make sure that it notifies you it is running. This is typically achieved by the basic \$ prompt providing some metadata:

```
Singularity ubuntu_docker.simg:~>
```

Keeping track of downloaded images may be necessary if space is a concern.

10.3.4 3.3: Keeping track of downloaded containers

By default, Singularity uses a temporary cache to hold Docker tarballs:

```
$ ls ~/.singularity
```

You can change these by specifying the location of the cache and temporary directory on your localhost:

```
$ sudo mkdir tmp
$ sudo mkdir scratch

$ SINGULARITY_TMPDIR=$PWD/scratch SINGULARITY_CACHEDIR=$PWD/tmp singularity --debug_
→pull --name ubuntu-tmpdir.simg docker://ubuntu
```

As an example, using Singularity we can run a UI program that was built from Docker, here I show the IDE RStudio *tidyverse* from [Rocker](#)

```
$ singularity exec docker://rocker/tidyverse:latest R
```

“An Introduction to Rocker: Docker Containers for R by Carl Boettiger, Dirk Eddelbuettel”

10.4 4. Building Singularity containers

Like Docker, which uses a *dockerfile* to build its containers, Singularity uses a file called *Singularity*

When you are building locally, you can name this file whatever you wish, but a better practice is to put it in a directory and name it *Singularity* - as this will help later on when developing on Singularity-Hub and Github.

Create Container and add content to it:

```
$ singularity image.create ubuntu14.simg
Creating empty 768MiB image file: ubuntu14.simg
Formatting image with ext3 file system
```

(continues on next page)

(continued from previous page)

```
Image is done: ubuntu14.simg

$ singularity build ubuntu14.simg docker://ubuntu:14.04
Building into existing container: ubuntu14.simg
Docker image path: index.docker.io/library/ubuntu:14.04
Cache folder set to /home/.../.singularity/docker
[5/5] |=====| 100.0%
Importing: base Singularity environment
Importing: /home/.../.singularity/docker/
↪sha256:c954d15f947c57e059f67a156ff2e4c36f4f3e59b37467ff865214a88ebc54d6.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:c3688624ef2b94ab3981564e23e1f48df8f1b988519373ccfb79d7974017cb85.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:848fe4263b3b44987f0eacdb2fc0469ae6ff04b2311e759985dfd27ae5d3641d.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:23b4459d3b04aa0bc7cb7f7021e4d7bbb5e87aa74a6a5f57475a0e8badbd9a26.tar.gz
Importing: /home/.../.singularity/docker/
↪sha256:36ab3b56c8f1a3188464886cbe41f42a969e6f9374e040f13803d796ed27b0ec.tar.gz
Importing: /home/.../.singularity/metadata/
↪sha256:c6a9ef4b9995d615851d7786fbc2fe72f72321bee1a87d66919b881a0336525a.tar.gz
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: ubuntu14.simg
Cleaning up...
```

Note, *image.create* uses an ext3 file system

Create a container using a custom Singularity file:

```
$ singularity build --name ubuntu.simg Singularity
```

In the above command:

- *--name* will create a container named *ubuntu.simg*

Pull a Container from Docker and make it writable using the *--writable* flag:

```
$ sudo singularity build --writable ubuntu.simg docker://ubuntu

Docker image path: index.docker.io/library/ubuntu:latest
Cache folder set to /root/.singularity/docker
Importing: base Singularity environment
Importing: /root/.singularity/docker/
↪sha256:1be7f2b886e89a58e59c4e685fcc5905a26ddef3201f290b96f1eff7d778e122.tar.gz
Importing: /root/.singularity/docker/
↪sha256:6fbc4a21b806838b63b774b338c6ad19d696a9e655f50b4e358cc4006c3baa79.tar.gz
Importing: /root/.singularity/docker/
↪sha256:c71a6f8e13782fed125f2247931c3eb20cc0e6428a5d79edb546f1f1405f0e49.tar.gz
Importing: /root/.singularity/docker/
↪sha256:4be3072e5a37392e32f632bb234c0b461ff5675ab7e362afad6359fbd36884af.tar.gz
Importing: /root/.singularity/docker/
↪sha256:06c6d2f5970057aef3aef6da60f0fde280db1c077f0cd88ca33ec1a70a9c7b58.tar.gz
Importing: /root/.singularity/metadata/
↪sha256:c6a9ef4b9995d615851d7786fbc2fe72f72321bee1a87d66919b881a0336525a.tar.gz
Creating empty Singularity writable container 120MB
Creating empty 150MiB image file: ubuntu.simg
Formatting image with ext3 file system
```

(continues on next page)

(continued from previous page)

```

Image is done: ubuntu.simg
Building Singularity image...
Singularity container built: ubuntu.simg
Cleaning up...

$ singularity shell ubuntu.simg

Singularity: Invoking an interactive shell within container...

Singularity ubuntu.simg:~> apt-get update

Reading package lists... Done
W: chmod 0700 of directory /var/lib/apt/lists/partial failed -
↳SetupAPTPartialDirectory (1: Operation not permitted)
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
Singularity ubuntu.simg:~> exit
exit

$ sudo singularity shell ubuntu.simg

Singularity: Invoking an interactive shell within container...

Singularity ubuntu.simg:~> apt-get update

Hit:1 http://archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [73.2 kB]
Get:6 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [585 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [405
↳kB]
Get:9 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages
↳[3486 B]
Get:10 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial-updates/universe Sources [241 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [953 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [13.
↳1 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [762
↳kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 Packages [18.
↳5 kB]
Get:17 http://archive.ubuntu.com/ubuntu xenial-backports/main amd64 Packages [5153 B]
Get:18 http://archive.ubuntu.com/ubuntu xenial-backports/universe amd64 Packages
↳[7168 B]
Fetched 23.2 MB in 4s (5569 kB/s)
Reading package lists... Done

Singularity ubuntu.simg:~> apt-get install curl --fix-missing

```

When I try to install software to the image without *sudo* it is denied, because root is the owner of the container. When I use *sudo* I can install software to the container. The software remain in the container after closing the container and restart.

Note: Bootstrapping *bootstrap* command is deprecated (v2.4), use *build* instead.

To install a container with Ubuntu from the ubuntu.com repository you need to use *debootstrap*

10.4.1 4.1: Docker2Singularity & Singularity2Docker

One of the other features of Singularity is the ability to convert [Docker containers to Singularity Containers](#), and [Singularity containers to Docker containers](#)

10.4.2 4.2: Exercise 3 (30 minutes): Create the Singularity file

[Recipes](#) can use any number of container registries for bootstrapping a container.

(Advanced) the *Singularity* file can be hosted on Github and will be auto-detected by Singularity-Hub when you set up your Container Collection.

Building your own containers requires that you have *sudo* privileges - therefore you'll need to develop these on your local machine or on a VM that you can gain root access on.

- The Header

The top of the file, selects the base OS for the container. *Bootstrap*: references the repository (e.g. *docker*, *debootstrap*, *sub*). *From*: selects the name of the owner/container.

```
Bootstrap: shub
From: vsoc/hello-world
```

Using *debootstrap* with a build that uses a mirror:

```
BootStrap: debootstrap
OSVersion: xenial
MirrorURL: http://us.archive.ubuntu.com/ubuntu/
```

Using a *localimage* to build:

```
Bootstrap: localimage
From: /path/to/container/file/or/directory
```

Using CentOS-like container:

```
Bootstrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-7/7/os/x86_64/
Include:yum
```

Note: to use *yum* to build a container you should be operating on a RHEL system, or an Ubuntu system with *yum* installed.

The container registries which Singularity uses are listed above in Section 3.1.

- The Singularity file uses sections to specify the dependencies, environmental settings, and runscripts when it build.
- *%help* - create text for a help menu associated with your container
- *%setup* - executed on the host system outside of the container, after the base OS has been installed.

- `%files` - copy files from your host system into the container
- `%labels` - store metadata in the container
- `%environment` - loads environment variables at the time the container is run (not built)
- `%post` - set environment variables during the build
- `%runscript` - executes a script when the container runs
- `%test` - runs a test on the build of the container
- Apps

In Singularity 2.4+ we can build a container which does multiple things, e.g. each app has its own runscripts. These use the prefix `%app` before the sections mentioned above. The `%app` architecture can exist in addition to the regular `%post` and `%runscript` sections.

```
Bootstrap: docker
From: ubuntu

% environment

%labels

#####
# foo
#####

%apprun foo
    exec echo "RUNNING FOO"

%applabels foo
    BESTAPP=FOO
    export BESTAPP

%appinstall foo
    touch foo.exec

%appenv foo
    SOFTWARE=foo
    export SOFTWARE

%apphelp foo
    This is the help for foo.

%appfiles foo
    avocados.txt

#####
# bar
#####

%apphelp bar
    This is the help for bar.

%applabels bar
    BESTAPP=BAR
    export BESTAPP
```

(continues on next page)

(continued from previous page)

```
%appinstall bar
    touch bar.exec

%appendv bar
    SOFTWARE=bar
    export SOFTWARE
```

- Setting up Singularity file system

%help section can be as verbose as you want

```
Bootstrap: docker
From: ubuntu

%help
This is the container help section.
```

%setup commands are executed on the localhost system outside of the container - these files could include necessary build dependencies. We can copy files to the `$SINGULARITY_ROOTFS` file system can be done during *%setup*

%files include any files that you want to copy from your localhost into the container.

%post includes all of the environment variables and dependencies that you want to see installed into the container at build time.

%environment includes the environment variables which we want to be run when we start the container

%runscript does what it says, it executes a set of commands when the container is run.

Example Singularity file bootstrapping a [Docker](#) Ubuntu (16.04) image.

```
BootStrap: docker
From: ubuntu:16.04

%post
    apt-get -y update
    apt-get -y install fortune cowsay lolcat

    # create bind points for additional storage
    mkdir /scratch

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    fortune | cowsay | lolcat

%labels
    Maintainer Tyson Swetnam
    Version v0.1
```

Build the container:

```
singularity build --name cowsay_container.simg Singularity
```

Run the container:

```
singularity run cowsay.simg
```

If you build a *squashfs* container, it is immutable (you cannot *–writable* edit it)

10.5 5. Running Singularity containers

Commands:

exec - command allows you to execute a custom command within a container by specifying the image file.

shell - command allows you to spawn a new shell within your container and interact with it.

run - assumes your container is set up with “runscripts” triggered with the *run* command, or simply by calling the container as though it were an executable.

inspect - inspects the container.

–writable - creates a writable container that you can edit interactively and save on exit.

–sandbox - copies the guts of the container into a directory structure.

10.5.1 5.1 Using the *exec* command

```
$ singularity exec shub://singularityhub/ubuntu cat /etc/os-release
```

10.5.2 5.2 Using the *shell* command

```
$ singularity shell shub://singularityhub/ubuntu
```

10.5.3 5.3 Using the *run* command

```
$ singularity run shub://singularityhub/ubuntu
```

10.5.4 5.4 Using the *inspect* command

You can inspect the build of your container using the *inspect* command

```
$ singularity pull shub://vsoch/hello-world
Progress |=====| 100.0%
Done. Container is at: /home/***/vsoch-hello-world-master-latest.simg

$ singularity inspect vsoch-hello-world-master-latest.simg
{
  "org.label-schema.usage.singularity.deffile.bootstrap": "docker",
  "MAINTAINER": "vanessasaur",
  "org.label-schema.usage.singularity.deffile": "Singularity",
  "org.label-schema.schema-version": "1.0",
  "WHATAMI": "dinosaur",
  "org.label-schema.usage.singularity.deffile.from": "ubuntu:14.04",
  "org.label-schema.build-date": "2017-10-15T12:52:56+00:00",
```

(continues on next page)

(continued from previous page)

```

"org.label-schema.usage.singularity.version": "2.4-feature-squashbuild-secbuild.
↪g780c84d",
"org.label-schema.build-size": "333MB"
}

```

10.5.5 5.5 Using the `--sandbox` and `--writable` commands

As of Singularity v2.4 by default *build* produces immutable images in the ‘squashfs’ file format. This ensures reproducible and verifiable images.

Creating a `--writable` image must use the *sudo* command, thus the owner of the container is *root*

```

$ sudo singularity build --writable ubuntu-master.simg shub://singularityhub/ubuntu
Cache folder set to /root/.singularity/shub
Progress |=====| 100.0%
Building from local image: /root/.singularity/shub/singularityhub-ubuntu-master-
↪latest.simg
Creating empty Singularity writable container 208MB
Creating empty 260MiB image file: ubuntu-master.simg
Formatting image with ext3 file system
Image is done: ubuntu-master.simg
Building Singularity image...
Singularity container built: ubuntu-master.simg
Cleaning up...

```

You can convert these images to writable versions using the `--writable` and `--sandbox` commands.

When you use the `--sandbox` the container is written into a directory structure. Sandbox folders can be created without the *sudo* command.

```

$ singularity build --sandbox lolcow/ shub://GodloveD/lolcow
WARNING: Building sandbox as non-root may result in wrong file permissions
Cache folder set to /home/.../.singularity/shub
Progress |=====| 100.0%
Building from local image: /home/.../.singularity/shub/GodloveD-lolcow-master-latest.
↪simg
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Singularity container built: lolcow/
Cleaning up...
@vml42-73:~$ cd lolcow/
@vml42-73:~/lolcow$ ls
bin boot dev environment etc home lib lib64 media mnt opt proc run sbin ↪
↪singularity srv sys tmp usr var

```

10.5.6 5.6 Test

Singularity can test the build of your container.

You can bypass the test by using `--no-test`.

10.5.7 5.7 Bind Paths

When Singularity creates the new file system inside a container it ignores directories that are not part of the standard kernel, e.g. `/scratch`, `/xdisk`, `/global`, etc. These paths can be added back into the container by binding them when the container is run.

```
$ singularity shell --bind /xdisk ubuntu14.simg
```

The system administrator can also define what is added to a container. This is important on campus HPC systems that often have a `/scratch` or `/xdisk` directory structure. By editing the `/etc/singularity/singularity.conf` a new path can be added to the system containers.

10.5.8 5.8 Overlay

You can make changes to an immutable container which only persist for the duration of the container being run.

First, download a container.

Next, create a new image in the ext3 format.

```
$ singularity image.create blank_slate.simg
```

Now, overlay your blank image file name with the container you just downloaded.

```
$ sudo singularity shell --overlay blank_slate.simg ubuntu14.simg
```

note: using the 'sudo' command to make the container writable

10.6 6. Singularity-Hub

You can host containers on Singularity's own container registry [Singularity-Hub](#)

Connect a GitHub repo to the Hub which contains a Singularity file. The image will be built automatically and be hosted on the Hub.

You can pull your built images from the `shub://` followed by your Github user identity and repo name.

Advanced Singularity

lsingularityl

11.1 1. Using HPC Environments

Conducting analyses on high performance computing clusters happens through very different patterns of interaction than running analyses on a VM. When you login, you are on a node that is shared with lots of people. Trying to run jobs on that node is not “high performance” at all. Those login nodes are just intended to be used for moving files, editing files, and launching jobs.

Most jobs on an HPC cluster are neither interactive, nor realtime. When you submit a job to the scheduler, you must tell it what resources you need (e.g. how many nodes, what type of nodes) and what you want to run. Then the scheduler finds resources matching your requirements, and runs the job for you when it can.

You can run a simple command on the login node as opposed to a large job:

```
module load singularity
singularity exec docker://python:latest python --version
```

The “User’s Guide” for Ocelote can be found at: <https://docs.hpc.arizona.edu>

11.1.1 How do HPC systems fit into the development workflow?

A few things to consider when using HPC systems:

1. Using ‘sudo’ is not allowed on HPC systems, and building a Singularity container from scratch requires sudo. That means you have to build your containers on a different development system. You can pull a docker image on HPC systems.
2. If you need to edit text files, command line text editors don’t support using a mouse, so working efficiently has a learning curve. There are text editors that support editing files over SSH. This lets you use a local text editor and just save the changes to the HPC system.

3. Singularity has changed image formats. Depending on the version of Singularity running on the HPC system, new squashFS or .sing formats may not work. The images take a lot less space
4. You can't run Docker containers - security stuff!

11.1.2 Tutorial #1

This is a review of knowledge already covered in this workshop. This is optional. If you create this container, it will have to be where you have root authority. Or, at the point where this container is transferred to HPC, you can use one you have created previously.

1. Build this where you have root authority
2. Some packages are more complex than "pip install numpy"
3. Include your bind points

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum wget
# best to build up container using kickstart mentality.
# ie, to add more packages to image,
# re-run bootstrap command again.
# bootstrap on existing image will build on top of it, not overwriting it/restarting.
↳ from scratch
# singularity .def file is like kickstart file
# unix commands can be run, but if there is any error, the bootstrap process ends
%setup
# commands to be executed on host outside container during bootstrap
%post
# commands to be executed inside container during bootstrap
# add python and install some packages
yum -y install vim wget python epel-release
yum -y install python-pip
# install tensorflow
pip install --upgrade pip
pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/
↳ tensorflow-0.9.0-cp27-none-linux_x86_64.whl
pip install --upgrade numpy scipy astropy
# create bind points for storage.
mkdir /extra
mkdir /xdisk
exit 0
%runscript
# commands to be executed when the container runs
echo "Arguments received: $*"
exec /usr/bin/python "$@"
%test
# commands to be executed within container at close of bootstrap process
```

Create container

```
singularity build astropy.img astropy.recipe
```

The next step is to copy this singularity to one of your directories on Ocelote. For example:

```
scp astropy.img chrisreidy@filexfer.hpc.arizona.edu:
```

Log into home directory on Ocelote then “mv” file to /extra/chrisreidy/singularity Test with these commands

```
$ module load singularity
$ singularity exec astropy.img python --version
Python 2.7.5
```

On an HPC system, your job submission script would look something like:

```
###=====
#!/bin/bash
#PBS -N singularity-job
#PBS -W group_list=GroupName
#PBS -q standard
#PBS -l select=1:ncpus=1:mem=6gb
#PBS -l walltime=01:00:00
#PBS -l cput=12:00:00
module load singularity
cd /extra/chrisreidy/singularity
date
singularity exec astropy.img python --version
date
```

This example uses PBS which is the scheduler available on Ocelote. ElGato uses LSF which has the same functions but different syntax. Run the job:

```
qsub astropy.pbs
```

It is usually possible to get an interactive session as well. For example:

```
qsub -I -N jobname -W group_list=YourGroup -q windfall -l select=1:ncpus=28:mem=168gb
↪ -l cput=1:0:0 -l walltime=1:0:0
```

11.2 2. Singularity and MPI

Singularity supports MPI fairly well. Since (by default) the network is the same inside and outside the container, the communication between containers usually just works. The more complicated bit is making sure that the container has the right set of MPI libraries. MPI is an open specification, but there are several implementations (OpenMPI, MVA-PICH2, and Intel MPI to name three which are available on Ocelote) with some non-overlapping feature sets. If the host and container are running different MPI implementations, or even different versions of the same implementation, tragedy may ensue.

The general rule is that you want the version of MPI inside the container to be the same version or newer than the host. You may be thinking that this is not good for the portability of your container, and you are right. Containerizing MPI applications is not terribly difficult with Singularity, but it comes at the cost of additional requirements for the host system.

Note: Many HPC Systems, like Ocelote, have highspeed, low latency networks that have special drivers. Ocelote and ElGato use Infiniband. When running MPI jobs, if the container doesn’t have the right libraries, it won’t be able to use those special interconnects to communicate between nodes.

Because you may have to build your own MPI enabled Singularity images (to get the versions to match), here is a 2.3 compatible example of what it may look like:

```
# Copyright (c) 2015-2016, Gregory M. Kurtzer. All rights reserved.
#
# "Singularity" Copyright (c) 2016, The Regents of the University of California,
# through Lawrence Berkeley National Laboratory (subject to receipt of any
# required approvals from the U.S. Dept. of Energy). All rights reserved.

BootStrap: debootstrap
OSVersion: xenial
MirrorURL: http://us.archive.ubuntu.com/ubuntu/

%runscript
    echo "This is what happens when you run the container..."

%post
    echo "Hello from inside the container"
    sed -i 's/$/ universe/' /etc/apt/sources.list
    apt update
    apt -y --allow-unauthenticated install vim build-essential wget      gfortran
    ↪bison libibverbs-dev libibmad-dev libibumad-dev librdmacm-dev      libmlx5-dev
    ↪libmlx4-dev
    wget http://mvapich.cse.ohio-state.edu/download/mvapich/mv2/      mvapich2-2.1.tar.
    ↪gz
    tar xvf mvapich2-2.1.tar.gz
    cd mvapich2-2.1
    ./configure --prefix=/usr/local
    make -j4
    make install
    /usr/local/bin/mpicc examples/hellow.c -o /usr/bin/hellow
```

You could also build in everything in a Dockerfile and convert the image to Singularity at the end.

Once you have a working MPI container, invoking it would look something like:

```
module load mvapich2
mpirun -np 4 singularity exec ./mycontainer.img /app.py arg1 arg2
```

This will use the **host MPI** libraries to run in parallel, and assuming the image has what it needs, can work across many nodes.

For a single node, you can also use the **container MPI** to run in parallel (usually you don't want this)

```
module load mvapich2
singularity exec ./mycontainer.img mpirun -np 4 /app.py arg1 arg2
```

11.3 3. Singularity and GPU Computing

GPU support in Singularity is fantastic

Since Singularity supported docker containers, it has been fairly simple to utilize GPUs for machine learning code like TensorFlow. On Ocelote we have downloaded Docker images from Nvidia for most ML workflows, and converted them to Singularity. They are kept in `/unsupported/singularity/nvidia`, and can be copied to your own directories.

11.3.1 Tutorial #2

This example is a case of running a simple container using an interactive session. You don't need to know anything about machine learning. From Ocelote:

```
cd /extra/netid
mkdir astro
cd astro
cp /unsupported/singularity/nvidia/nvidia-tensorflow.18.03-py3.simg .
cp /unsupported/singularity/nvidia/tensorflow_example.py .
# Work from a compute node. This step is likely to take more than a minute depending
  ↳ on how busy the scheduler is.
qsub -I -N jobname -m bea -W group_list=YourGroup -q standard -l
  ↳ select=1:ncpus=28:mem=168gb:ngpus=1 -l cput=1:0:0 -l walltime=1:0:0
# Load the singularity module
module load singularity
cd /extra/netid/astro
singularity exec --nv nvidia-tensorflow.18.03-py3.simg python tensorflow_example.py
```

Please note that the `--nv` flag specifically passes the GPU drivers into the container. If you leave it out, the GPU will not be detected.

11.3.2 Tutorial #3

This example is a little different. It demonstrates the ability to pull a Docker image, embed it in Singularity and run it on a GPU node. For TensorFlow, you can directly pull their latest GPU image and utilize it as follows.

```
# Start an interactive session after you are on the login node. Edit as needed:
qsub -I -N jobname -W group_list=YourGroup -q standard -l
  ↳ select=1:ncpus=28:mem=168gb:ngpus=1 -l cput=1:0:0 -l walltime=1:0:0
cd /extra/netid/astro
# Get the software
git clone https://github.com/tensorflow/models.git ~/models
# Pull the image
singularity pull docker://tensorflow/tensorflow:latest-gpu
# Run the code
singularity exec --nv tensorflow-latest-gpu.simg python $HOME/models/tutorials/image/
  ↳ mnist/convolutional.py
```

Note: You probably noticed that we check out the models repository into your `$HOME` directory. This is because your `$HOME` and `$WORK` directories are only available inside the container if the root folders `/home` and `/work` exist inside the container. In the case of `tensorflow-latest-gpu.simg`, the `/work` directory does not exist, so any files there are inaccessible to the container.

Booting an Atmosphere computer instance for your use!

What we're going to walk through how to start up a running computer (an "instance") on the CyVerse Atmosphere Cloud service.

Below, we've provided screenshots of the whole process. You can click on them to zoom in. The important areas to fill in are highlighted.

First, go to the [Atmosphere](#) application and then click *login*

Important: As described in the [pre-workshop setup](#), you need to have access to the CyVerse Atmosphere Cloud. If you are not able to log-in for some reason, please let us know and we will fix it immediately.

1. Fill in the username and password and click "LOGIN"
 - Fill in the username, which is your CyVerse username, and then enter the password (which is your CyVerse password).

Enter your Username and Password

Username:
ckc

Password:
.....

☐ Warn me before logging me into other sites.

LOGIN [clear](#) [Register](#)

Additional Help?

[Need to reset your password?](#)

[Contact Support](#)

[Other questions?](#)

For security reasons, please Log Out and Exit your web browser when you are done accessing services that require authentication!

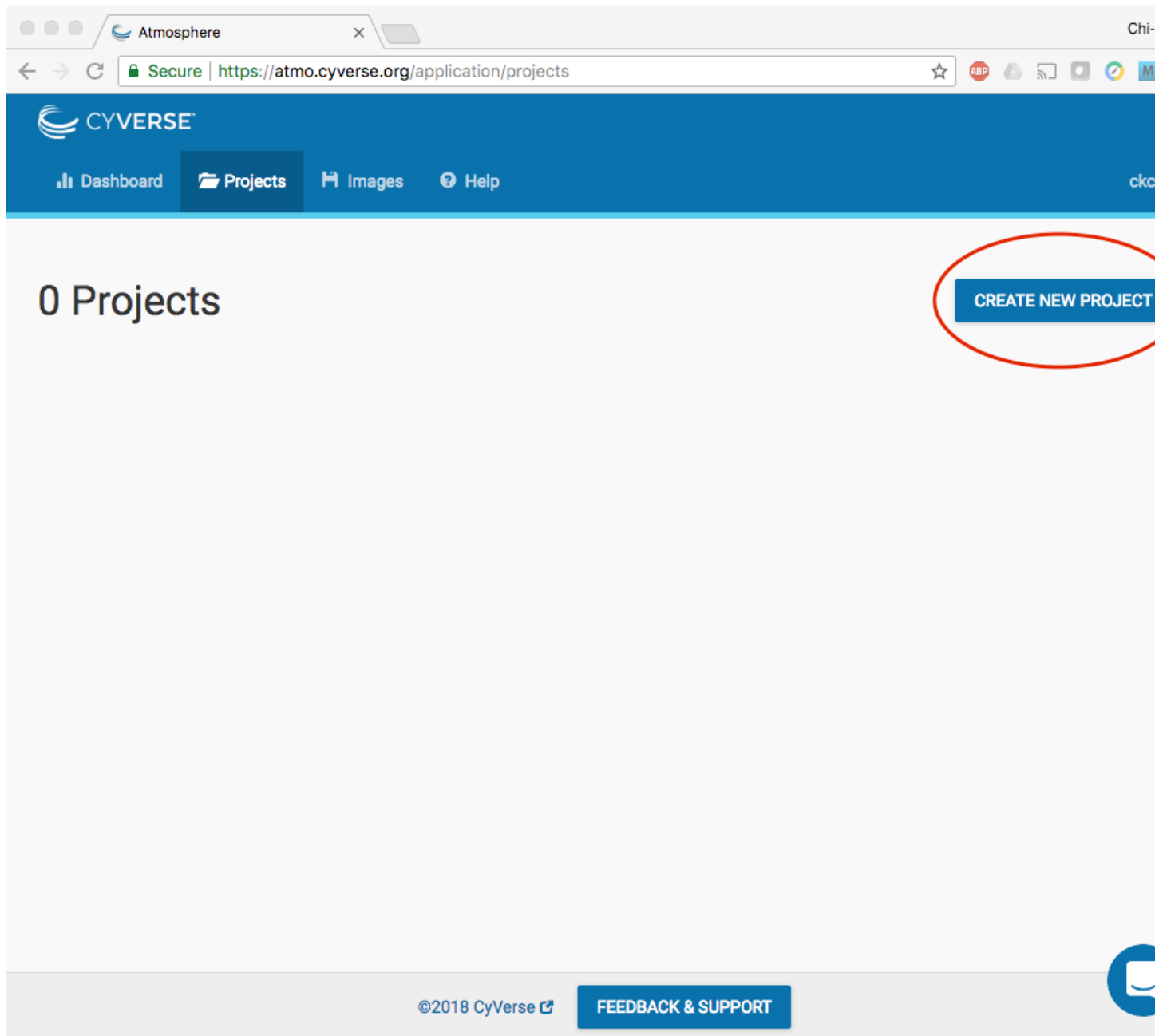
Languages:

[English](#) | [Spanish](#) | [French](#) | [Russian](#) | [Nederlands](#) | [Svenskt](#) | [Italiano](#) | [Urdu](#) | [Chinese \(Simplified\)](#) | [Deutsch](#) | [Japanese](#) | [Croatian](#) | [Czech](#) | [Slovenian](#) | [Catalan](#) | [Macedonian](#) | [Polish](#)

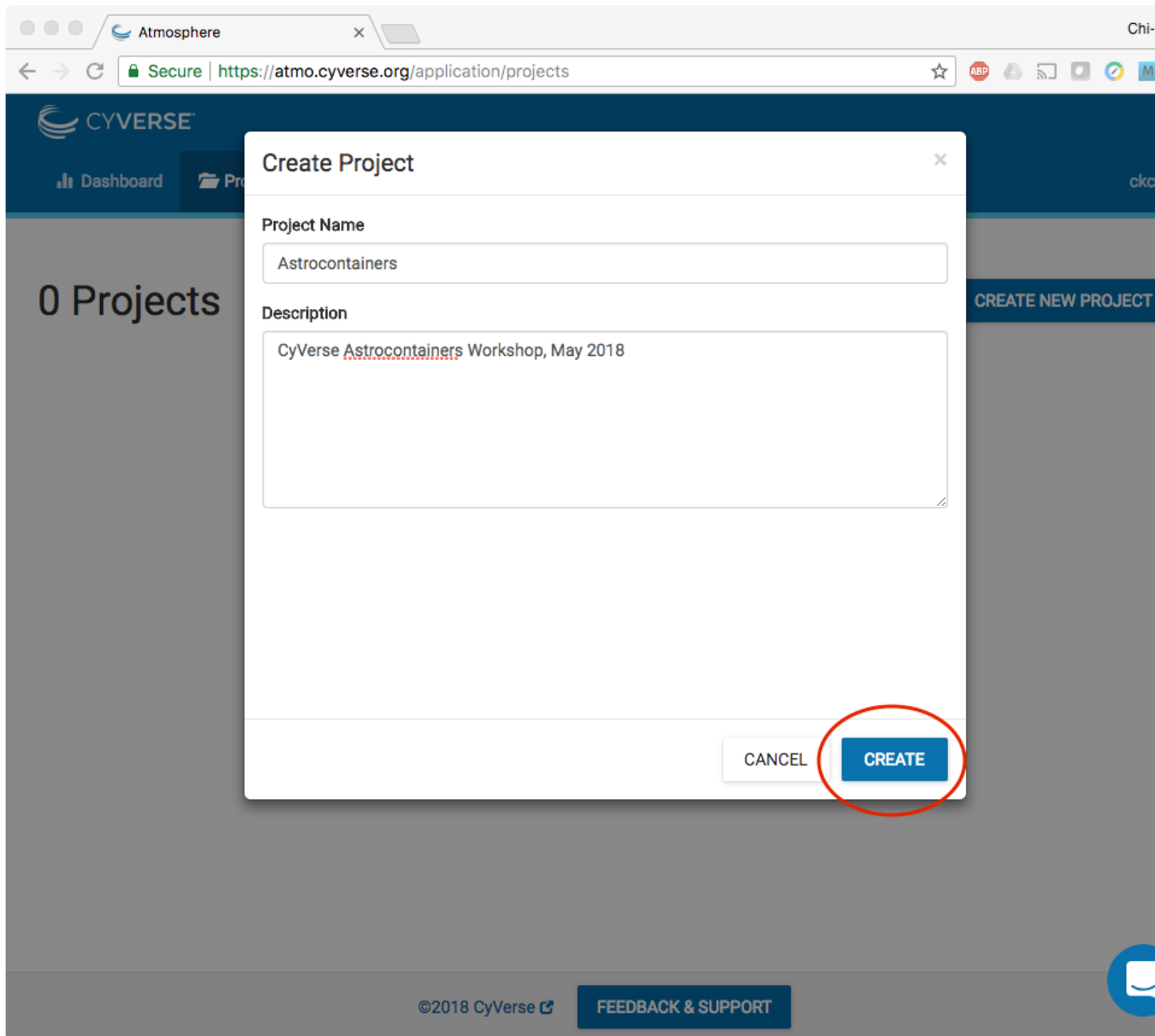
Copyright © 2005 - 2010 Jasig, Inc. All rights reserved.
Powered by [Jasig Central Authentication Service 4.0.1](#)

JASIG

2. Select the “Projects” tab and then click the “CREATE NEW PROJECT” button
 - This is something you only need to do once.
 - A project is a workspace that lets you keep things together.
 - Click on the “Projects” tab on the top and then click the “CREATE NEW PROJECT” button

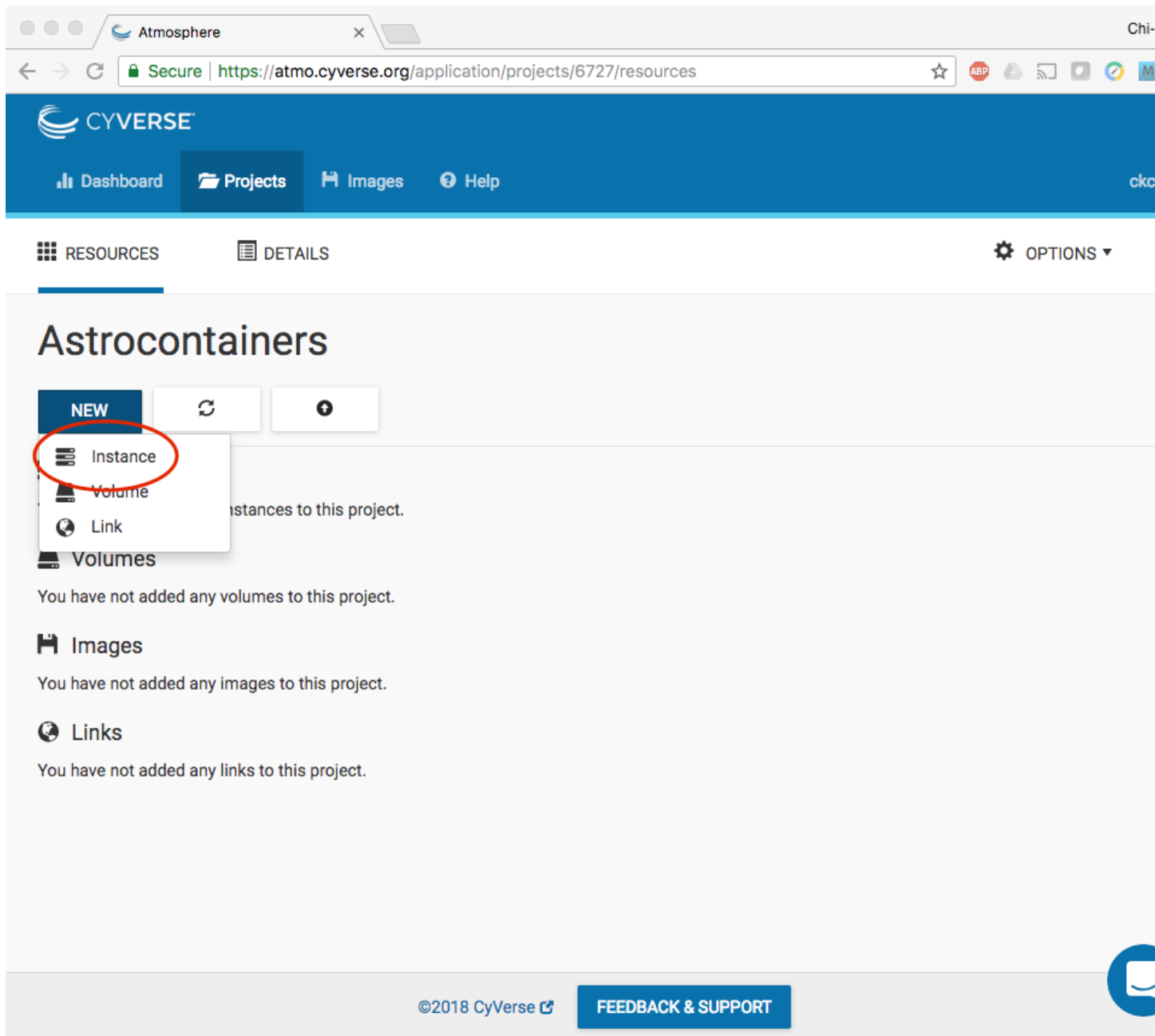


- Enter the name “Astrocontainers” into the Project Name and put something simple like “CyVerse AstroContainers Workshop, May 2018” into the description. Then click “CREATE”.



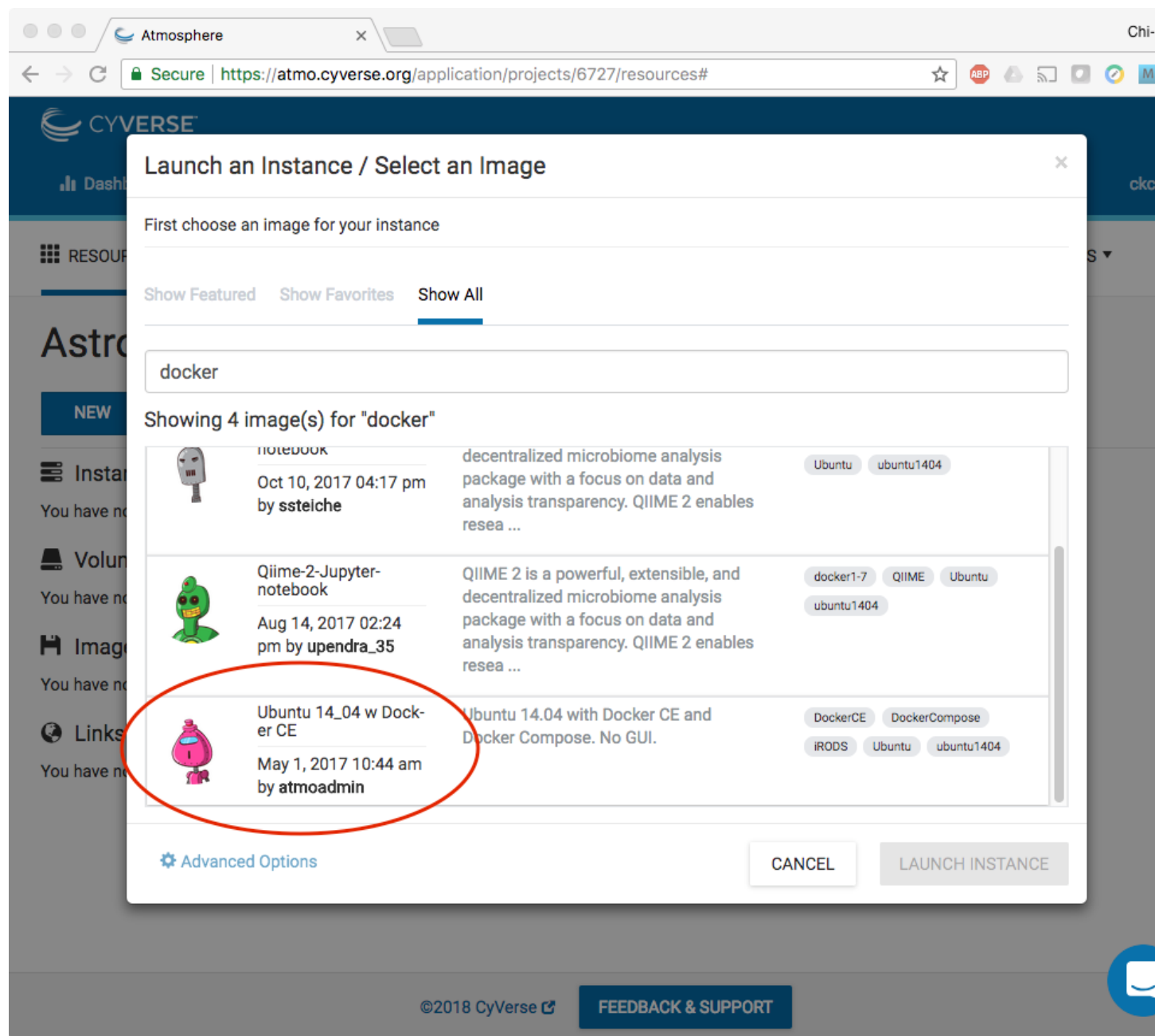
3. Select the newly created project

- Click on your newly created project.
- Click “NEW” and then “Instance” from the dropdown menu to start up a new virtual machine.

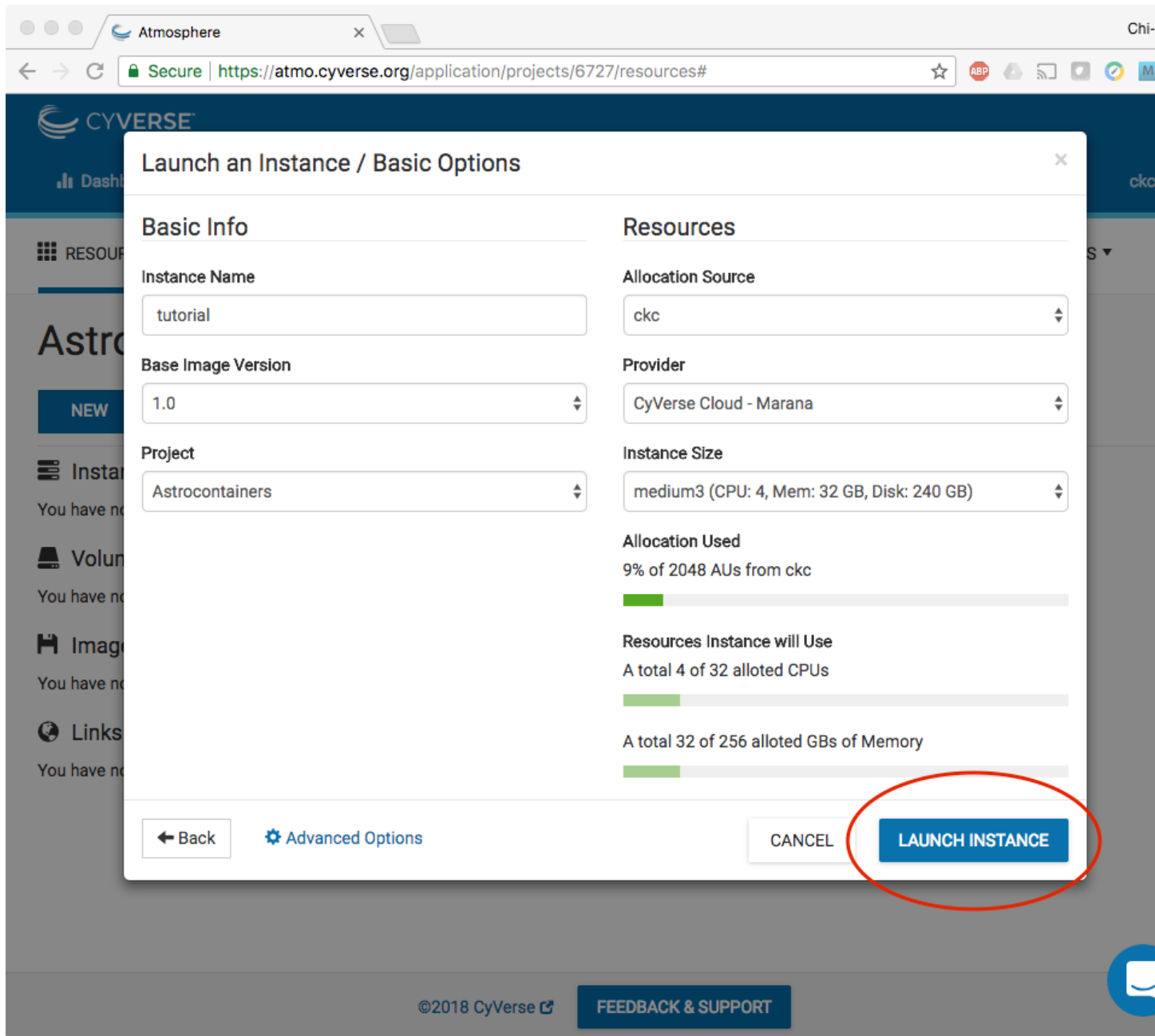


The screenshot shows the CyVerse Atmosphere web interface. The browser address bar displays the URL <https://atmo.cyverse.org/application/projects/6727/resources>. The interface has a blue header with the CyVerse logo and navigation links: Dashboard, Projects, Images, and Help. Below the header, there are tabs for RESOURCES and DETAILS, with an OPTIONS menu on the right. The main content area is titled 'Astrocontainers'. Under the 'NEW' button, a dropdown menu is open, showing three options: 'Instance' (selected), 'Volume', and 'Link'. Below the dropdown, there are sections for 'Volumes', 'Images', and 'Links', each with a message stating 'You have not added any [resource] to this project.' The footer contains the copyright notice '©2018 CyVerse' and a 'FEEDBACK & SUPPORT' button.

- Search for “docker” in the “Show All” tab; click the “Ubuntu 14_04 w Docker CE” image.



- Name your virtual machine something simple such as “tutorial” and select an appropriate instance size, such as “medium3 (CPU: 4, Mem: 32GB, Disk: 240GB)”.
- Leave rest of the fields as default.



- Wait for it to become active
- It will now be booting up! This will take 2-10 minutes. Just wait! Don't reload or do anything.

Atmosphere

Secure | <https://atmo.cyverse.org/application/projects/6727/resources>

CYVERSE

Dashboard Projects Images Help

RESOURCES DETAILS OPTIONS

Astrocontainers

NEW Refresh Info

Instances

Name	Status	Activity	IP Address	Size	Provider
tutorial	Active	N/A	128.196.142.33	Medium3	CyVerse Cloud - Marana

Volumes

You have not added any volumes to this project.

Images

You have not added any images to this project.

Links

You have not added any links to this project.

©2018 CyVerse FEEDBACK & SUPPORT

- One the virtual machine is ready, the “Status” column will turn green and described as “Active”.
- Click on your new instance’s name to get more information!
- Now, you can either click “Open Web Shell”, *or*, if you know how to use ssh, you can ssh in with your CyVerse username on the IP address of the machine

The screenshot shows the CyVerse Atmosphere web interface. The browser address bar displays <https://atmo.cyverse.org/application/projects/6727/instances/37950>. The interface has a blue header with the CyVerse logo and navigation links: Dashboard, Projects, Images, and Help. The main content area is divided into two columns. The left column contains the 'Allocation Source' (ckc), 'Allocation Used' (9% of 2048 AUs from ckc), and 'Instance Details' table. The right column contains a sidebar with actions: Report, Reboot, Redeploy, Delete (highlighted in red), and Links. The 'Links' section includes 'Open Old Web Shell' and 'Open Web Shell' (circled in red). The footer shows '©2018 CyVerse' and a 'FEEDBACK & SUPPORT' button.

Allocation Source

ckc

Allocation Used
9% of 2048 AUs from ckc

Instance Details

Status	Active
Activity	N/A
Size	medium3 (CPU: 4, Mem: 32 GB, Disk: 240 GB root)
IP Address	128.196.142.33 Copy
Launched	May 5, 2018 (31 minutes ago)
Based on	Ubuntu 14_04 w Docker CE v1.0
Provider	CyVerse Cloud - Marana

Actions: Report, Reboot, Redeploy, Delete, Links

Links: Open Old Web Shell, Open Web Shell

©2018 CyVerse [FEEDBACK & SUPPORT](#)

4. Deleting your instance

- To completely remove your instance, you can select the “Delete” button from the instance details page.
- This will open up a dialogue window. Select the “Yes, delete this instance” button.

The screenshot shows the Atmosphere web interface for managing CyVerse instances. The main content area displays details for an instance with ID 37950. The 'Allocation Source' is set to 'ckc'. The 'Allocation Used' is 9% of 2048 AUs. The 'Instance Details' table shows the instance is 'Active', has a 'medium3' size, and was launched on May 5, 2018. The right-hand sidebar contains a list of actions: 'Report', 'Reboot', 'Redeploy', and 'Delete'. The 'Delete' button is highlighted with a red circle. Below the actions are links to 'Open Old Web Shell' and 'Open Web Shell'. The footer includes the copyright notice '©2018 CyVerse' and a 'FEEDBACK & SUPPORT' button.

Atmosphere

Secure | <https://atmo.cyverse.org/application/projects/6727/instances/37950>

CYVERSE

Dashboard Projects Images Help

Allocation Source

ckc

Allocation Used

9% of 2048 AUs from ckc

Instance Details

Status	Active
Activity	N/A
Size	medium3 (CPU: 4, Mem: 32 GB, Disk: 240 GB root)
IP Address	128.196.142.33 Copy
Launched	May 5, 2018 (31 minutes ago)
Based on	Ubuntu 14_04 w Docker CE v1.0
Provider	CyVerse Cloud - Marana

Report

Reboot

Redeploy

Delete

Links

Open Old Web Shell

Open Web Shell

©2018 CyVerse

FEEDBACK & SUPPORT

- It may take Atmosphere a few minutes to process your request. The instance should disappear from the project when it has been successfully deleted.

The screenshot shows a web browser window with the URL `https://atmo.cyverse.org/application/projects/6727/resources`. The page has a blue header with the CyVerse logo and navigation links: Dashboard, Projects (active), Images, and Help. Below the header, there are tabs for RESOURCES (active) and DETAILS, and an OPTIONS menu. The main content area is titled 'Astrocontainers' and contains three buttons: 'NEW', a refresh icon, and an information icon. Below these buttons are four sections: 'Instances', 'Volumes', 'Images', and 'Links'. Each section has a message stating 'You have not added any [resource] to this project.' The footer contains the copyright notice '©2018 CyVerse' and a 'FEEDBACK & SUPPORT' button.

Atmosphere

Secure | <https://atmo.cyverse.org/application/projects/6727/resources>

CYVERSE

Dashboard Projects Images Help

RESOURCES DETAILS OPTIONS

Astrocontainers

NEW

Instances

You have not added any instances to this project.

Volumes

You have not added any volumes to this project.

Images

You have not added any images to this project.

Links

You have not added any links to this project.

©2018 CyVerse

FEEDBACK & SUPPORT

Note: It is advisable to delete the machine if you are not planning to use it in future to save valuable resources. However if you want to use it in future, you can suspend it.

CHAPTER 13

Docker related resources

[Awesome Docker](#)

[Docker labs](#)

[Docker Community Slack](#)

[Docker Community Forums](#)

[Docker hub](#)

[Docker documentation](#)

[Docker on StackOverflow](#)

[Docker on Twitter](#)

[Play With Docker Hands-On Labs](#)

[Docker tips](#)

[Docker cloud](#)

[Docker store](#)

Interesting tutorials and blog posts:

1. [Docker Blog](#)
2. [A beginner friendly intro to VMs and Docker](#)
3. [Intro to Docker from Neurohackweek](#)
4. [Understanding Images](#)

Singularity related resources

[Singularity Homepage](#)

[Singularity Hub](#)

[University of Arizona Singularity Tutorials](#)

[NIH HPC](#)

[Dolmades - Windows Apps in Linux Docker-Singularity Containers](#) *Warning not tested*

14.1 Singularity Talks

Gregory Kurtzer, creator of Singularity has provided two good talks online: [Introduction to Singularity](#), and [Advanced Singularity](#).

Vanessa Sochat, lead developer of Singularity Hub, also has given a great talk on [Singularity](#) which you can see online.

CHAPTER 15

Other resources

University of Arizona Campus Resources

- [UA Campus Accessibility](#)
- [UA Campus Transportation](#)
- [Family Spaces and Lactation Support](#)
- [BIO5 Institute](#)
- [Transportation beyond BIO5 and UA campus](#)

CHAPTER 16

For instructors!

Coordinating Web site work

Please create a pull request as soon as you start editing something, rather than waiting! That way you can tell others what you're working on.

You could/should also mention it on Slack in the “cc-leads” channel.

Technical info re adding content to the Web site

All the AstroContainers Workshop tutorials are stored on [GitHub](#).

We will use [GitHub Flow](#) for updates: from the command line,

- fork astrocontainers repository;
- edit, change, add, etc;
- submit a PR;
- when ready to review & merge say ‘ready for review & merge @ac2018’.

It's important that all updates go through code review by someone. Anyone with push access to the repo can review and merge!

From the Web site, you should be able to edit the files and then set up a PR directly. You can also fork the repo, perform multiple edits and submit a PR through the web interface.

Updating the “official” Web site.

The [Web site](#), will update automatically from GitHub. However, it may take 5-15 minutes to do so.

Building a local copy of the Web site.

Briefly,

- clone the repo:

```
git clone https://github.com/CyVerse-learning-materials/astrocontainers_workshop_2018.git
```

- set up a virtualenv with python2 or python3:

```
python -m virtualenv buildenv -p python3.5; . ~/buildenv/bin/activate
```

- install the prerequisites:

```
pip install -r requirements.txt
```

- build site:

```
make html
```

- open / click on

```
_build/html/index.html
```

Formatting, guidelines, etc.

Everything can/should be in [Restructured text](#) If you're not super familiar with Restructured text, you can use [online restructured text editor](#) to write your tutorials.

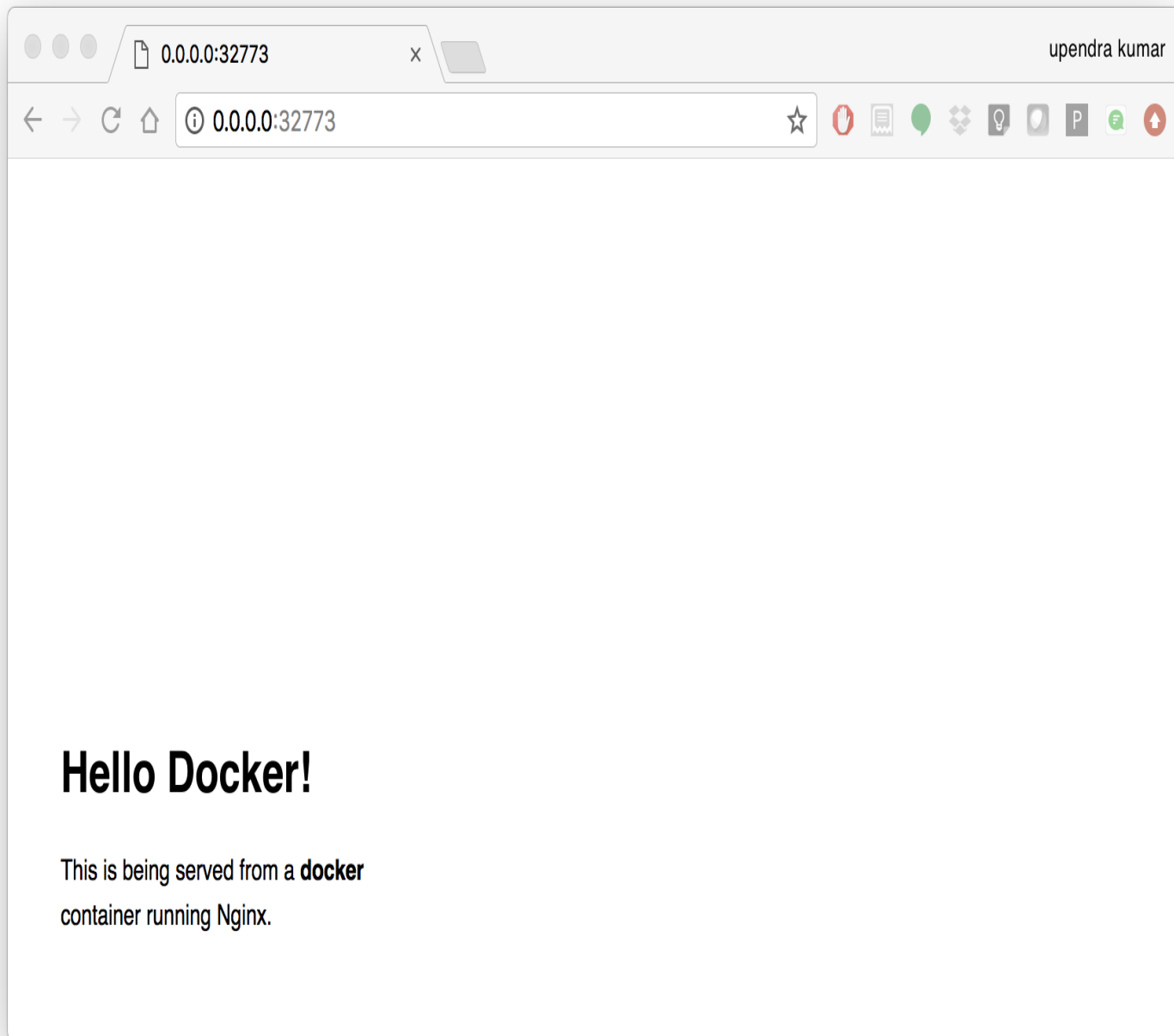
(Note that you can go visit the github repo and it will helpfully render *.rst* files for you if you click on them! They just won't have the full site template.)

Files and images that don't need to be "compiled" and should just be served up through the web site can be put in the *_static* directory; their URL will then be

https://cyverse-container-camp-workshop-2018.readthedocs-hosted.com/_static/filename

Images

Image formatting in Restructured text is pretty straightforward. Here is an example



```
.. |static_site_docker| image:: ../img/static_site_docker.png
```

```
:width: 750
```

```
:height: 700
```

Problems? Bugs? Questions?

- If there is a bug and you can fix it: submit a PR. Make sure that I know who you are so that I can thank you.
- If there is a bug and you can't fix it, but you can reproduce it: submit an issue explaining how to reproduce.
- If there is a bug and you can't even reproduce it: sorry. It is probably an Heisenbug. We can't act on it until it's reproducible, alas.
- If you have attended this workshop and have feedback, or if you want somebody to deliver that workshop at your conference or for your company: you can contact one of us!

upendra at cyverse dot org

Thank you!