CyVerse Documentation

CyVerse

Sep 18, 2020

Getting Started

1 Workshop Code of Conduct and Other Information 3
2 Pre-Workshop Setup 5
3 Agenda 7
4 About CyVerse 9
5 Training session in Docker 11
6 Training session in Singularity 13
7 Breakout sessions 15
8 Introduction to Docker 17
9 Advanced Docker 33
10 Docker hands-on exercises 57
11 Introduction to Singularity 63
12 Advanced Singularity 73
13 Setting up Singularity file system 77
14 Biocontainers 83
15 NVIDIA-Docker 85
16 Containerized Workflows 89
17 Docker for Datascience 91
18 Booting a CyVerse Atmosphere instance 101
19 Deploying apps in CyVerse Discovery Environment 109
20 Deploying interactive apps in CyVerse Discovery Environment 119

21 Docker related resources 129

22 Singularity related resources 131

23 Other resources 133

24 For instructors! 135

25 Problems? Bugs? Questions? 139

CYVERSE® CONTAINER CAMP
How to Build & Deploy Containerized Workflows
March 6-8 2019
University of Arizona, Tucson

Container technologies are letting researchers easily share, scale, and reuse tools and workflows for all types of computational analyses. CyVerse Container Camp is an intensive three day hands-on workshop to learn how to create, us, and deploy containers across a variety of compute systems (your computer, local HPC cloud compute environments, and national resources such as OSG).

In this 3-day workshop, users will blend practical theory and hands-on exercises where small groups deploy tools and workflows they bring to the workshop.

- How to containerize applications and workflows
- How to use other containerized applications and workflows
- How to build/deploy containerized applications and workflows
- How to scale out your computation from laptop to cloud to HPC/OSG

CHAPTER 1

Workshop Code of Conduct and Other Information

All attendees, speakers, staff and volunteers at Container Camp are required to follow our code of conduct.

CyVerse expects and appreciates cooperation from all participants to help ensure a safe, collaborative environment for everyone. Harrassment by any individual will not be tolerated and may result in the individual being removed from the Camp.

Harassment includes: offensive verbal comments related to gender, gender identity and expression, age, sexual orientation, disability, physical appearance, body size, race, ethnicity, religion, technology choices, sexual images in public spaces, deliberate intimidation, stalking, following, harassing photography or recording, sustained disruption of talks or other events, inappropriate physical contact, and unwelcome sexual attention.

Participants who are asked to stop any harassing behavior are expected to comply immediately.

Workshop staff are also subject to the anti-harassment policy. In particular, staff should not use sexualised images, activities, or other material.

If a participant engages in harassing behavior, the workshop organisers may take any action they deem appropriate, including warning the offender or expulsion from the workshop with no refund.

If you are being harassed, or notice that someone else is being harassed, or have any other concerns, please contact a member of the workshop staff immediately. Staff can be identified as they'll be wearing badges or nametags.

Workshop staff will be happy to help participants contact local law enforcement, provide escorts, or otherwise assist those experiencing harassment to feel safe for the duration of the workshop. We value your attendance.

We expect participants to follow these rules at conference and workshop venues and conference-related social events.

See http://www.ashedryden.com/blog/codes-of-conduct-101-faq for more information on codes of conduct.

FAIR Data principles

Container Camp 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, enabling interoperability and reusability of data.

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.

Container Camp Goes Green

This year, we encourage all Container Camp participants to help minimize our waste footprint. If possible, bring your own reusable beverage containers, such as coffee mugs and water bottles, to use during snack breaks. Thanks in advance!

Who should attend?

Faculty, researchers, postdocs, and graduate students who use and analyze data of all types (genomics, astronomy, image data, Big Data, 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 also talk to any of the instructors or TAs if you need immediate help.
- Chat with us 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 Container Camp workshop at CyVerse, The University of Arizona, which will run from March 6-8, 2019.

Prerequisite	Notes	Additional notes
Wi-Fi-enabled laptop	You should be able to use any laptop (Windows/MacOS/Linux.). We **strongly recommend** Firefox or Chrome browser. It is recommended that you have administrative/install permissions on your laptop.	• 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 https://user.cyverse.org/.
Github Account	Please ensure that you have a Github account if you don't have one already	Register for your Github account at https://github.com/.
Dockerhub Account	Please ensure that you have a Dockerhub account if you don't have one already	Register for your Dockerhub account at https://hub.docker.com/.
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	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 Downloads

Listed below are some extra downloads that are not 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	• 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.	• 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 Container Camp at The University of Arizona, which will run from March 6th to 8th, 2019. Container Camp will be held in U Arizona's Drachman Hall, Rm A116. You can find a map of that location here. For nearest parking garage, it is the Highland Street Garage, where daily parking is $8/day; it's about a 2 block walk (east) to Drachman from the garage.

This workshop runs under a Code of Conduct. Please respect it and be excellent to each other!

Twitter hash tag: #cc2019

Day	Time	Topic/Activity	Notes/Links
03/06/19 (Wednesday)	8:30-9:00	General introduction to CyVerse and Camp logistics (Nirav Merchant & Upendra Devisetty)	Intro slides
	9:00-9:30	General overview of container technology landscape (Nirav Merchant)	Container technology
	10:15-10:30	Coffee and snack break with networking	Served in A127-29 across the hall (pls no food/bev in A116)
	10:30-12:00	Introduction to Docker (Julian Pistorius)	Introduction to Docker
	12:00-1:00	Lunch break (on your own)	
	1:00-3:00	Advanced Docker hands on (Tyson Swetnam)	Advanced Docker
	3:00-3:30	Coffee and snack break with networking	Served in A127-29 across the hall (pls no food/bev in A116)
	3:30-5:00	Docker hands-on exercises (CK Chan)	Docker hands-on exercises
	5:00-6:00	Debriefing with instructors	
03/07/19 (Thursday)	8:30-9:00	Containers in Astronomy: Presentation by CK Chan	Slides
	9:00-9:30	Review Day 1 (Questions, Comments, suggestions etc.,)	
	9:30-12:00	Project pitches (1 min) and BYOD/BYOA	Coffee and snacks will be served in A127-29 across the hall (pls no food/bev in A116)
	12:00-1:00	Lunch Break (on your own)	
	1:00-1:30	Deploying your containers in DE (Upendra Devisetty)	• Non-interactive apps in DE • Interactive apps in DE
	1:30-4:00	Breakout sessions	Breakout sessions
	4:00-5:00	Optional: Mirror Lab tour or UA HPC Facility tour	• https://mirrorlab.arizona.edu • https://it.arizona.edu/service/high-performance-computing
	5:00-6:00	Debriefing with instructors	
03/08/19 (Friday)	8:30-9:00	Review Day 2 (Questions, Comments)	
	9:00-10:00	500k containers a day - Remote Presentation (Mats Rynge)	Slides
	10:00-10:30	Coffee and snacks with networking	Served in A127-29 across the hall (pls no food/bev in A116)
	10:30-12:00	Introduction to Singular-	

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.

Originally created as the iPlant Collaborative to serve U.S. plant science communities, the cyberinfrastructure we have built is germane to all life sciences disciplines and works equally well on data from plants, animals, or microbes. Thus, iPlant was renamed CyVerse to reflect the broader community now served by our infrastructure. 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, DBI-1265383 and DBI-1743442.

Please cite CyVerse appropriately when you make use of our resources, CyVerse citation policy

CHAPTER 5

Training session in Docker

In this session we will explain the various aspects of the Docker. Starting with the basics of Docker which focus on the installation and configuration of Docker, the session will gradually move to advanced topics such as managing data using volumes and pushing and pull containers from registries. Overall this session covers the development aspects of Docker and how you can get up and running on the development environments using Docker containers.

- Docker Introduction

This is the introductory session for the concept of Docker. The topics include Docker installation, running prebuilt Docker containers, deploying web applications with Docker, building and running your own Docker containers, etc.

- Advanced Docker

This is the advanced session for the concept of Docker. The topics include pushing and pulling Docker containers to public and private registries, automated Docker image building from github/bitbucket repositories, managing data in Docker containers, Docker compose for building multiple Docker containers, improving your data science workflows using Docker containers, etc.

CHAPTER 6

Training session in Singularity

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 Introduction

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

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.

CHAPTER 7

Breakout sessions

1. **Biocontainers** - Lead: Amanda Cooksey, CyVerse Scientific Analyst

   Biocontainer breakout session content

   In this breakout session, you'll learn about Biocontainers and apply what you've learned about basic container technology, such as Docker, with open source bioinformatics apps for Proteomics, Genomics, Transcriptomics, and Metabolomics.

2. **NVIDIA-Docker** - Lead: Tyson Swetnam, CyVerse Scientific Analyst

   nvidia docker breakout session content

   In this breakout session, you'll learn about NVIDIA GPU Cloud, and how to use NVIDIA-based containers in Docker and Singularity. NVIDIA GPUs are widely used for 3D modelling and Machine Learning applications. We will discuss the complexities of working with these different container types, with hands on examples running CyVerse and HPC.

3. **Containerized workflows** - Lead: Sateesh Peri, CyVerse power user, University of Nevada-Reno

   Containerized workflows breakout session content

   In this breakout session you'll learn about Snakemake, a workflow management system consisting of a text-based workflow specification language and a scalable execution environment. You will be introduced to the Snakemake workflow definition language and how to use the execution environment to scale workflows to compute servers and clusters while adapting to hardware specific constraints.

4. **Docker for Data Science** - Lead: Upendra Devisetty, CyVerse Scientific Analyst

   Docker for Data Science breakout session content

   Why do Data Scientists care about Docker? For a long time statisticians and Data Scientists have ignored the software aspect of data analysis. In this breakout session, you'll learn what's in Docker for Data Scientists. In particular how Docker helps Data Scientists to save time, enable reproducible research and distributing their environment to the outside world.

CHAPTER 8

Introduction to Docker

8.1 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 developing web applications will be helpful but is not required.

8.2 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.

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

8.2.1 2.1 Testing Docker installation

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 18.09.3, build 774a1f4
```

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.
.......
```

8.3 3. Running Docker containers from prebuilt images

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`. 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          1.
↪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 --all`

```
$ docker ps --all
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 --latest
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 --latest` command again:

```
$ docker ps --latest
CONTAINER ID        IMAGE                 COMMAND                  CREATED             ␣
↪ STATUS                       PORTS                   NAMES
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
```

8.4 4. Build Docker images which contain your own code

Great! so you have now looked at `docker run`, played with a Docker containers and also got the hang of some terminology. Armed with all this knowledge, you are now ready to get to the real stuff — deploying your own applications with Docker.

8.4.1 4.1 Deploying a command-line app

Note: Code for this section is in this repo in the examples/ directory

In this section, 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.

4.1.1 Docker images

Docker images are the basis of containers. In the previous example, you pulled the `alpine` 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
ubuntu                     bionic              47b19964fb50        4 weeks ago            ␣
↪ 88.1MB
alpine                     latest              caf27325b298        4 weeks ago            ␣
↪ 5.53MB
hello-world                latest              fce289e99eb9        2 months ago           ␣
↪ 1.84kB
.........
```

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                                                       DESCRIPTION                     ␣
↪                    STARS                OFFICIAL               AUTOMATED
 ubuntu                                                     Ubuntu is a Debian-based␣
↪Linux operating sys...   7310                  [OK]
 dorowu/ubuntu-desktop-lxde-vnc                             Ubuntu with openssh-server␣
↪and NoVNC              163                                       [OK]
 rastasheep/ubuntu-sshd                                     Dockerized SSH service,␣
↪built on top of offi...   131                                         [OK]
 ansible/ubuntu14.04-ansible                                Ubuntu 14.04 LTS with␣
↪ansible                   90                                          [OK]
 ubuntu-upstart                                             Upstart is an event-based␣
↪replacement for th...   81                  [OK]
 neurodebian                                                NeuroDebian provides␣
↪neuroscience research s...   43                  [OK]
 ubuntu-debootstrap                                         debootstrap --
↪variant=minbase --components=m...   35                 [OK]
 1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5   ubuntu-16-nginx-php-
↪phpmyadmin-mysql-5          26                                          [OK]
 nuagebec/ubuntu                                            Simple always updated Ubuntu␣
↪docker images w...   22                                        [OK]
 tutum/ubuntu                                               Simple Ubuntu docker images␣
↪with SSH access      18
 ppc64le/ubuntu                                             Ubuntu is a Debian-based␣
↪Linux operating sys...   11
 i386/ubuntu                                                Ubuntu is a Debian-based␣
↪Linux operating sys...   9
 1and1internet/ubuntu-16-apache-php-7.0                     ubuntu-16-apache-php-7.0    ␣
↪                    7                                          [OK]
 eclipse/ubuntu_jdk8                                        Ubuntu, JDK8, Maven 3, git,␣
↪curl, nmap, mc, ...   5                                           [OK]
 darksheer/ubuntu                                           Base Ubuntu Image -- Updated␣
↪hourly              3                                          [OK]
 codenvy/ubuntu_jdk8                                        Ubuntu, JDK8, Maven 3, git,␣
↪curl, nmap, mc, ...   3                                           [OK]
 1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4         ubuntu-16-nginx-php-5.6-
↪wordpress-4            2                                          [OK]
 1and1internet/ubuntu-16-nginx                              ubuntu-16-nginx             ␣
↪                    2                                          [OK]
 pivotaldata/ubuntu                                         A quick freshening-up of the␣
↪base Ubuntu doc...   1
 smartentry/ubuntu                                          ubuntu with smartentry      ␣
↪                    0                                          [OK]
 pivotaldata/ubuntu-gpdb-dev                                Ubuntu images for GPDB␣
↪development               0
 1and1internet/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
 1and1internet/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.

4.1.2 Meet our Python app

Now that you have a better understanding of images, it's time to create an image that sandboxes a small Python application. We'll do this by creating a small Python script which prints a welcome message, then dockerizing it by writing a Dockerfile, and finally we'll build the image and run it.

- Create a Python script
- Build the image
- Run your image

1. Create a Python script which prints a welcome message

Start by creating a directory called `simple-script` in your home directory where we'll create the following files:

- `app.py`
- `Dockerfile`

```
$ cd ~ && mkdir simple-script && cd simple-script
```

1.1 **app.py**

Create the `app.py` file with the following content. You can use any of favorite text editor to create this file.

```
print('hello world!')
print('this is my first attempt')
```

Note: If you want, you can run this app through your laptop's native Python installation first just to see what it looks like. Run `python app.py`.

You should see the message:

`hello world! this is my first attempt`

This is totally optional - but some people like to see what the app's supposed to do before they try to Dockerize it.

1.2. **Dockerfile**

A **Dockerfile** is a text file that contains a list of commands that the Docker daemon calls while creating an image. The Dockerfile contains all the information that Docker needs to know to run the app — a base Docker image to run from, location of your project code, any dependencies it has, and what commands to run at start-up. It is a simple way to automate the image creation process. The best part is that the commands you write in a Dockerfile are almost identical to their equivalent Linux commands. This means you don't really have to learn new syntax to create your own Dockerfiles.

We want to create a Docker image with this app. As mentioned above, all user images are based on a base image. Since our application is written in Python, we will build our own Python image based on `Alpine`. We'll do that using a Dockerfile.

Create a file called Dockerfile in the `simple-script` directory, and add content to it as described below.

```
# our base image# our base image
FROM alpine:3.9

# install python and pip
RUN apk add --update py3-pip

# copy files required for the app to run
COPY app.py /usr/src/app/

# run the application
CMD python3 /usr/src/app/app.py
```

Now let's see what each of those lines mean..

1.2.1 We'll start by specifying our base image, using the FROM keyword:

```
FROM alpine:3.9
```

1.2.2. The next step usually is to write the commands of copying the files and installing the dependencies. But first we will install the Python pip package to the alpine linux distribution. This will not just install the pip package but any other dependencies too, which includes the python interpreter. Add the following `RUN` command next:

```
RUN apk add --update py3-pip
```

1.2.3. Copy the file you have created earlier into our image by using `COPY` command.

```
COPY app.py /usr/src/app/
```

1.2.4. The last step is the command for running the application. Use the `CMD` command to do that:

```
CMD python3 /usr/src/app/app.py
```

The primary purpose of `CMD` is to tell the container which command it should run by default when it is started.

2. Build the image

Now that you have your Dockerfile, you can build your image. The `docker build` command does the heavy-lifting of creating a docker image from a Dockerfile.

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:

Note: When you run the `docker build` command given below, make sure to replace `<YOUR_DOCKERHUB_USERNAME>` with your username. This username should be the same one you created when registering on Docker hub. If you haven't done that yet, please go ahead and create an account in Dockerhub.

```
YOUR_DOCKERHUB_USERNAME=<YOUR_DOCKERHUB_USERNAME>
```

For example this is how I assign my dockerhub username

```
YOUR_DOCKERHUB_USERNAME=jpistorius
```

Now build the image using the following command:

```
$ docker build -t $YOUR_DOCKERHUB_USERNAME/simple-script .
Sending build context to Docker daemon  10.24kB
Step 1/4 : FROM alpine:3.9
 ---> caf27325b298
Step 2/4 : RUN apk add --update py3-pip
 ---> Using cache
 ---> dad2a197fcad
Step 3/4 : COPY app.py /usr/src/app/
 ---> Using cache
 ---> a8ebf6cd2735
Step 4/4 : CMD python3 /usr/src/app/app.py
 ---> Using cache
 ---> a1fb2906a937
Successfully built a1fb2906a937
Successfully tagged jpistorius/simple-script:latest
```

If you don't have the `alpine:3.9 image`, the client will first pull the image and then create your image. Therefore, your output on running the command will look different from mine. If everything went well, your image should be ready! Run `docker images` and see if your image `$YOUR_DOCKERHUB_USERNAME/simple-script` shows.

3. Run your image

When Docker can successfully build your Dockerfile, test it by starting a new container from your new image using the docker run command.

```
$ docker run $YOUR_DOCKERHUB_USERNAME/simple-script
```

You should see something like this:

```
hello world!
this is my first attempt
```

8.4.2 4.2 Deploying a Jupyter Notebook

In this section, let's build a Docker image which can run a Jupyter Notebook

4.2.1 Suitable Docker images for a base

Search for images on Docker Hub which contain the string 'jupyter'

```
$ docker search jupyter
NAME                                       DESCRIPTION                                     ␣
↪ STARS               OFFICIAL             AUTOMATED
jupyter/datascience-notebook               Jupyter Notebook Data Science Stack from␣
↪htt...   446
jupyter/all-spark-notebook                 Jupyter Notebook Python, Scala, R, Spark,␣
↪Me...   223
jupyterhub/jupyterhub                      JupyterHub: multi-user Jupyter notebook␣
↪serv...   195                                          [OK]
jupyter/scipy-notebook                     Jupyter Notebook Scientific Python Stack␣
↪fro...   155
```

(continues on next page)

(continued from previous page)

```
jupyter/tensorflow-notebook                 Jupyter Notebook Scientific Python Stack w/␣
↪...   116
jupyter/pyspark-notebook                    Jupyter Notebook Python, Spark, Mesos Stack␣
↪...   95
jupyter/minimal-notebook                    Minimal Jupyter Notebook Stack from https://
↪...   73
ermaker/keras-jupyter                       Jupyter with Keras (with Theano backend and␣
↪...   66                                            [OK]
jupyter/base-notebook                       Small base image for Jupyter Notebook␣
↪stacks...   60
xblaster/tensorflow-jupyter                 Dockerized Jupyter with tensorflow                ␣
↪ 52                                             [OK]
jupyter/r-notebook                          Jupyter Notebook R Stack from https://
↪github...   22
jupyterhub/singleuser                       single-user docker images for use with␣
↪Jupyt...   21                                             [OK]
...
```

4.2.2 Meet our model

Let's deploy a Python function inside a Docker image along with Jupyter.

- *Create a Python file containing a function*
- **'Build the image'_**
- **'Run your image'_**

1. Create a Python file containing a function

Start by creating a directory called `mynotebook` in your home directory where we'll create the following files:

- model.py
- Dockerfile

```
$ cd ~ && mkdir mynotebook && cd mynotebook
```

1.1 **model.py**

Create the `model.py` file with the following content. You can use any of favorite text editor to create this file.

```
def introduce(name):
    return 'Hello ' + name
```

1.2. **Dockerfile**

Since we want to use a Jupyter notebook to call our function, we will build an image based on `jupyter/minimal-notebook`.

Note: This is one of the official Docker images provided by the Jupyter project for you to build your own data science notebooks on:

https://jupyter-docker-stacks.readthedocs.io/en/latest/

Create a file called Dockerfile in the `mynotebook` directory, and add content to it as described below.

```
# our base image
FROM jupyter/minimal-notebook

# copy files required for the model to work
COPY model.py /home/jovyan/work/

# tell the port number the container should expose
EXPOSE 8888
```

Now let's see what each of those lines mean..

1.2.1 We'll start by specifying our base image, using the FROM keyword:

```
FROM jupyter/minimal-notebook
```

1.2.2. Copy the file you have created earlier into our image by using `COPY` command.

```
COPY model.py /home/jovyan/work/
```

1.2.3. Specify the port number which needs to be exposed. Since Jupyter runs on 8888 that's what we'll expose.

```
EXPOSE 8888
```

1.2.4. What about `CMD`?

Notice that unlike our previous Dockerfile this one does not end with a `CMD` command. This is on purpose.

Remember: The primary purpose of `CMD` is to tell the container which command it should run by default when it is started.

Can you guess what will happen if we don't specify our own 'entrypoint' using `CMD`?

2. Build the image

Note: Remember to replace `<YOUR_DOCKERHUB_USERNAME>` with your username. This username should be the same one you created when registering on Docker hub.

```
YOUR_DOCKERHUB_USERNAME=<YOUR_DOCKERHUB_USERNAME>
```

For example this is how I assign my dockerhub username

```
YOUR_DOCKERHUB_USERNAME=jpistorius
```

Now build the image using the following command:

```
$ docker build -t $YOUR_DOCKERHUB_USERNAME/mynotebook .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM jupyter/minimal-notebook
 ---> 36c8dd0e1d8f
Step 2/3 : COPY model.py /home/jovyan/work/
 ---> b61aefd7a735
Step 3/3 : EXPOSE 8888
 ---> Running in 519dcabe4eb3
Removing intermediate container 519dcabe4eb3
 ---> 7983fe164dc6
Successfully built 7983fe164dc6
Successfully tagged jpistorius/mynotebook:latest
```

If everything went well, your image should be ready! Run `docker images` and see if your image `$YOUR_DOCKERHUB_USERNAME/mynotebook` shows.

3. Run your image

When Docker can successfully build your Dockerfile, test it by starting a new container from your new image using the docker run command. Don't forget to include the port forwarding options you learned about before.

```
$ docker run -p 8888:8888 $YOUR_DOCKERHUB_USERNAME/mynotebook
```

You should see something like this:

```
Executing the command: jupyter notebook
[I 07:21:25.396 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.
↪local/share/jupyter/runtime/notebook_cookie_secret
[I 07:21:25.609 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.
↪7/site-packages/jupyterlab
[I 07:21:25.609 NotebookApp] JupyterLab application directory is /opt/conda/share/
↪jupyter/lab
[I 07:21:25.611 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 07:21:25.611 NotebookApp] The Jupyter Notebook is running at:
[I 07:21:25.611 NotebookApp] http://(29a022bb5807 or 127.0.0.1):8888/?token=copy-your-
↪own-token-not-this-one
[I 07:21:25.611 NotebookApp] Use Control-C to stop this server and shut down all␣
↪kernels (twice to skip confirmation).
[C 07:21:25.612 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(29a022bb5807 or 127.0.0.1):8888/?token=copy-your-own-token-not-this-
↪one
```

Head over to http://localhost:8888 and your Jupyter notebook server should be running.

Note: Copy the token from your own `docker run` output and paste it into the 'Password or token' input box.

8.5 5. Dockerfile commands summary

Here's a quick summary of the few basic commands we used in our Dockerfiles.

- **FROM** starts the Dockerfile. It is a requirement that the Dockerfile must start with the FROM command. Images are created in layers, which means you can use another image as the base image for your own. The FROM command defines your base layer. As arguments, it takes the name of the image. Optionally, you can add the Dockerhub username of the maintainer and image version, in the format username/imagename:version.
- **RUN** is used to build up the Image you're creating. For each RUN command, Docker will run the command then create a new layer of the image. This way you can roll back your image to previous states easily. The syntax for a RUN instruction is to place the full text of the shell command after the RUN (e.g., RUN mkdir /user/local/foo). This will automatically run in a /bin/sh shell. You can define a different shell like this: RUN /bin/bash -c 'mkdir /user/local/foo'
- **COPY** copies local files into the container.
- **CMD** 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. So example CMD commands would be:

```
CMD ["python", "./app.py"]

CMD ["/bin/bash", "echo", "Hello World"]
```

- EXPOSE creates a hint for users of an image which ports provide services. It is included in the information which can be retrieved via `$ docker inspect <container-id>`.

Note: The EXPOSE command does not actually make any ports accessible to the host! Instead, this requires publishing ports by means of the `-p` flag when using `docker run`.

- PUSH pushes your image to Docker Cloud, or alternately to a private registry

Note: If you want to learn more about Dockerfiles, check out Best practices for writing Dockerfiles.

8.6 6. Demos

8.6.1 6.1 Portainer

Portainer is an open-source lightweight managment 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.

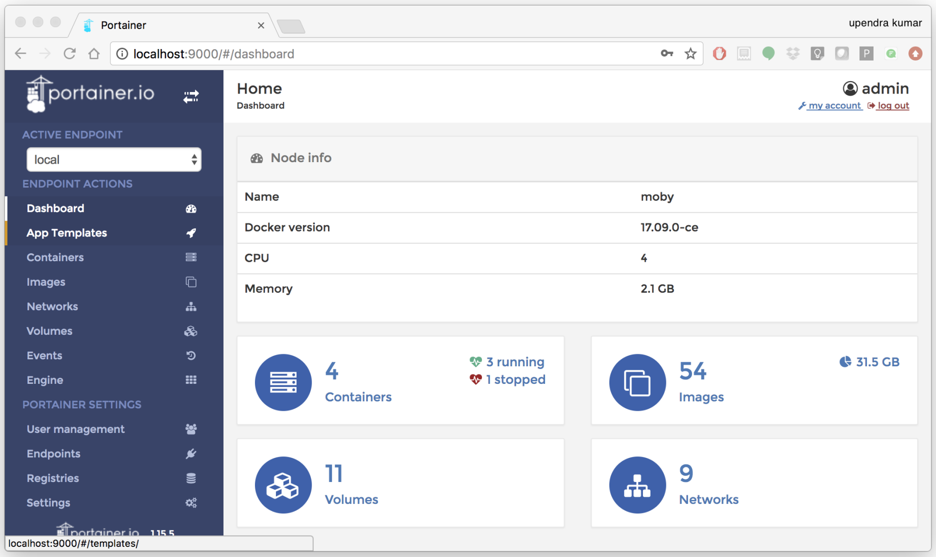
6.1.1 Installation

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.



Name	moby
Docker version	17.09.0-ce
CPU	4
Memory	2.1 GB

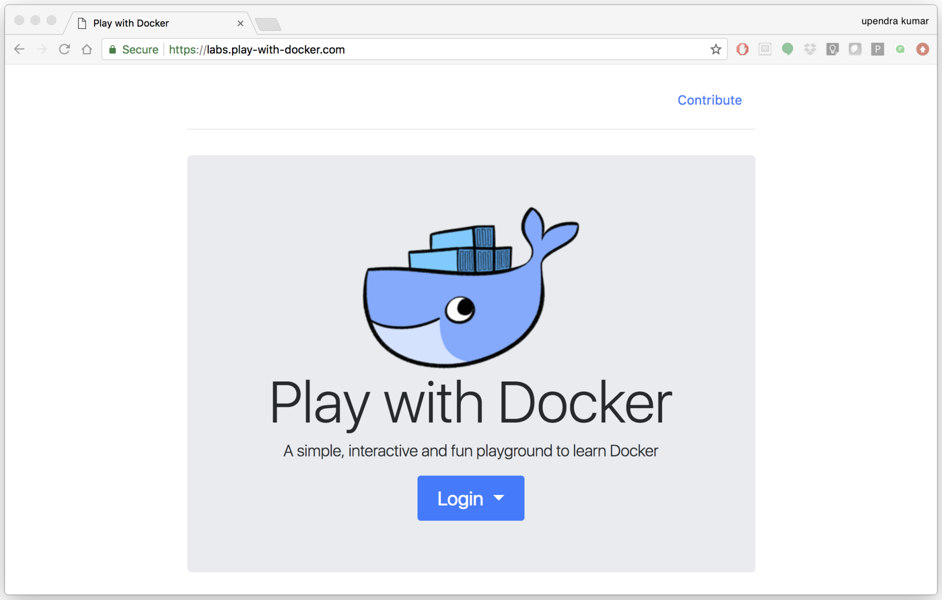
8.6.2 6.2 Play-with-docker (PWD)

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.

6.2.1 Installation

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

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


Play with Docker
upendra kumar
Secure https://labs.play-with-docker.com
Contribute
Play with Docker
A simple, interactive and fun playground to learn Docker
Login

CHAPTER 9

Advanced Docker

Now that we are *relatively* comfortable with Docker, lets look at some advanced Docker topics, such as:

- Registry
- Porting a Docker image to a Registry & Repository (public and private)
- Managing data within containers
- Deploying containers on cloud services

9.1 1. Docker Registries

To demonstrate the portability of what we just created, let's upload our built Docker image to a Docker Registry and then run it somewhere else (i.e. CyVerse Atmosphere).

In this exercise, you'll learn how to push built containers to registries, pull those containers from registries, and run those containers on remote hosts (virtual machines).

This will benefit you when you want to deploy new containers to production environments where testing is not possible.

Important: So what *EXACTLY* is a **Registry**?

A registry is a collection of Repositories, and a Repository is a collection of Images. A Docker Registry is sort of like a GitHub Repository, except the code is already compiled, in this case, into a container. You must have an account on a registry. You 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:

- Push images
- Find images
- Pull images

- Share images

9.1.1 1.1 Popular Registries

Some examples of public/private registries to consider for your research needs:

- Docker Cloud
- Docker Hub
- Docker Trusted Registry
- Amazon Elastic Container Registry
- Google Container Registry
- Azure Container Registry
- NVIDIA GPU Cloud
- Private Docker Registry - not official Docker
- Gitlab Container Registry
- CoreOS Quay
- TreeScale
- Canister

1.1.1 Log into a Registry 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 an 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.

If you want to authenticate to a different Registry, type the name of the registry after `login`:

```
$ docker login <registry-name>
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/tswetnam/.docker/config.
↪json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

If it is your first time logging in you will be queried for your `username` and `password`.

Login with your Docker ID to push and pull images from Docker Hub or private registry.

If you don't have a Docker ID, head over to https://hub.docker.com to create one.

1.1.2 Tagging images

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`

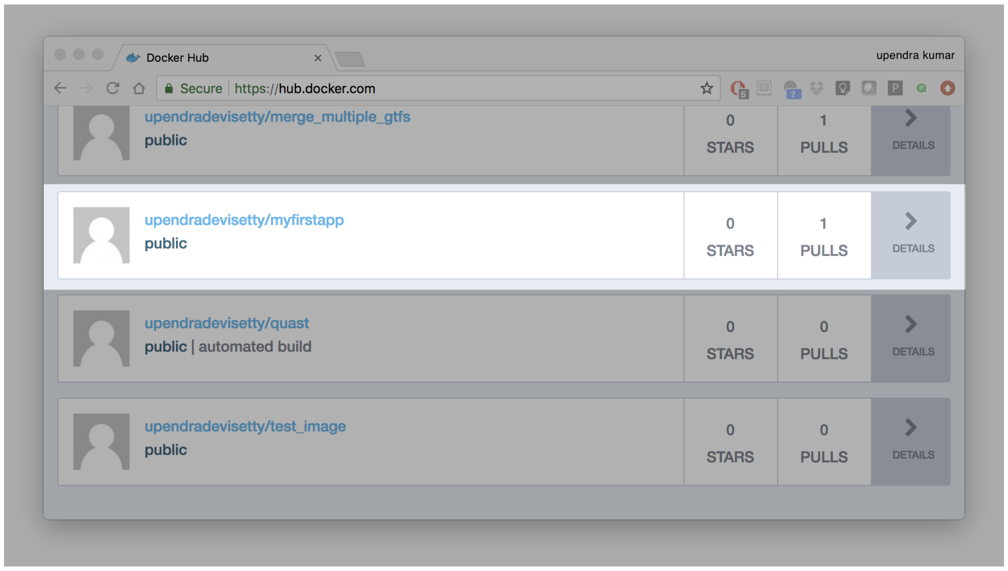
```
$ docker tag $YOUR_DOCKERHUB_USERNAME/mynotebook:latest $YOUR_DOCKERHUB_USERNAME/
↪mynotebook:1.0
```

1.1.3 Publish the image

Upload your tagged image to the Dockerhub repository

```
$ docker push $YOUR_DOCKERHUB_USERNAME/mynotebook: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!

1.1.4 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 -p 8888:8888 --name notebooktest $YOUR_DOCKERHUB_USERNAME/mynotebook: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.

Head over to `http://<vm-address>:8888` and your app should be live.

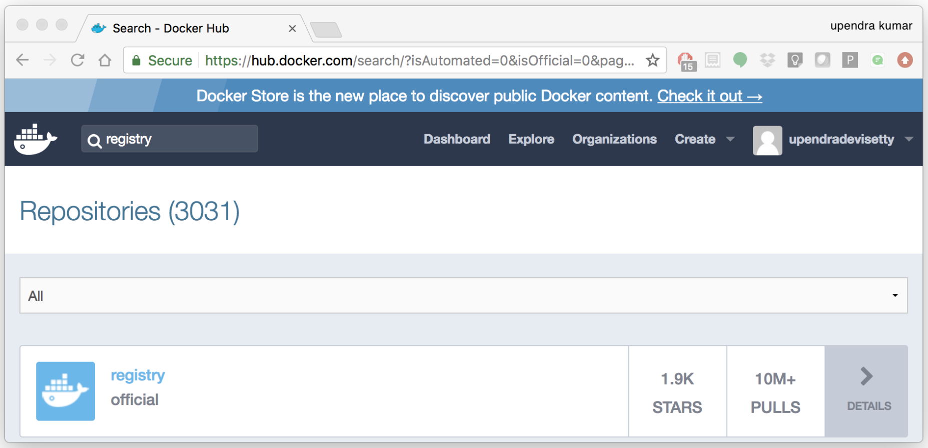
9.1.2 1.2 Private repositories

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 it own private registry which will be used to push the Docker images.

1.2.1 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 --latest` to check the recent container from this Docker image

```
$ docker ps --latest
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
```

1.2.2 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 $YOUR_DOCKERHUB_USERNAME/mynotebook:1.0 $REGISTRY/$(whoami)/mynotebook:1.
↪0
```

1.2.2 Publish the image into the local registry

Finally push the image to the local registry

```
$ docker push $REGISTRY/$(whoami)/mynotebook:1.0
The push refers to a repository [localhost:5000/julianp/mynotebook]
64436820c85c: Pushed
831cff83ec9e: Pushed
c3497b2669a8: Pushed
1c5b16094682: Pushed
c52044a91867: Pushed
60ab55d3379d: Pushed
1.0: digest: sha256:5095dea8b2cf308c5866ef646a0e84d494a00ff0e9b2c8e8313a176424a230ce␣
↪size: 1572
```

1.2.3 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 -P --name=mynotebooklocal $REGISTRY/$(whoami)/mynotebook:1.0
```

9.2 2. 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.

9.2.1 2.1 Prerequisites

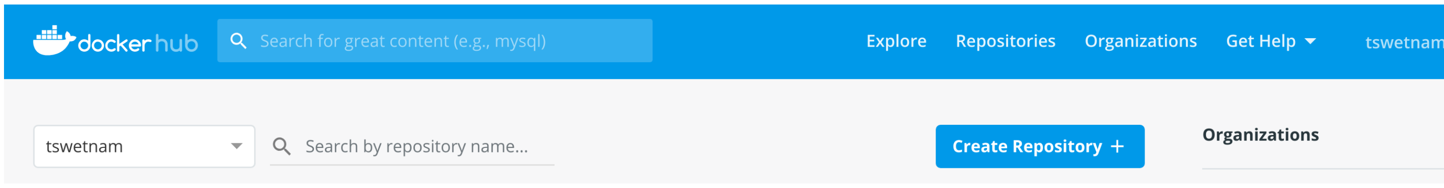
To use automated builds, you first must have an account on Docker Hub and on the hosted repository provider (GitHub or Bitbucket). While Docker Hub supports linking both GitHub and Bitbucket repositories, here we will use a GitHub repository. If you don't already have one, make sure you have a GitHub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

Note:

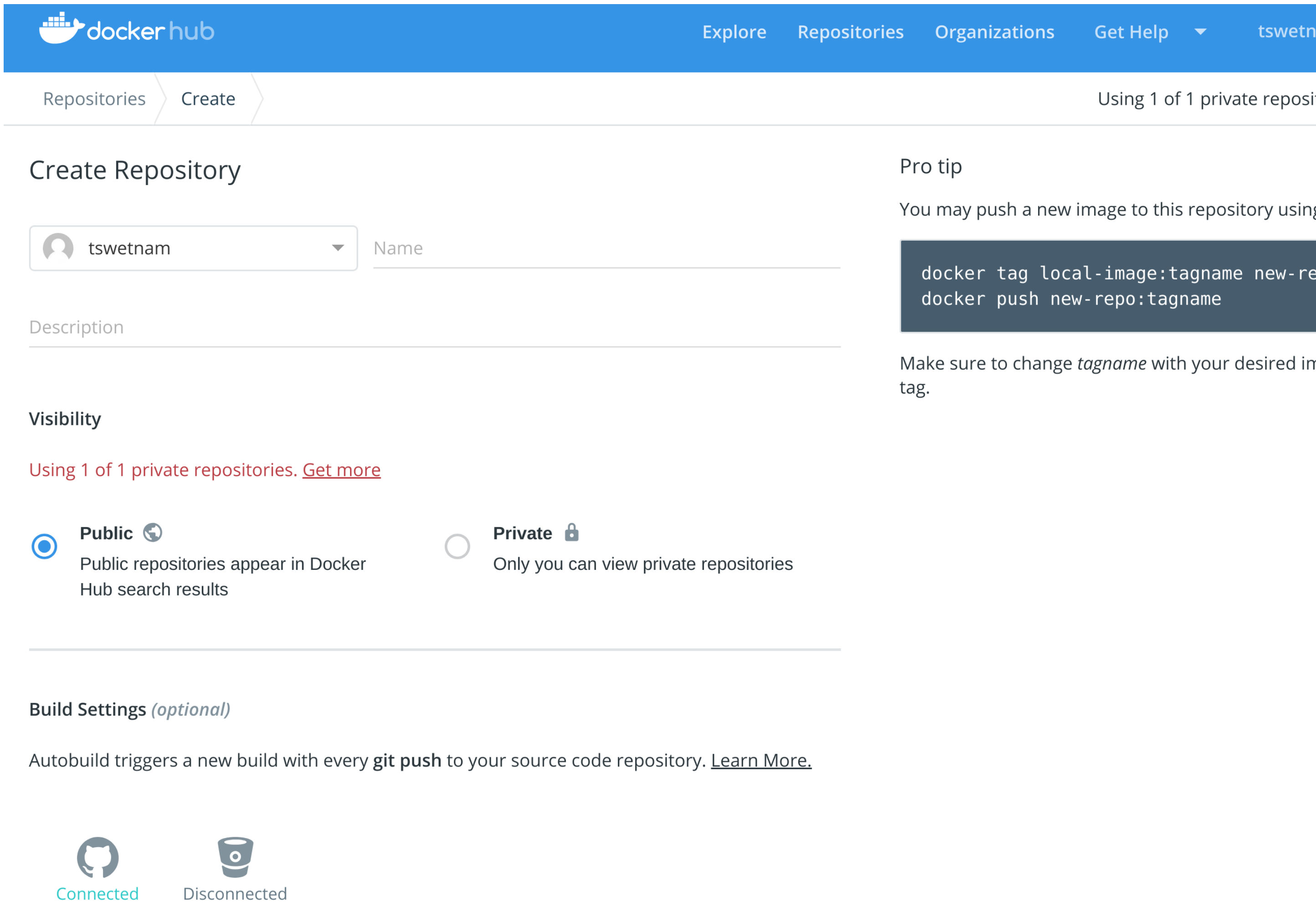
- If you have previously linked your Github or Bitbucket account, you must have chosen the Public and Private connection type. To view your current connection settings, log in to Docker Hub and choose Profile > Settings > Linked Accounts & Services.
- Building Windows containers is not supported.

9.2.2 2.2 Link your Docker Hub account to GitHub

1. Log into Docker Hub.
2. Click "Create Repository+"



3. Click the Build Settings and select `GitHub`.



Create Repository

tswetnam

Name

Description

Visibility

Using 1 of 1 private repositories. Get more

[✓ Public] Public repositories appear in Docker Hub search results

Private Only you can view private repositories

Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository. Learn More.

Connected

Disconnected

Pro tip

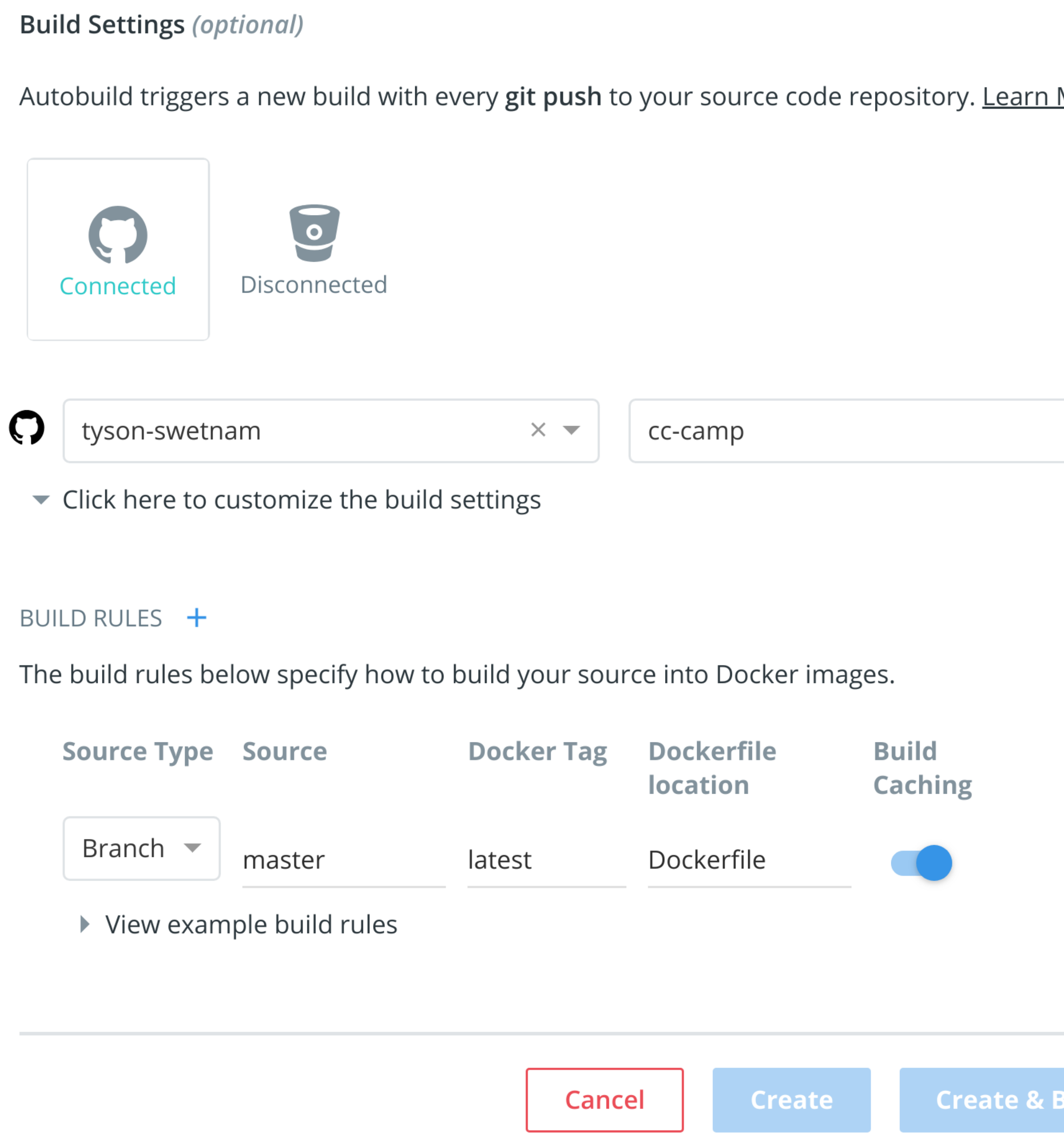
You may push a new image to this repository using

```
docker tag local-image:tagname new-re
docker push new-repo:tagname
```

Make sure to change *tagname* with your desired im tag.

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.



Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository. Learn M

tyson-swetnam

cc-camp

Click here to customize the build settings

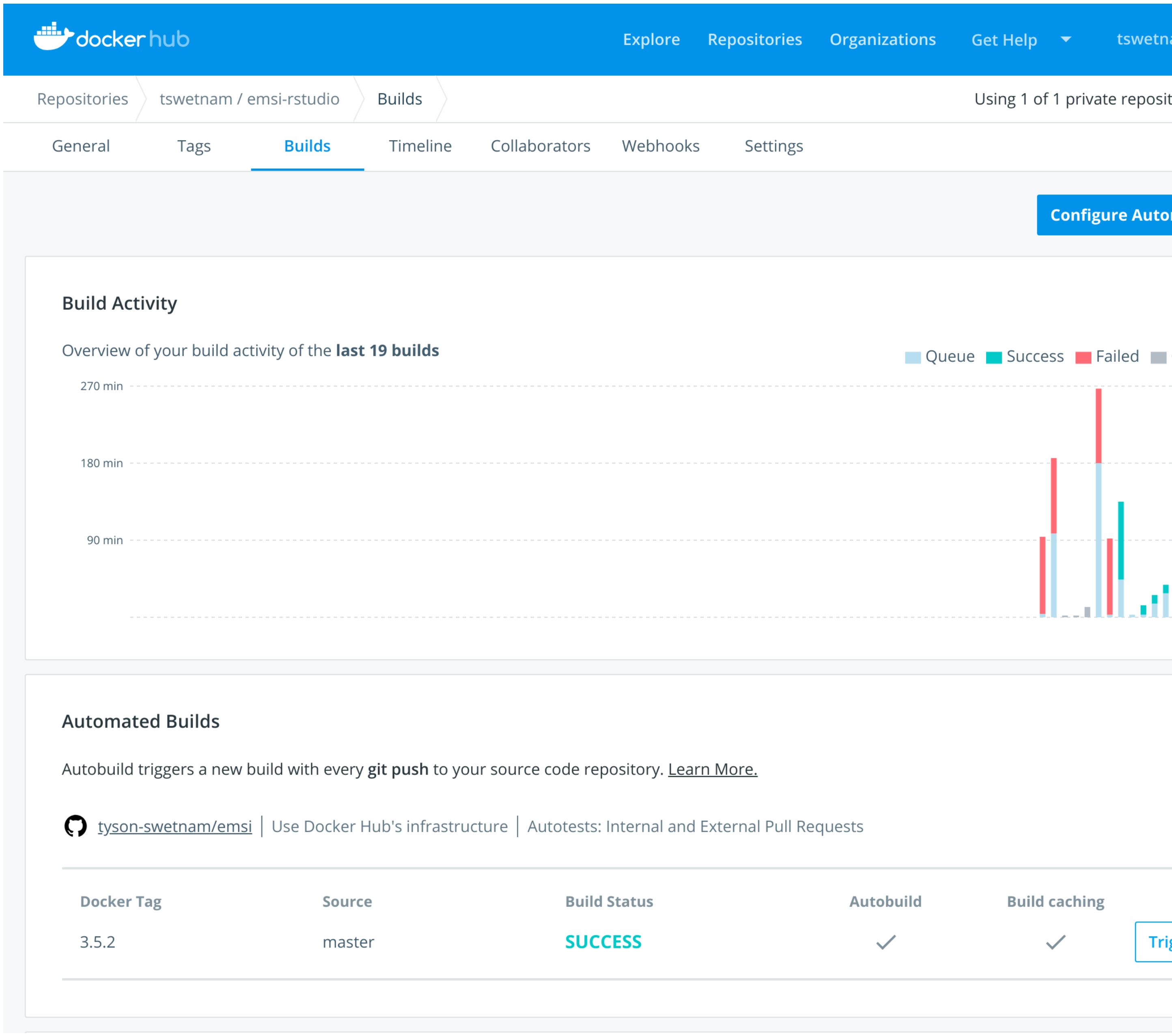
BUILD RULES +

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Caching
Branch	master	latest	Dockerfile	●

View example build rules

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:



Docker Tag	Source	Build Status	Autobuild	Build caching
3.5.2	master	SUCCESS	✓	✓

9.2.3 2.3 Automated Container Builds

Automated build repositories rely on the integration with a version control system (GitHub or Gitlab) where your `Dockerfile` is kept.

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

1. Initialize git repository for the *mynotebook* directory you created for your `Dockerfile`

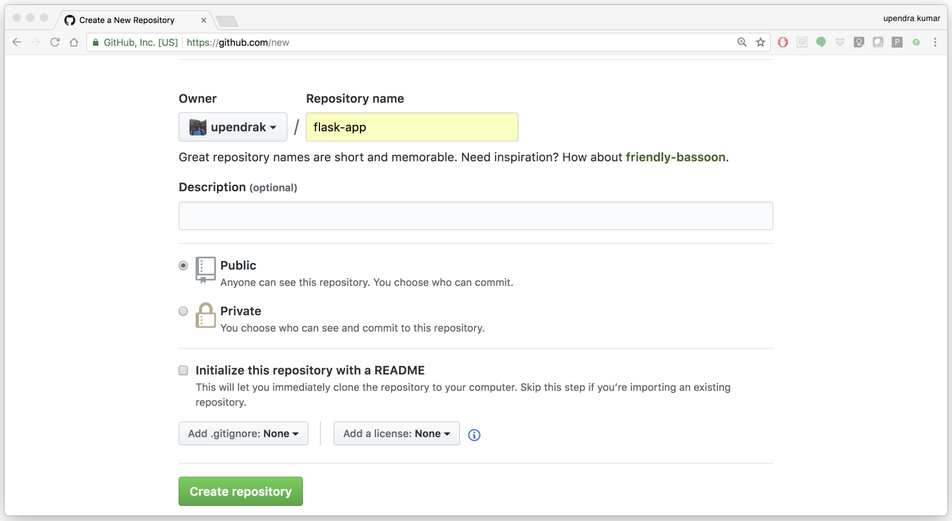
```
$ git init
Initialized empty Git repository in /home/julianp/mynotebook/.git/

$ git status
```

(continues on next page)

(continued from previous page)

```
On branch master

Initial commit

Untracked files:
(use "git add <file>..." to include in what will be committed)

        Dockerfile
        model.py

nothing added to commit but untracked files present (use "git add" to track)

$ git add * && git commit -m "Add files and folders"
[master (root-commit) a4f732a] Add files and folders
 2 files changed, 10 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 model.py
```

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



Owner: upendrak / Repository name: flask-app

Great repository names are short and memorable. Need inspiration? How about friendly-bassoon.

Description (optional)

Public — Anyone can see this repository. You choose who can commit.

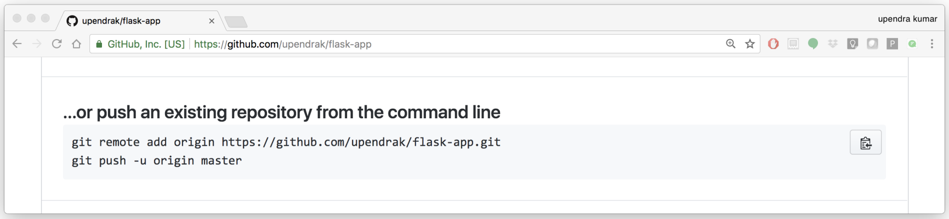
Private — You choose who can see and commit to this repository.

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

Note: Don't initialize the repository with a README and don't add a license.

3. Push the repository to github



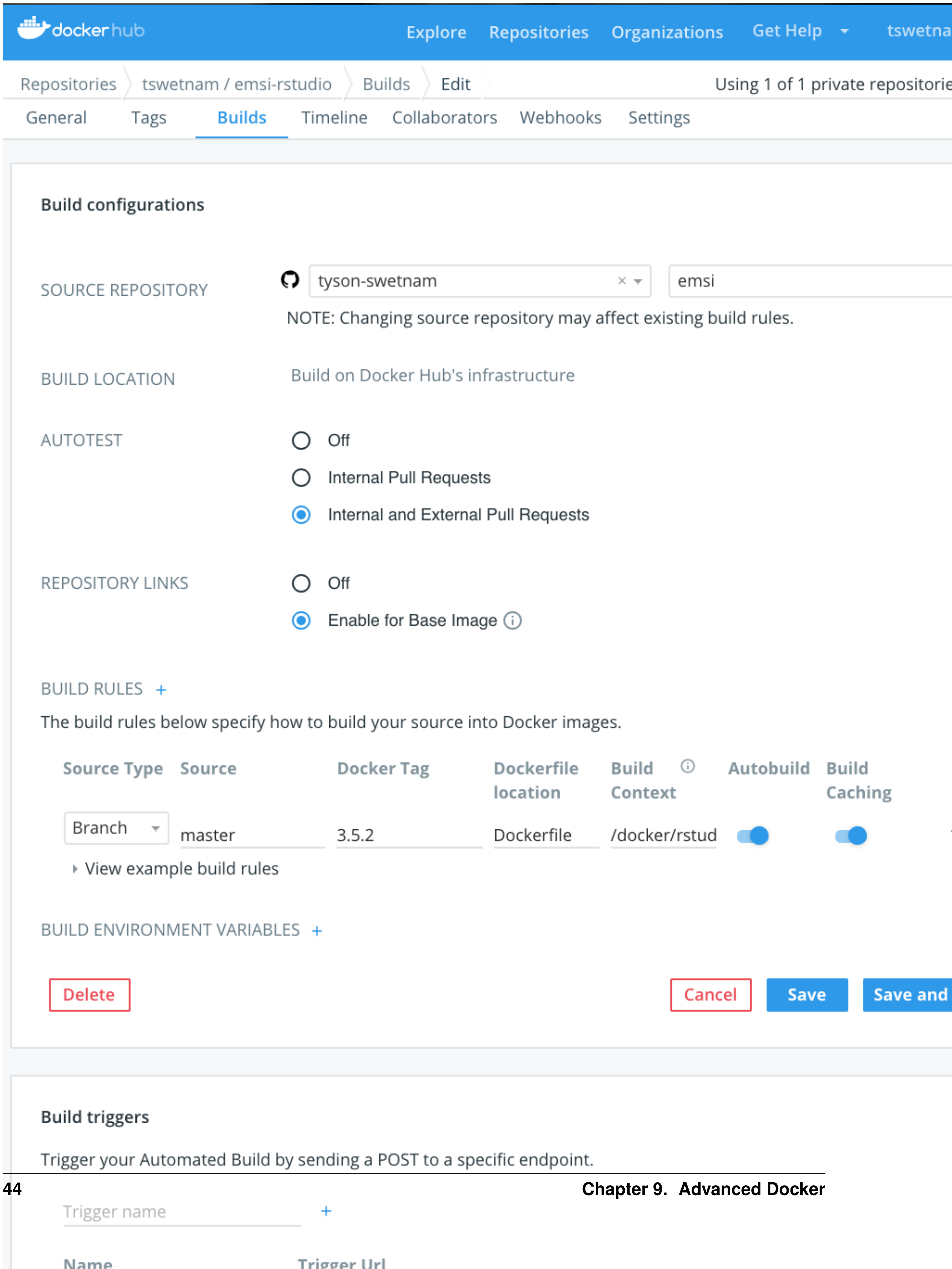
...or push an existing repository from the command line

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

```
$ git remote add origin https://github.com/<your-github-username>/mynotebook.git

$ git push -u origin master
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 1.44 KiB | 0 bytes/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/<your-github-username>/mynotebook.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

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 `mynotebook`. Type in "Jupyter Test" 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.



Build configurations

SOURCE REPOSITORY: tyson-swetnam | emsi

NOTE: Changing source repository may affect existing build rules.

BUILD LOCATION: Build on Docker Hub's infrastructure

AUTOTEST: Off [✓ Internal and External Pull Requests] Internal Pull Requests

REPOSITORY LINKS: Off [✓ Enable for Base Image]

BUILD RULES +

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Context	Autobuild	Build Caching
Branch	master	3.5.2	Dockerfile	/docker/rstud	●	●

View example build rules

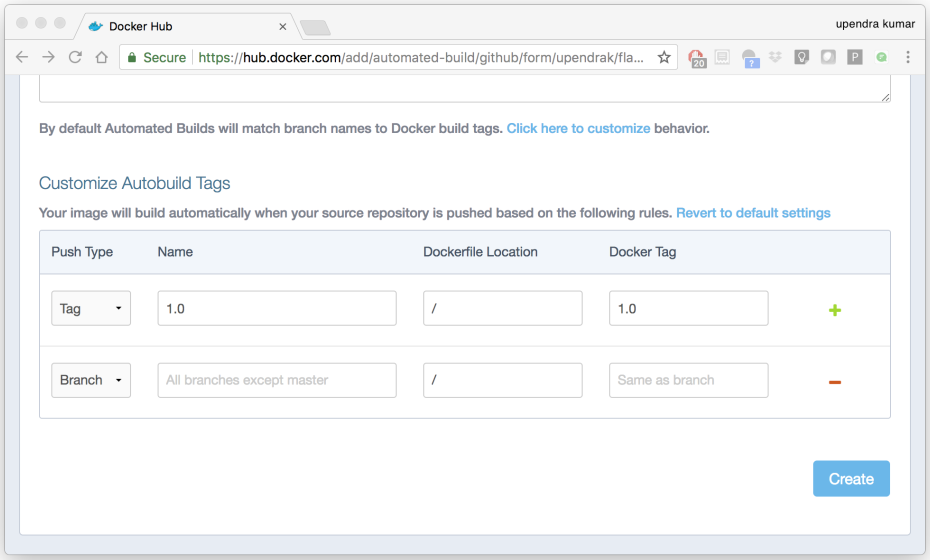
BUILD ENVIRONMENT VARIABLES +

Build triggers

Trigger your Automated Build by sending a POST to a specific endpoint.

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.


Docker Hub
upendra kumar
Secure https://hub.docker.com/add/automated-build/github/form/upendrak/fla...

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	−

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.

- Enter the `master` (default) for the name of the branch.
- 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.

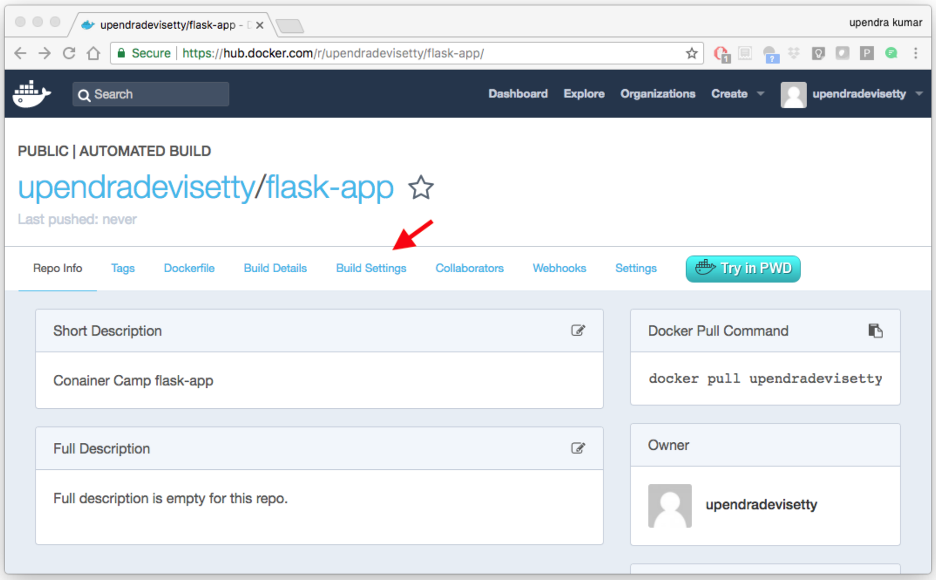
Warning: You can only trigger one build at a time and no more than one every five minutes. If you already have a build pending, or if you recently submitted a build request, Docker ignores new requests.

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 `mynotebook` repo, you'll trigger a manual build. Triggering a manual build ensures everything is working correctly.

From your automated build page choose `Build Settings`



Short Description

Conainer Camp flask-app

Docker Pull Command

```
docker pull upendradevisetty
```

Full Description

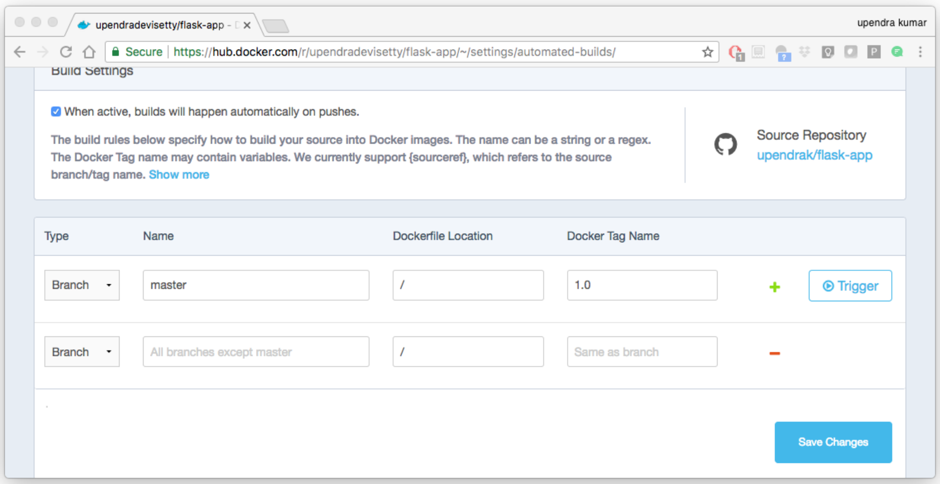
Full description is empty for this repo.

Owner

upendradevisetty

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.


upendradevisetty/flask-app
upendra kumar
Secure | https://hub.docker.com/r/upendradevisetty/flask-app/~/settings/automated-builds/
Build Settings
Source Repository
upendrak/flask-app
Show more
Trigger
Save Changes

☑ 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 {sourceref}, which refers to the source branch/tag name.

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

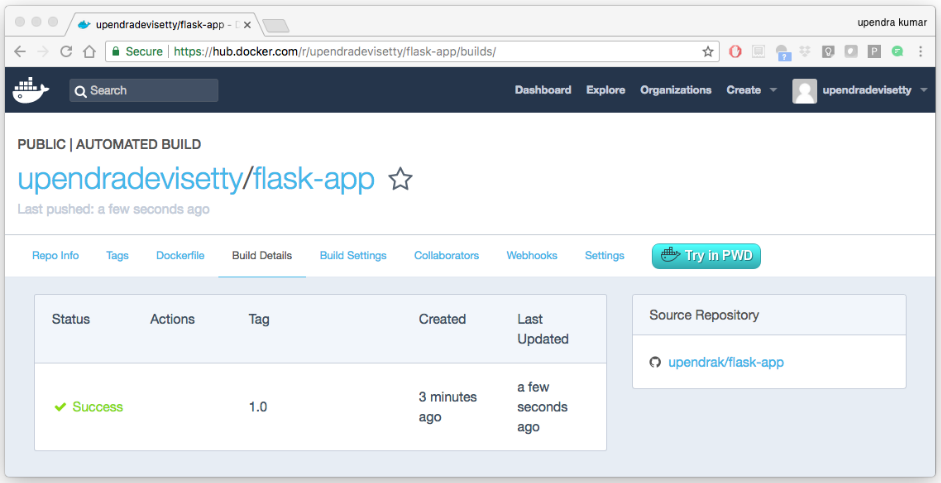
8. Review the build results

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/flask-app

Last pushed: a few seconds ago

Status	Actions	Tag	Created	Last Updated
✓ Success		1.0	3 minutes ago	a few seconds ago

Source Repository

upendrak/flask-app

9.2.4 Exercise 1 (5-10 mins): Updating and automated building

- `git add`, `commit` and `push` to your GitHub or Gitlab repo
- Trigger automatic build with a new tag (2.0) on Docker Hub
- Pull your Docker image from Docker Hub to a new location.
- Run the instance to make sure it works

9.3 3. Managing Data in Docker

It is possible to store data within the writable layer of a container, but 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.
- Its better to put your data into the container **AFTER** it is build - this keeps the container size smaller and easier to move across networks.

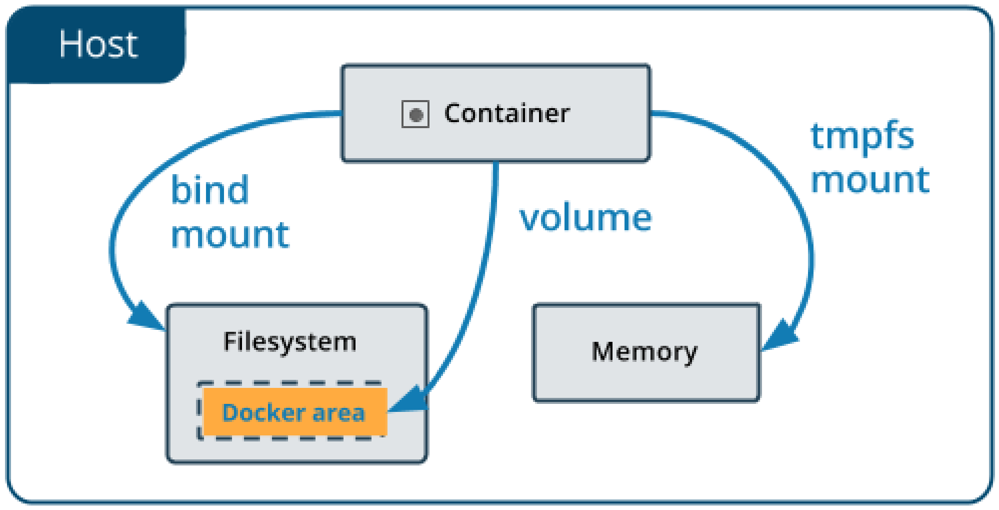
Docker offers three different ways to mount data into a container from the Docker host:

- **volumes**,
- **bind mounts**,
- **tmpfs volumes**.

When in doubt, volumes are almost always the right choice.

9.3.1 3.1 Volumes

Volumes are created and managed by Docker. You can create a new volume explicitly using the `docker volume create` command, or Docker can create a volume in the container when the container is built.



Volumes are often a better choice than persisting data in a container's writable layer, because using a volume does not increase the size of containers using it, and the volume's contents exist outside the lifecycle of a given container. While bind mounts (which we will see later) are dependent on the directory structure of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- A new volume's contents can be pre-populated by a container.

Note: If your container generates non-persistent state data, consider using a `tmpfs` mount to avoid storing the data anywhere permanently, and to increase the container's performance by avoiding writing into the container's writable layer.

3.1.1 Choose the -v or –mount flag for mounting volumes

Originally, the `-v` or `--volume` flag was used for standalone containers and the `--mount` flag was used for swarm services. However, starting with Docker 17.06, you can also use `--mount` with standalone containers. In general, `--mount` is more explicit and verbose. The biggest difference is that the `-v` syntax combines all the options together in one field, while the `--mount` syntax separates them. Here is a comparison of the syntax for each flag.

Tip: New users should use the `--mount` syntax. Experienced users may be more familiar with the `-v` or `--volume` syntax, but are encouraged to use `--mount`, because research has shown it to be easier to use.

`-v` or `--volume`: Consists of three fields, separated by colon characters (:). The fields must be in the correct order, and the meaning of each field is not immediately obvious.

- In the case of named volumes, the first field is the name of the volume, and is unique on a given host machine.
- The second field is the path where the file or directory are mounted in the container.
- The third field is optional, and is a comma-separated list of options, such as `ro`.

3.2 Bind mounts

When you run a container, you can bring a directory from the host system into the container, and give it a new name and location using the `-v` or `--volume` flag.

```
$ mkdir -p ~/local-data-folder
$ echo "some data" >> ~/local-data-folder/data.txt
$ docker run -v ${HOME}/local-data-folder:/data $YOUR_DOCKERHUB_USERNAME/
↪mynotebook:latest cat /data/data.txt
```

In the example above, you can mount a folder from your localhost, in your home user directory into the container as a new directory named `/data`.

3.3 Create and manage volumes

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

A given volume can be mounted into multiple containers simultaneously. When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using `docker volume prune` command.

When you create a Docker volume, it is stored within a directory on the Docker Linux host (`/var/lib/docker/`

Note: File location on Mac OS X is a bit different: https://timonweb.com/posts/getting-path-and-accessing-persistent-volumes-in-docker-for-mac/

Let's create a volume

```
$ docker volume create my-vol
```

List volumes:

```
$ docker volume ls

local                my-vol
```

Inspect a volume by looking at the Mount section in the *docker volume inspect*

```
$ docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
```

(continues on next page)

(continued from previous page)

```
        "Scope": "local"
    }
]
```

Remove a volume

```
$ docker volume rm my-vol
$ docker volume ls
```

3.3.1 Populate a volume using a container

This example starts an `alpine` container and populates the new volume `output-vol` with the some output created by the container.

```
docker volume create output-vol
docker run --name=data-app --mount source=output-vol,target=/data alpine sh -c 'env >>
↪ /data/container-env.txt'
```

Use `docker inspect output-vol` to see where the volume data lives on your host, and then use `cat` to confirm that it contains the output created by the container.

```
docker volume inspect output-vol
sudo cat /var/lib/docker/volumes/output-vol/_data/container-env.txt
```

You should see something like:

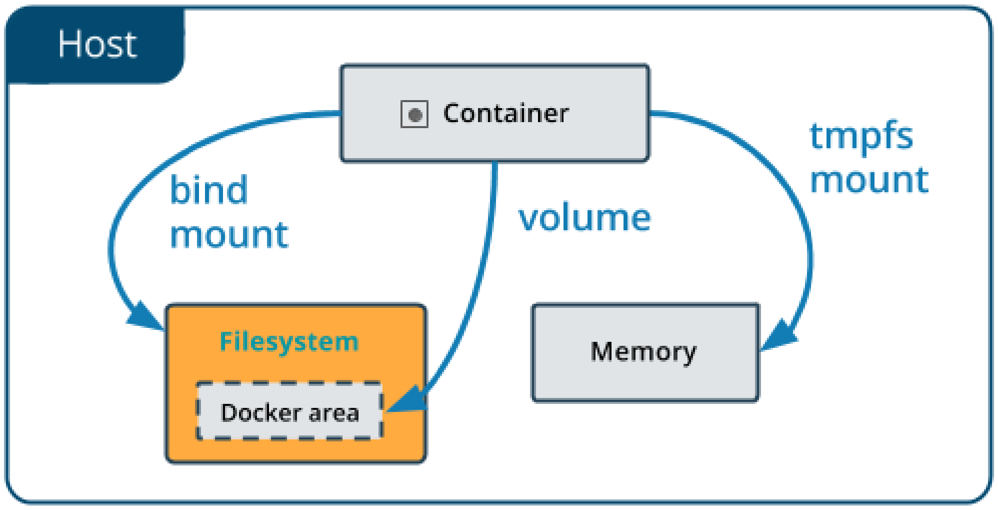
After running either of these examples, run the following commands to clean up the container and volume.

```
docker rm data-app
docker volume rm output-vol
```

9.3.2 3.4 Bind mounts

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

Tip: If you are developing new Docker applications, consider using named **volumes** instead. You can't use Docker CLI commands to directly manage bind mounts.



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.

3.2.1 Start a container with a bind mount

Create a `bind-data` directory in your home directory.

```
cd ~
mkdir -p ~/bind-data
```

Run a container, mounting this directory inside the container, and the container should create some data in there.

```
docker run --mount type=bind,source="$(pwd)"/bind-data,target=/data alpine sh -c 'env␣
↪>> /data/container-env.txt'
```

Check that the output looks right.

```
cat ~/bind-data/container-env.txt
```

3.4.1 Use a read-only bind mount

For some development applications, the container needs to write into the bind mount, so changes are propagated back to the Docker host. At other times, the container only needs read access.

This example modifies the one above but mounts the directory as a read-only bind mount, by adding `ro` to the (empty by default) list of options, after the mount point within the container. Where multiple options are present, separate them by commas.

```
docker run --mount type=bind,source="$(pwd)"/bind-data,target=/data,readonly alpine␣
↪sh -c 'ls -al /data/ && env >> /data/container-env.txt'
```

You should see an error message about not being able to write to a read-only file system.

```
sh: can't create /data/container-env.txt: Read-only file system
```

9.4 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

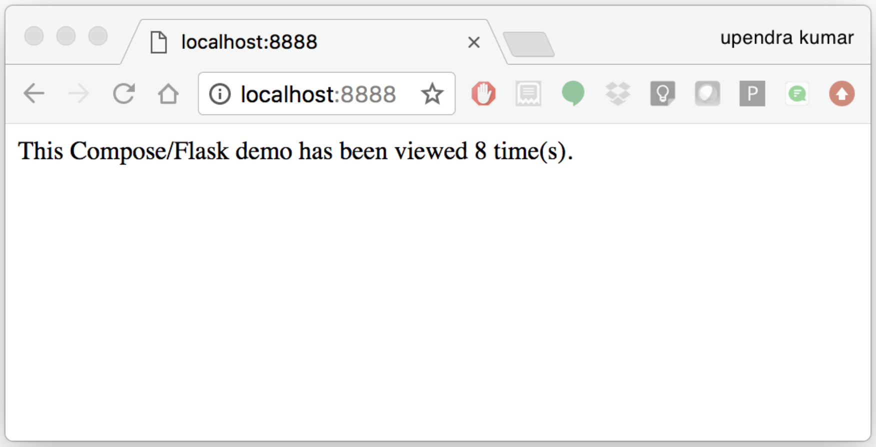
```
$ docker-compose up -d

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
```

(continues on next page)

(continued from previous page)

```
Digest: sha256:8cb593cb9cd1834429f0b4953a25617a8457e2c79b3e111c0f70bffd21acc467
Status: Downloaded newer image for python:2.7
 ---> 9e92c8430ba0
Step 2/5 : ADD . /code
 ---> 746bcecfc3c9
Step 3/5 : WORKDIR /code
 ---> c4cf3d6cb147
Removing intermediate container 84d850371a36
Step 4/5 : RUN pip install -r requirements.txt
 ---> Running in d74c2e1cfbf7
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
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 d74c2e1cfbf7
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`



This Compose/Flask demo has been viewed 8 time(s).

```
$ cat output.txt
Prediction of DecisionTreeClassifier:['apple' 'orange' 'apple']
```

CHAPTER 10

Docker hands-on exercises

10.1 Use case 1: Deploy a custom Docker image

- Download the sample code from https://github.com/Azure-Samples/docker-django-webapp-linux.git
- Build the image using the Dockerfile in that repo using `docker build` command
- Run an instance from that image
- Verify the web app and container are functioning correctly

10.2 Use case 2: Simple Bioinformatics example

Let's say if you find a cool tool/software and want to run it on your computer and as we found out in the morning session, it's not always easy to install the tool onto your computer or on a server natively. Since this workshop is about containers, let's containerize this tool.

For this simple hands-on exercise, let's containerize `fastqe` tool - https://github.com/lonsbio/fastqe. For those of you who are not from Bioinformatics, this tool generates read one or more FASTQ files, then it will compute quality stats for each file and print those stats as emoji… for some reason.

```
In [2]: mean_emoji("ERR048396_1.fastq")

🙀🙀🙀😬🙈😬😬🙈🙉🙉🙉🙉🙊🐵🐵🐵🐵🐵🐵🐵🐵🐵🐵🙊🐵🐵🐵🐵🐵🐵🙊🙊🐵🐵🐵🐵🙊🙊🙊🙊🙊🙊🙊
🙊🙊🙊🙊🙉🙉🙉🙉🙉🙉🙉🙉🙉🙉🙈😬🙈🙈🙈🙈🙈🙈🙈🙈😬😬😬😬😬😬😿🙀
```

```
In [3]: mean_emoji("ERR048396_2.fastq")

🙀🙀🙀😬😬😬😬😬🙉🙉🙉🙉🙉🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙊🙉🙉🙉🙉
🙉🙉🙉🙉🙉🙉🙉🙈🙉🙉🙉🙉🙉🙉🙉🙉🙉🙉🙉🙉🙉🙉🙈🙈🙈😬😬😬😬😬😬⚠️
```

Given a fastq file in Illumina 1.8+/Sanger format, calculate the mean (rounded) score for each position and print a corresponding emoji!

Tip: Natively you would install this tool like `pip install fastqe`. Now think of how you can dockerize this with appropriate base image and dependencies

After dockerizing the tool, for this exercise we don't have to bind mount the volume but just print the *fastqe* help and make sure that it is actually working.

Tip: Natively you would print the help of the tool as `fastqe -h`

10.3 1. Data Management Hands-on

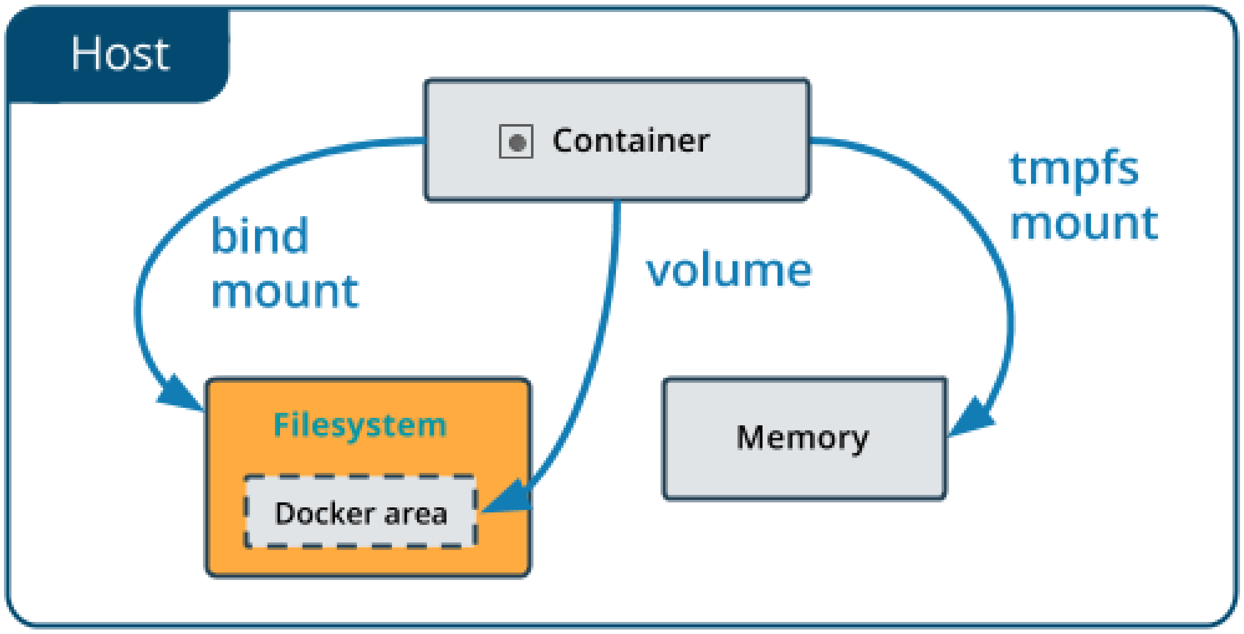
Form the "Introduction to Docker" session this morning, we learned that a running Docker container is an isolated environment created from a Docker image. This means, although it is possible to store data within the "writable layer" of a container, there are some limitations:

- 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 use bind mounts in our hands-on session, even though volumes is the more powerful and usable option for most use cases.

10.3.1 1.1 Bind mounts

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



Warning: A 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.

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

```
$ git clone https://github.com/CyVerse-learning-materials/ccw-2019-astro.git
```

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

```
$ cd ccw-2019-astro/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. Looking for the "Mounts" section,

```
$ docker inspect hops | grep -A9 Mounts
"Mounts": [
    {
        "Type": "bind",
        "Source": "/Users/ckchan/ccw-2019-astro/hops",
        "Destination": "/root",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

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

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

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

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

10.4 2. Jupyter Notebook Hands-on

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.

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

In this second hands-on, 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 `ccw-2019-astro` git repository that you downloaded earlier, there is a sample Galaxy simulation:

```
    $ pwd
    /Users/ckchan/ccw-2019-astro/hops
    $ cd ../galaxy/
    $ pwd
    /Users/ckchan/ccw-2019-astro/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/
↪jupyter/lab
```

(continues on next page)

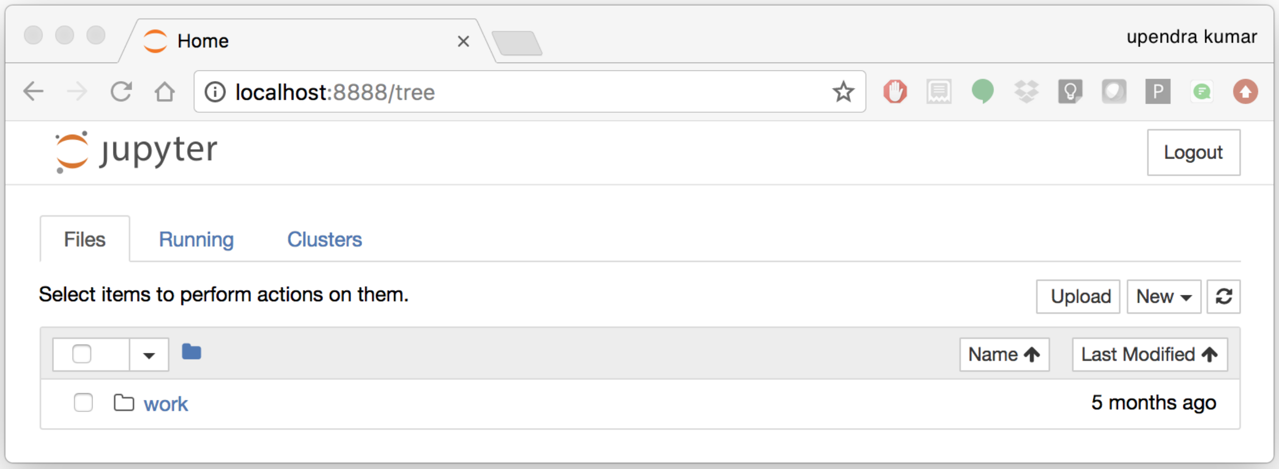
(continued from previous page)

```
    [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
```

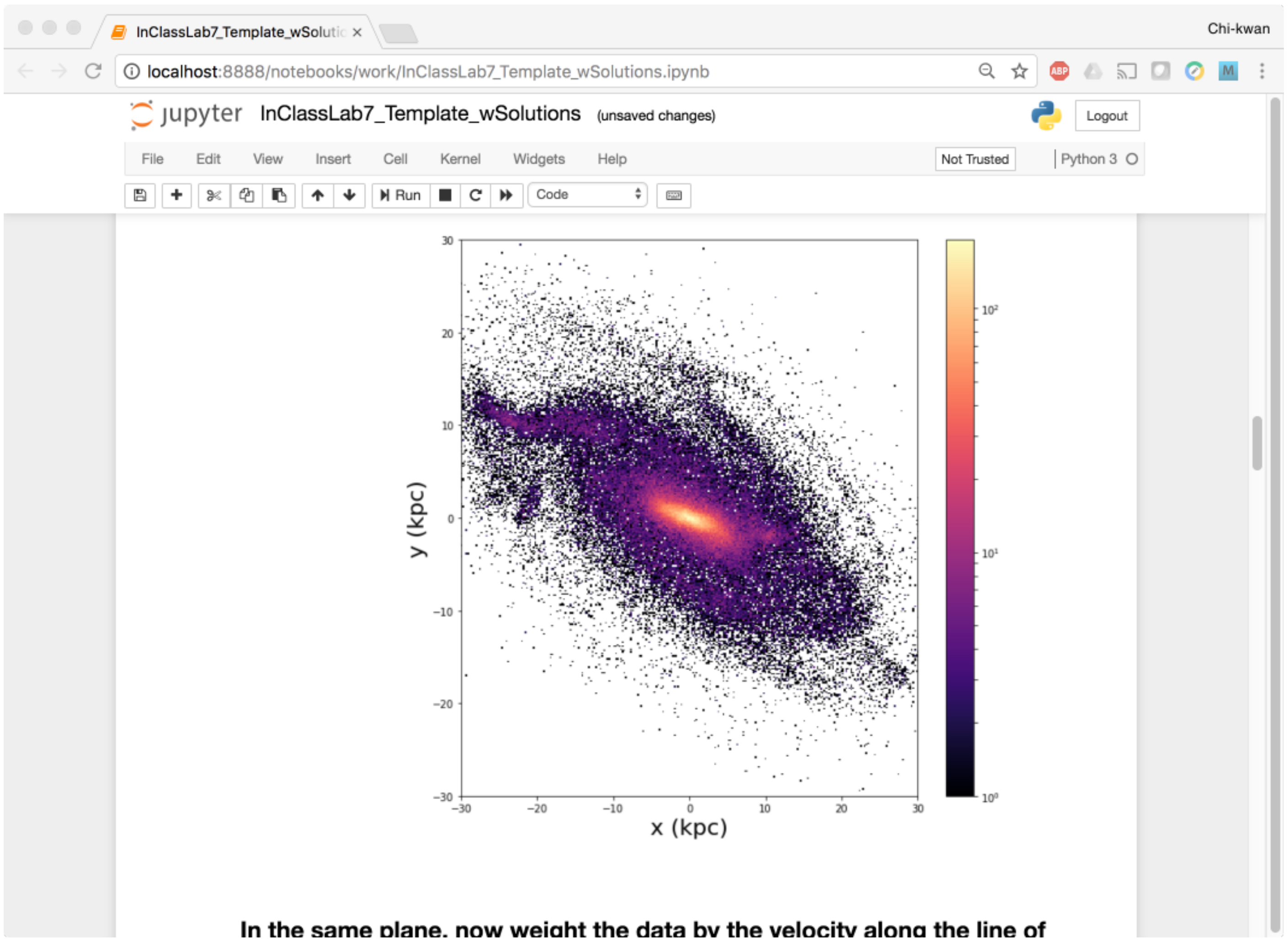
Warning: Do not copy and paste the above URL in your browser as this URL is specific to my environment.



Select items to perform actions on them.

work 5 months ago

You should be greeted by your own containerised Jupyter service! Now open `galaxy/InClassLab7_Template_wSolutions.ipynb` and try analysis a Galaxy simulation!



In the same plane, now weight the data by the velocity along the line of

To shut down the container, simply hit `Ctrl-C` in the terminal/command prompt twice. 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.

CHAPTER 11

Introduction to Singularity

11.1 1. Prerequisites

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

Note:

Important: Singularity is compatible with Docker, but they do have distinct differences.

Key Differences:

Docker:

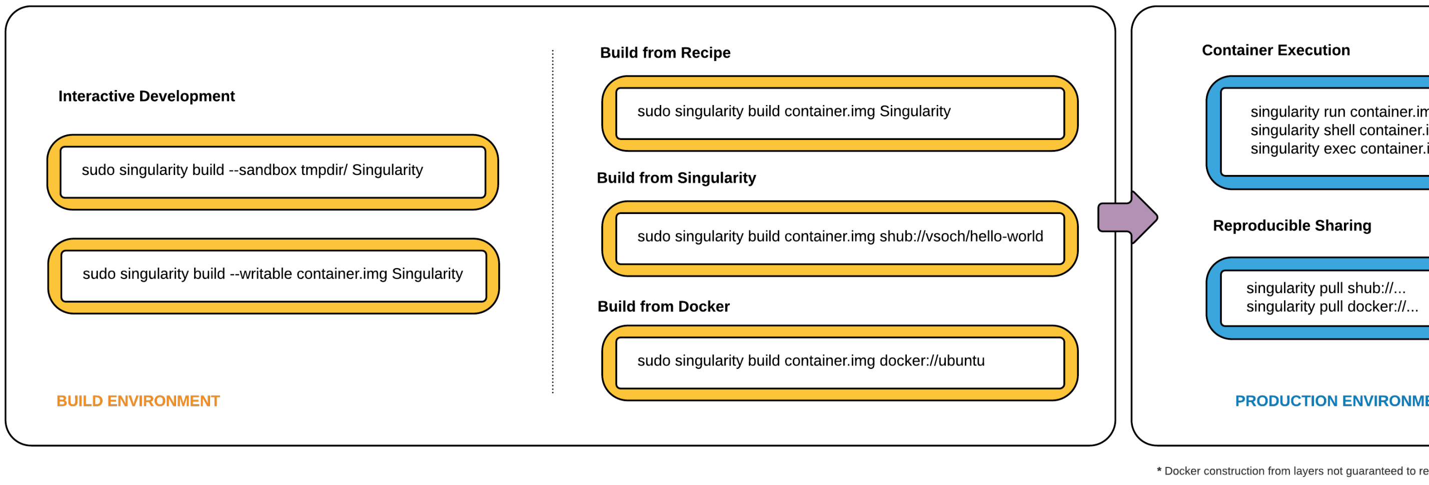
- Inside a Docker container the user has escalated privileges, effectively making them *root* on that host system. This privilege is not supported by *most* administrators of High Performance Computing (HPC) centers. Meaning that Docker is not, and will likely never be, installed natively on your HPC.

Singularity:

- Same user inside as outside the container
- User only has root privileges if elevated with *sudo* when container is run
- Can run (and modify!) existing Docker images and containers

These key differences allow Singularity to be installed on most HPC centers. Because you can run virtually all Docker containers in Singularity, you can effectively run Docker on an HPC.

Singularity uses a 'flow' whereby you (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.


Interactive Development
sudo singularity build --sandbox tmpdir/ Singularity
sudo singularity build --writable container.img Singularity
BUILD ENVIRONMENT
Build from Recipe
sudo singularity build container.img Singularity
Build from Singularity
sudo singularity build container.img shub://vsoch/hello-world
Build from Docker
sudo singularity build container.img docker://ubuntu
Container Execution
singularity run container.im
singularity shell container.i
singularity exec container.i
Reproducible Sharing
singularity pull shub://...
singularity pull docker://...
PRODUCTION ENVIRONME
* Docker construction from layers not guaranteed to re

11.2 2. Singularity Installation

Sylabs Singularity homepage: https://www.sylabs.io/docs/

While Singularity is more likely to be used on a remote system, e.g. HPC or cloud, you may want to develop your own containers first on a local machine or dev system (See figure above).

Note: Singularity exits as two major versions - 2 and 3. In this workshop we will be using version 3

11.2.1 2.1 Install Singularity on Laptop

To Install Singularity on your laptop or desktop PC follow the instructions from Singularity: https://www.sylabs.io/guides/3.0/user-guide/installation.html#installation

11.2.2 2.2 HPC

Load the Singularity module on a HPC

If you are interested in working on HPC, you may need to contact your systems administrator and request they install Singularity. Because singularity ideally needs setuid, your admins may have some qualms about giving Singularity this privilege. If that is the case, you might consider forwarding this letter to your admins.

Most HPC systems are running Environment Modules with the simple command *module*. You can check to see what is available:

```
$ module avail singularity
```

If Singularity is installed:

```
$ module load singularity/3/3.1
```

11.2.3 2.3 Atmosphere Cloud

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* <../cyverse/boot.html>_.

Type in the following:

```
$ ezs -r 3.1.0
    DEBUG: set version to 3.1.0

    * Updating ez singularity and installing singularity (this may take a few minutes,
↪ coffee break!)
    Cloning into '/opt/cyverse-ez-singularity'...
    remote: Enumerating objects: 6, done.
    remote: Counting objects: 100% (6/6), done.
    remote: Compressing objects: 100% (5/5), done.
    remote: Total 24 (delta 1), reused 4 (delta 1), pack-reused 18
    Unpacking objects: 100% (24/24), done.
    * singularity was updated successfully

    You shouldn't need to use ezs again on this system, unless you want to update␣
↪singularity itself

    To test singularity, type: singularity run shub://vsoch/hello-world
    Hint: it should output "RaawwWWWWWRRRR!!")
```

11.2.4 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
    WARNING: Authentication token file not found : Only pulls of public images will␣
↪succeed
     62.32 MiB / 62.32 MiB␣
↪[===============================================================================================]␣
↪100.00% 30.61 MiB/s 2s
```

Singularity's command line interface allows you to build and interact with containers transparently. You can run programs inside a container as if they were running on your host system. You can easily redirect IO, use pipes, pass arguments, and access files, sockets, and ports on the host system from within a container.

The help command gives an overview of Singularity options and subcommands as follows:

```
$ 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 runscript within container
    shell      Run a Bourne shell within container
    test       Launch a testscript 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/
```

Information about subcommand can also be viewed with the help command.

```
$ singularity help pull
Pull a container from a URI

Usage:
  singularity pull [pull options...] [output file] <URI>

Description:
  The 'pull' command allows you to download or build a container from a given
  URI.  Supported URIs include:

  library: Pull an image from the currently configured library
      library://[user[collection/[container[:tag]]]]
```

(continues on next page)

(continued from previous page)

```
  docker: Pull an image from Docker Hub
      docker://user/image:tag

  shub: Pull an image from Singularity Hub to CWD
      shub://user/image:tag

Options:
      --docker-login     interactive prompt for docker authentication
  -F, --force            overwrite an image file if it exists
  -h, --help             help for pull
      --library string   the library to pull from (default
                         "https://library.sylabs.io")
      --nohttps          do NOT use HTTPS, for communicating with local
                         docker registry


Examples:
  From Sylabs cloud library
  $ singularity pull alpine.sif library://alpine:latest

  From Docker
  $ singularity pull tensorflow.sif docker://tensorflow/tensorflow:latest

  From Shub
  $ singularity pull singularity-images.sif shub://vsoch/singularity-images


For additional help or support, please visit https://www.sylabs.io/docs/
```

11.3 3. Downloading pre-built images

The easiest way to use a Singularity is to `pull` an existing container from one of the Registries.

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:

- *library* - images hosted on Sylabs Cloud
- *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

11.3.1 3.1 Pulling an image from Singularity Hub

Similar to previous example, in this example I am pulling a base Ubuntu container from Singularity-Hub:

```
$ singularity pull shub://singularityhub/ubuntu
WARNING: Authentication token file not found : Only pulls of public images will␣
↪succeed
    88.58 MiB / 88.58 MiB␣
↪[===============================================================================================]␣
↪100.00% 31.86 MiB/s 2s
```

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

```
$ singularity pull --name ubuntu_test.simg shub://singularityhub/ubuntu
WARNING: Authentication token file not found : Only pulls of public images will␣
↪succeed
    88.58 MiB / 88.58 MiB␣
↪[===============================================================================================]␣
↪100.00% 35.12 MiB/s 2s
```

The above command will save the alpine image from the Container Library as `alpine.sif`

11.3.2 3.2 Pulling an image from Docker Hub

This example pulls an `ubuntu:16.04` image from DockerHub and saves it to the working directory.

```
$ singularity pull docker://ubuntu:16.04
WARNING: Authentication token file not found : Only pulls of public images will␣
↪succeed
INFO:    Starting build...
Getting image source signatures
Copying blob sha256:7b722c1070cdf5188f1f9e43b8413157f8dfb2b4fe84db3c03cb492379a42fcc
 41.51 MiB / 41.51 MiB [====================================================] 1s
Copying blob sha256:5fbf74db61f1459176d8647ba8f53f8e6cf933a2e56f73f0e8da81213117b7e9
 847 B / 847 B [============================================================] 0s
Copying blob sha256:ed41cb72e5c918bdbd78e68f02930a3f1cf1d6079402b0a5b19de8508e67b766
 526 B / 526 B [============================================================] 0s
Copying blob sha256:7ea47a67709ebea8efed59fbda703dbd00a0d2cae7e2808959744bfa30bfc0e9
 168 B / 168 B [============================================================] 0s
Copying config sha256:288b5aca25f70512e5874c289a8a216b60808ecc47f687fa502fd848e5c3f875
 2.42 KiB / 2.42 KiB [======================================================] 0s
Writing manifest to image destination
Storing signatures
INFO:    Creating SIF file...
INFO:    Build complete: ubuntu_16.04.sif
```

Warning: Pulling Docker images reduces reproducibility. If you were to pull a Docker image today and then wait six months and pull again, you are not guaranteed to get the same image. If any of the source layers has changed the image will be altered. If reproducibility is a priority for you, try building your images from the Container Library.

11.3.3 3.3 Pulling an image from Sylabs cloud library

Let's use an easy example of `alpine.sif` image from the container library

```
$ singularity pull library://alpine:latest
WARNING: Authentication token file not found : Only pulls of public images will␣
↪succeed
INFO:    Downloading library image
2.08 MiB / 2.08 MiB␣
↪[==========================================================================================
↪100.00% 5.06 MiB/s 0s
```

Tip: You can use `singularity search alpine` command to locate groups, collections, and containers of interest on the Container Library

11.4 4 Interact with images

You can interact with images in several ways such as `shell`, `exec` and `run`.

For these examples we will use a `lolcow_latest.sif` image that can be pulled from the Container Library like so.

```
$ singularity pull library://sylabsed/examples/lolcow
```

11.4.1 4.1 Shell

The `shell` command allows you to spawn a new shell within your container and interact with it as though it were a small virtual machine.

```
$ singularity shell lolcow_latest.sif
  Singularity lolcow_latest.sif:~>
```

The change in prompt indicates that you have entered the container (though you should not rely on that to determine whether you are in container or not).

Once inside of a Singularity container, you are the same user as you are on the host system.

```
$ Singularity lolcow_latest.sif:~> whoami
upendra_35
Singularity lolcow_latest.sif:~> id
uid=14135(upendra_35) gid=10013(iplant-everyone) groups=10013(iplant-everyone),
↪100(users),119(docker),10000(staff),10007,10054,10064(atmo-user),10075(myplant-
↪users),10083(tito-admins),10084(tito-qa-admins),10092(geco-admins),10100(sciteam)
```

Note: `shell` also works with the library://, docker://, and shub:// URIs. This creates an ephemeral container that disappears when the shell is exited.

11.4.2 4.2 Executing commands

The exec command allows you to execute a custom command within a container by specifying the image file. For instance, to execute the `cowsay` program within the lolcow_latest.sif container:

```
$ singularity exec lolcow_latest.sif cowsay container camp rocks
______________________
< container camp rocks >
 ----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Note: `exec` also works with the library://, docker://, and shub:// URIs. This creates an ephemeral container that executes a command and disappears.

11.4.3 4.3 Running a container

Singularity containers contain runscripts. These are user defined scripts that define the actions a container should perform when someone runs it. The runscript can be triggered with the `run` command, or simply by calling the container as though it were an executable.

```
singularity run lolcow_latest.sif
 _________________________________________
/  You will remember, Watson, how the     \
| dreadful business of the Abernetty      |
| family was first brought to my notice   |
| by the depth which the parsley had sunk |
| into the butter upon a hot day.         |
|                                         |
\ -- Sherlock Holmes                      /
 -----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

# Exercise - 1

Now that you know how to run containers from Docker, I want you to run a Singular container from *simple-script* Docker image that you create on Day 1 of the workshop.

Note: If you don't have `simple-script` you can use my image on docker hub - https://hub.docker.com/r/upendradevisetty/simple-script-auto

Here are the brief steps:

1. Go to Docker hub and look for the Dockerhub image that you built on Day 1
2. Use `singularity pull` command to pull the Docker image onto your working directory on the Atmosphere
3. Use `singularity run` command to launch a container from the Docker image and check to see if you get the same output that as you get from running `docker run`

11.4.4 4.3 Running a container on HPC

For running a container on HPC, you need to have Singularity module available on HPC. Let's first look to see if the Singularity module is available on HPC or not

> **Warning:** The following instructions are from running on UA HPC. It may or may not work on other HPC. Please refer to HPC documentation to find similar commands

```
$ module avail singularity
---------------------------------------------------------- /cm/shared/uamodulefiles --
↪---------------------------------------------------------
singularity/2/2.6.1  singularity/3/3.0.2  singularity/3/3.1
```

You can see that there are three different versions of Singularity are available. For this workshop, we will use `singularity/3/3.1`. Let's load it now

```
$ module load singularity/3/3.1
```

4.3.1 Running fastqc

Let's run fastqc on UA HPC

```
$ mkdir fastqc && cd fastqc

$ wget https://de.cyverse.org/dl/d/A48695A7-69A7-46C1-B6BB-E036F4922EB2/test.R1.fq.gz
```

Write a pbs script (`fastqc_job.sh`) for job submission

```
#!/bin/bash
# Your job will use 1 node, 1 core, and 1gb of memory total.
#PBS -q standard
#PBS -l select=1:ncpus=2:mem=1gb:pcmem=6gb

### Specify a name for the job
#PBS -N fastqc

### Specify the group name
#PBS -W group_list=nirav

### Used if job requires partial node only
#PBS -l place=pack:shared

### CPUtime required in hhh:mm:ss.
### Leading 0's can be omitted e.g 48:0:0 sets 48 hours
#PBS -l cput=0:15:0

### Walltime is created by cputime divided by total cores.
### This field can be overwritten by a longer time
#PBS -l walltime=0:15:0

date
module load singularity/3/3.1
cd /extra/upendradevisetty/fastqc
singularity pull docker://quay.io/biocontainers/fastqc:0.11.8--1
singularity exec fastqc_0.11.8--1.sif fastqc test.R1.fq.gz
date
```

Submit the job now to UAHPC

```
$ qsub fastqc_job.sh
```

After the job is submitted, expect to get these outputs

```
-rwxr-xr-x 1 upendradevisetty nirav 260M Mar  8 09:29 fastqc_0.11.8--1.sif
-rw------- 1 upendradevisetty nirav 3.4K Mar  8 09:30 fastqc.e1875372
-rw-r--r-- 1 upendradevisetty nirav 434K Mar  8 09:30 test.R1_fastqc.zip
-rw-r--r-- 1 upendradevisetty nirav 625K Mar  8 09:30 test.R1_fastqc.html
-rw------- 1 upendradevisetty nirav  241 Mar  8 09:30 fastqc.o1875372
```

# Exercsise -2

- For those of you, who have access to HPC, try to run the container from `simple-script` Dockerhub on HPC.

CHAPTER 12

Advanced Singularity

12.1 5.0 Building your own Containers from scratch

In this section we'll go over the creation of Singularity containers from a recipe file, called `Singularity` (equivalent to `Dockerfile`).

12.2 5.1 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
```

(continues on next page)

(continued from previous page)

```
$ SINGULARITY_TMPDIR=$PWD/scratch SINGULARITY_CACHEDIR=$PWD/tmp singularity --debug␣
↪pull --name ubuntu-tmpdir.sif docker://ubuntu
```

12.2.1 5.2 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 a container using a custom Singularity file:

```
$ singularity build ubuntu-latest.sif Singularity
```

We've already covered how you can pull an existing container from Docker Hub, but we can also build a Singularity container from docker using the build command:

```
$ sudo singularity build --sandbox ubuntu-latest/  docker://ubuntu

$ singularity shell --writable ubuntu-latest/

Singularity ubuntu-latest.sif:~> apt-get update
```

Does it work?

```
$ sudo singularity shell ubuntu-latest.sif

Singularity: Invoking an interactive shell within container...

Singularity ubuntu-latest.sif:~> apt-get update
```

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 sandbox container after closing the container and restart.

In order to make these changes permanant, I need to rebuild the sandbox as a `.sif` image

```
$ sudo singularity build ubuntu-latest.sif ubuntu-latest/
```

Note: Why is creating containers in this way a **bad** idea?

12.3 5.2.1: Exercise (~30 minutes): Create a Singularity file

SyLabs User Guide

A `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.

- **Header**

The top of the file, selects the base OS for the container, just like `FROM` in Docker.

Bootstrap: references another registry (e.g. `docker` for DockerHub, `debootstrap`, or `shub` for Singularity-Hub).

`From:` selects the tag name.

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

Pulls a container from Singularity Hub (< v2.6.1)

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 in the Introduction Section 3.1.

- The Singularity file uses sections to specify the dependencies, environmental settings, and runscripts when it builds.

The additional sections of a Singularity file include:

- %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

CHAPTER 13

Setting up Singularity file system

- **Help**

%help section can be as verbose as you want

```
Bootstrap: docker
From: ubuntu

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

- **Setup**

%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**

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

- **Post**

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

- **Environment**

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

- **Runscript**

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

13.1 Example Singularity file

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

```
BootStrap: docker
From: ubuntu:18.04

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

%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 cowsay.sif Singularity
```

Run the container:

```
singularity run cowsay.sif
```

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

13.1.1 6.1 Using HPC Environments

Conducting analyses on high performance computing clusters happens through different patterns of interaction than running analyses on a cloud VM. When you login, you are on a node that is shared with lots of people, typically called the "login node". Trying to run jobs on the login node is not "high performance" at all (and will likely get you an admonishing email from the system administrator). Login nodes are intended to be used for moving files, editing files, and launching jobs.

Importantly, most jobs run 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, how much RAM, what type of nodes, and for how long) in addition to what you want to run. Then the scheduler finally has resources matching your requirements, it runs the job for you. If your request is very large, or very long, you may never make it out of the queue.

For example, on a VM if you run the command:

```
singularity exec docker://python:latest /usr/local/bin/python
```

The container will immediately start.

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

```
#!/bin/bash
#
#SBATCH -J myjob                      # Job name
#SBATCH -o output.%j                  # Name of stdout output file (%j expands to␣
↪jobId)
```

(continues on next page)

(continued from previous page)

```
#SBATCH -p development                # Queue name
#SBATCH -N 1                          # Total number of nodes requested (68 cores/
↪node)
#SBATCH -n 17                         # Total number of mpi tasks requested
#SBATCH -t 02:00:00                   # Run time (hh:mm:ss) - 4 hours

module load singularity/3/3.1
singularity exec docker://python:latest /usr/local/bin/python
```

This example is for the Slurm scheduler. Each of the #SBATCH lines looks like a comment to the bash kernel, but the scheduler reads all those lines to know what resources to reserve for you.

It is usually possible to get an interactive session as well, by using an interactive flag, *-i*.

Note: Every HPC cluster is a little different, but they almost universally have a "User's Guide" that serves both as a quick reference for helpful commands and contains guidelines for how to be a "good citizen" while using the system. For TACC's Stampede2 system, see the user guide. For The University of Arizona, see the user guide.

13.2 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 is in the process of changing image formats. Depending on the version of Singularity running on the HPC system, new squashFS or .simg formats may not work.

13.2.1 6.2 Singularity and MPI

Singularity supports MPI fairly well. Since (by default) the network is the same insde 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, MVAPICH2, and Intel MPI to name three) with some non-overlapping feature sets. If the host and container are running different MPI implementations, or even different versions of the same implementation, hilarity 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 Stampede2, have high-speed, low-latency networks that have special drivers. Infiniband, Ares, and OmniPath are three different specs for these types of networks. 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 3.1 compatible example of what it may look like:

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:

```
mpirun -np 4 singularity exec ./mycontainer.sif /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)

```
singularity exec ./mycontainer.sif mpirun -np 4 /app.py arg1 arg2
```

13.2.2 6.3 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. From Maverick, which is TACC's GPU system:

```
# Load the singularity module
module load singularity/3/3.1

# Pull your image

singularity pull docker://nvidia/caffe:latest

singularity exec --nv caffe-latest.sif caffe device_query -gpu 0
```

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.

```
singularity exec caffe-latest.sif caffe device_query -gpu 0
```

For TensorFlow, you can directly pull their latest GPU image and utilize it as follows.

```
# Change to your $WORK directory
cd $WORK
#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.sif 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.img, the /work directory does not exist, so any files there are inaccessible to the container.

The University of Arizona HPS Singularity examples.

13.3 7.0 Cryptographic Security

Documentation

CHAPTER 14

Biocontainers

Biocontainers for container camp by Amanda Cooksey - https://de.cyverse.org/dl/d/7AD8F379-5FAA-43F9-B528-252BAECE6066/BioContainers_for_Container_Camp.pdf

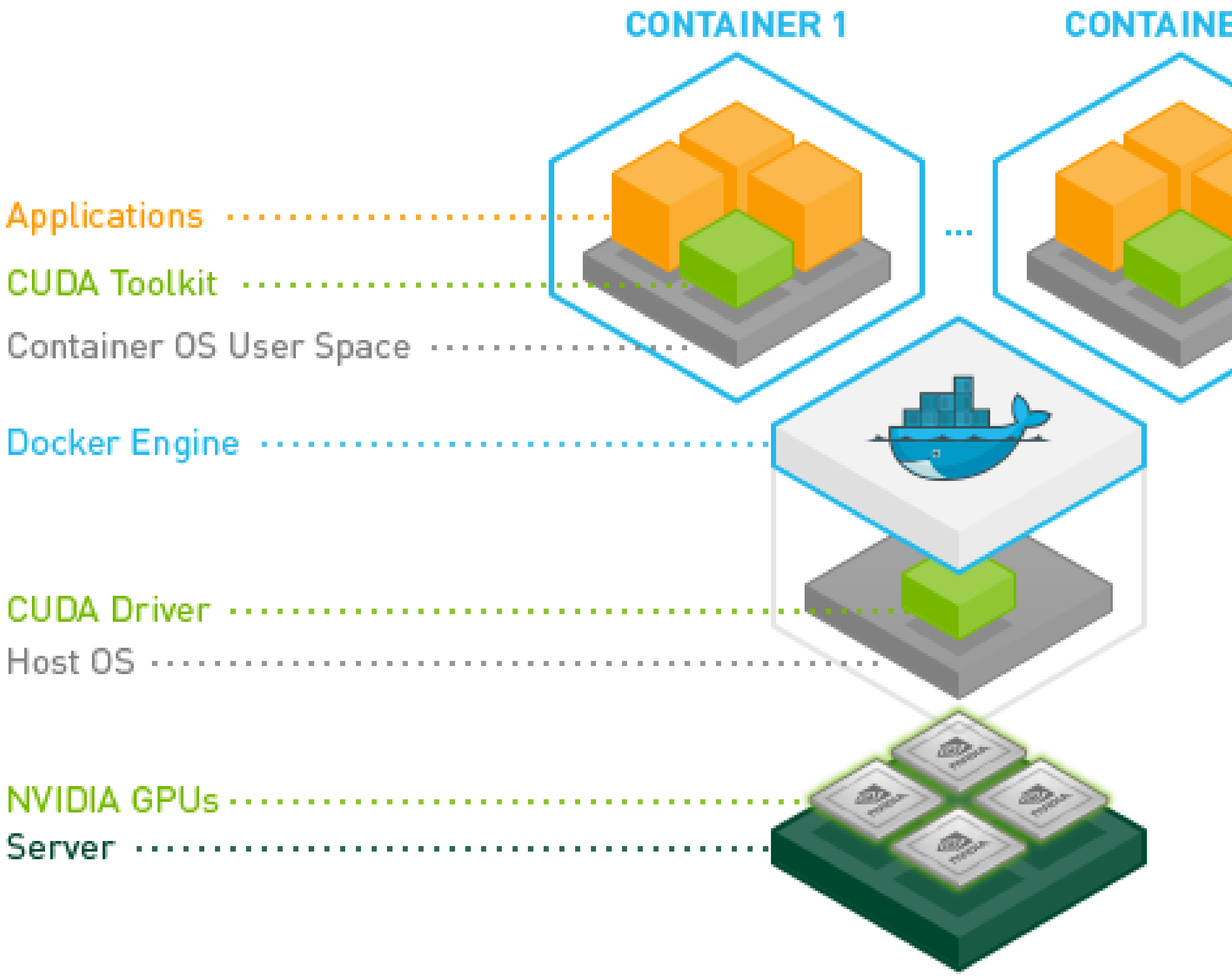
CHAPTER 15

NVIDIA-Docker

15.1 Background

NVIDIA is one of the leading makers of graphic processing units (GPU). GPU were established as a means of handling graphics processing operations for video cards, but have been greatly expanded for use in generalized computing applications. GPU are used for various applications in Machine Learning, image processing, and matrix-based linear algebras.

15.2 NVIDIA Docker



NVIDIA have created their own set of Docker containers for running on CPU-GPU enabled systems.

NVIDIA-Docker runs atop the NVIDIA graphics drivers on the host system, the NVIDIA drivers are imported to the container at runtime.

NVIDIA Docker Hub hosts numerous NVIDIA Docker containers, from which you can build your own images.

15.3 NVIDIA GPU Cloud

NVIDIA GPU Cloud hosts numerous containers for HPC and Cloud applications. You must register an account with them (free) to access these.

15.3.1 Registry

NVIDIA GPU Cloud hosts three registry spaces

- *nvcr.io/nvidia* - catalog of fully integrated and optimized deep learning framework containers.
- *nvcr.io/nvidia-hpcvis* - catalog of HPC visualization containers (beta).
- *nvcr.io/hpc* - popular third-party GPU ready HPC application containers.

15.4 Running NVIDIA Docker with Singularity

You can run NVIDIA Docker on HPC by using Singularity, but it requires a few steps, including authentication to NVIDIA GPU cloud.

NVIDIA to Singularity Official YouTube demo

15.5 Running GPU accelarated GUI applications with NVIDIA-Docker

NVIDIA Docker can be used as a base-image to create containers running graphical applications remotely. High resolution 3D screens are piped to a remote desktop platform.

Programs which leverage 3D applications include VirtualGL, TurboVNC, & TigerVNC.

An example application of a graphics-enabled remote desktop is the use of Blender for creating high level of detail images or animations.

CHAPTER 16

Containerized Workflows

16.1 Intro to workflows for efficient automated data analysis, using snakemake

Authors:

> C. Titus Brown, titus@idyll.org

> Sateesh Peri, https://sateeshperi.github.io/

In this breakout session you'll learn about snakemake, a workflow management system consisting of a text-based workflow specification language and a scalable execution environment. You will be introduced to the Snakemake workflow definition language and how to use the execution environment to scale workflows to compute servers and clusters while adapting to hardware specific constraints.

Snakemake is designed specifically for computationally intensive and/or complex data analysis pipelines. The name is a reference to the programming language Python, which forms the basis for the Snakemake syntax.

Note: You don't need to be an expert at Python to use Snakemake, but it can sometimes be very useful.

Click here for Snakemake setup and complete tutorial.

CHAPTER 17

Docker for Datascience

For a data scientist, running a container that is already equipped with the libraries and tools needed for a particular analysis eliminates the need to spend hours debugging packages across different environments or configuring custom environments.

But why Set Up a Data Science Environment in a Container?

- One reason is speed. Docker containers allow a Jupyter or RStudio session to launch in minutes, not hours.
- Containerization benefits both data science and IT/technical operations teams.
- Containers solve a lot of common problems associated with doing data science work at the enterprise level. They take the pressure off of IT to produce custom environments for every analysis, standardize how data scientists work, and ensure that old code doesn't stop running because of environment changes.
- Configuring a data science environment can be a pain. Dealing with inconsistent package versions, having to dive through obscure error messages, and having to wait hours for packages to compile can be frustrating. This makes it hard to get started with data science in the first place, and is a completely arbitrary barrier to entry.

Thanks to the rich ecosystem, there are already several readily available images for the common components in data science pipelines. Here are some Docker images to help you quickly spin up your own data science pipeline:

- MySQL
- Postgres
- Redmine
- MongoDB
- Hadoop
- Spark
- Zookeeper
- Kafka
- Cassandra
- Storm

- Flink
- R

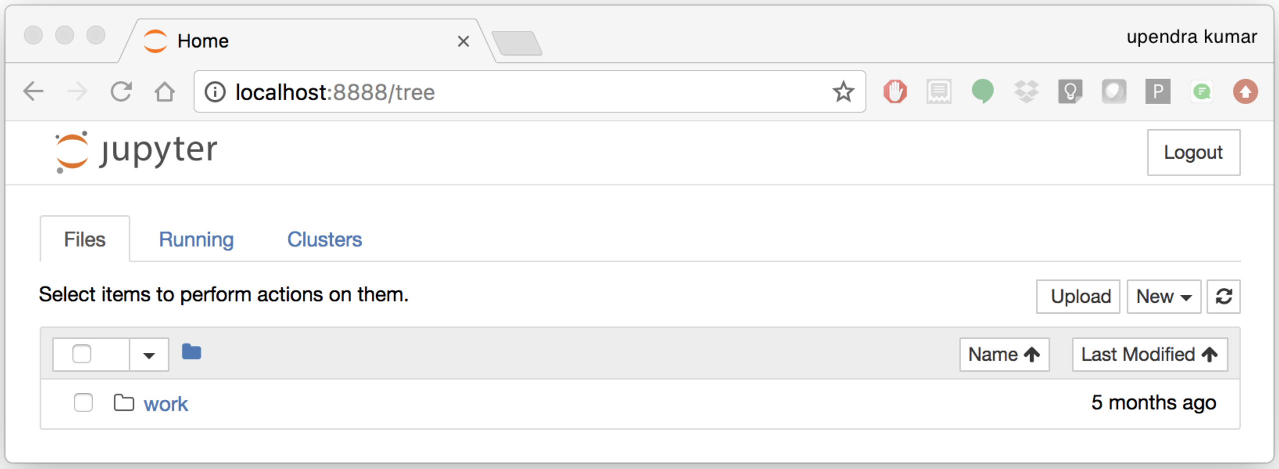
Motivation: Say you want to play around with some cool data science libraries in Python or R but what you don't want to do is spend hours on installing Python or R, working out what libraries you need, installing each and every one and then messing around with the tedium of getting things to work just right on your version of Linux/Windows/OSX/OS9—well this is where Docker comes to the rescue! With Docker we can get a Jupyter 'Data Science' notebook stack up and running in no time at all. Let's get started! We will see few examples of these in the following sections. . .

17.1 1. Launch a Jupyter notebook conatiner

Docker allows us to run a 'ready to go' Jupyter data science stack in what's known as a container:

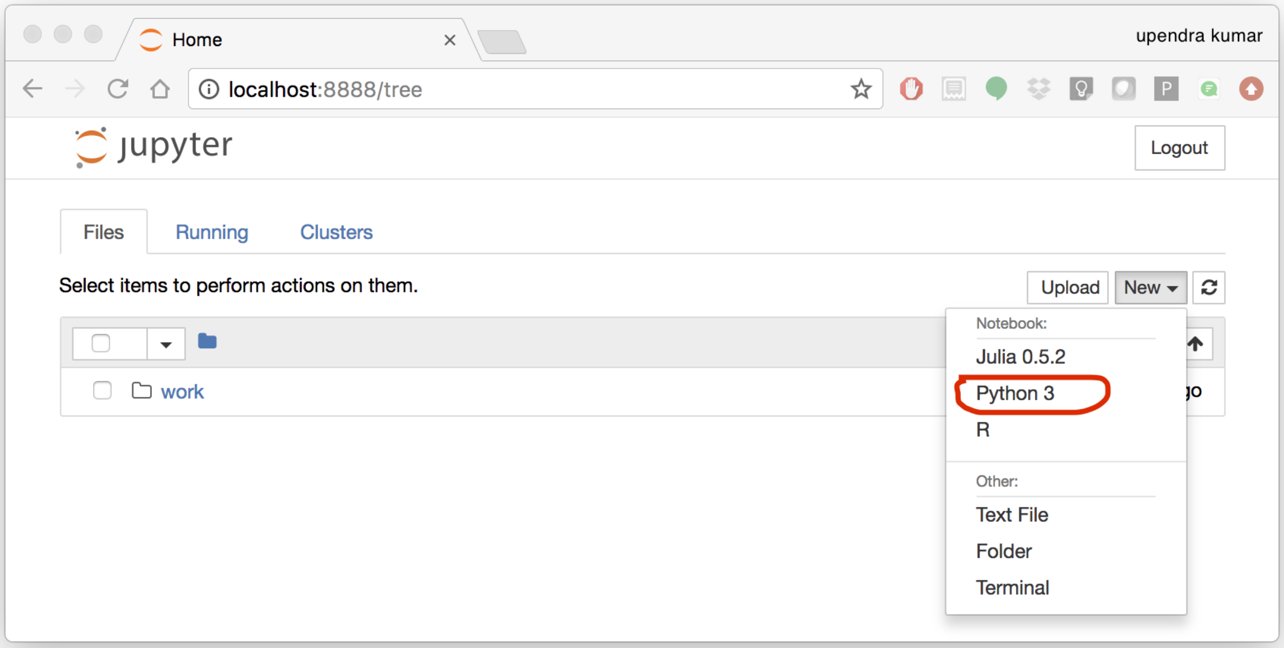
```
$ docker run --rm -p 8888:8888 jupyter/minimal-notebook
```

Once you've done that you should be greeted by your very own containerised Jupyter service!



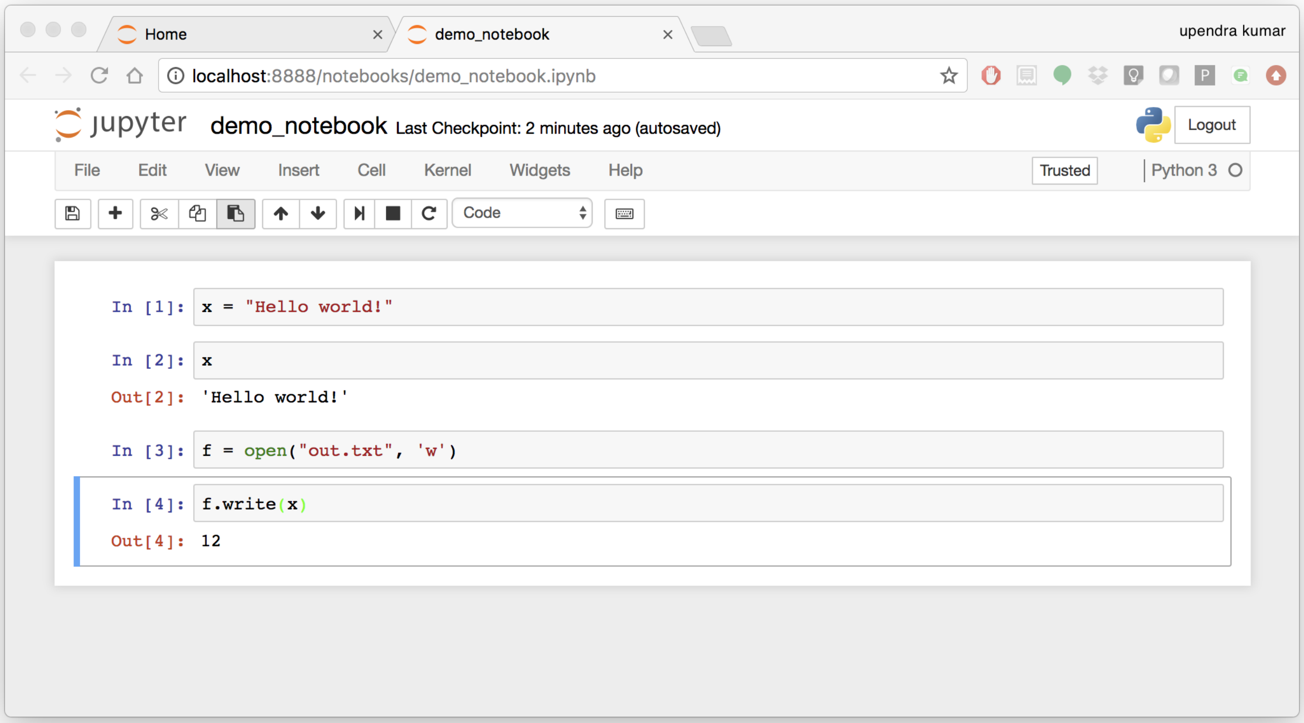
Select items to perform actions on them.

To create your first notebook, drill into the work directory and then click on the 'New' button on the right hand side and choose 'Python 3' to create a new Python 3 based Notebook.

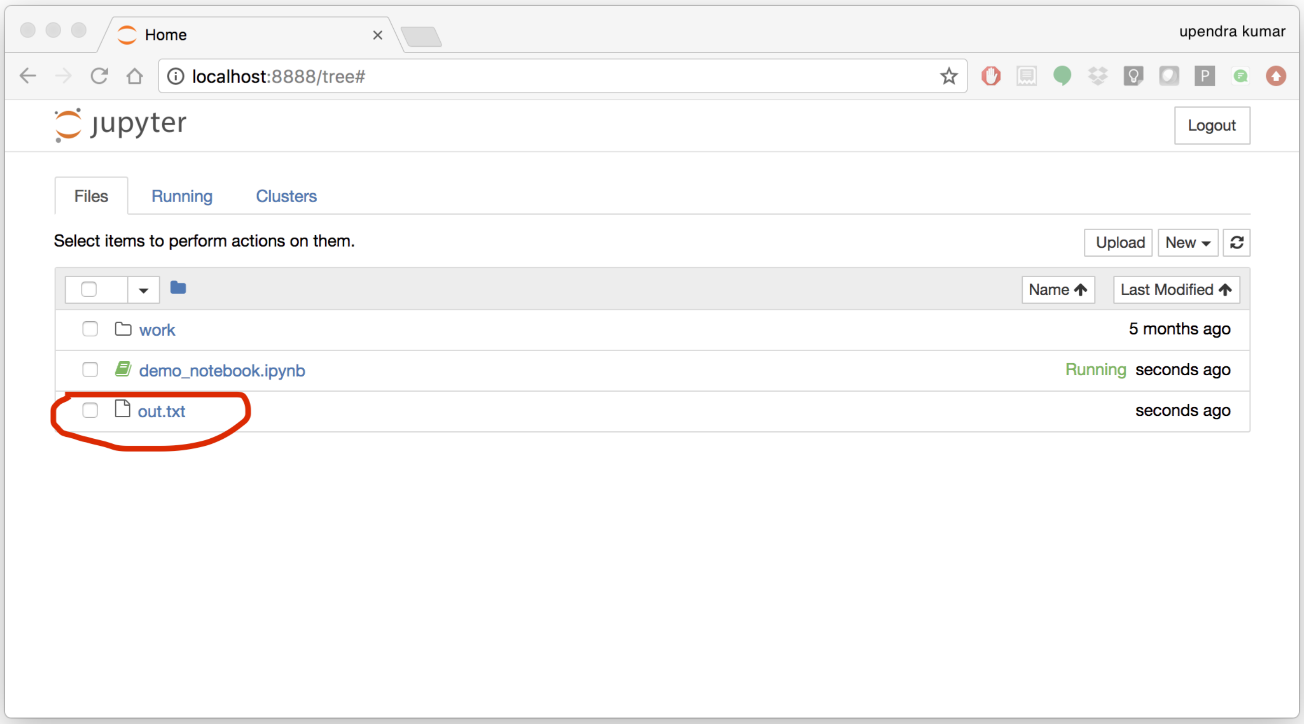


Select items to perform actions on them.

Now you can write your python code. Here is an example



```
In [1]: x = "Hello world!"

In [2]: x
Out[2]: 'Hello world!'

In [3]: f = open("out.txt", 'w')

In [4]: f.write(x)
Out[4]: 12
```



Select items to perform actions on them.

work		5 months ago
demo_notebook.ipynb	Running	seconds ago
out.txt		seconds ago

To mount the host directory inside the Jupyter notebook container, you must first grant the within-container notebook user or group (NB_UID or NB_GID) write access to the host directory

```
sudo chown 1000 <host directory>
```

you can run the command as below

```
$ docker run --rm -p 8888:8888 -v $PWD:/work -w /home/jovyan/work jupyter/minimal-
↪notebook
```

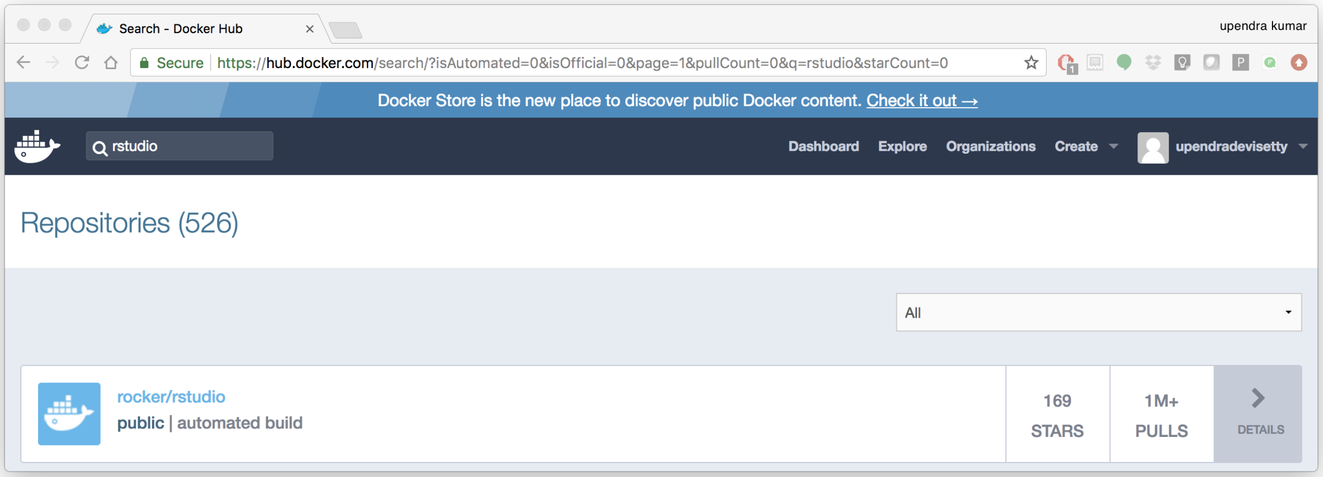
Tip: If you want to run *Jupyter-lab* instead of the default Jupyter notebook, you can do so by adding *jupyter-lab* at the end of the command.

More options for Datascience jupyter notebook - https://github.com/Paperspace/jupyter-docker-stacks/tree/master/datascience-notebook

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.

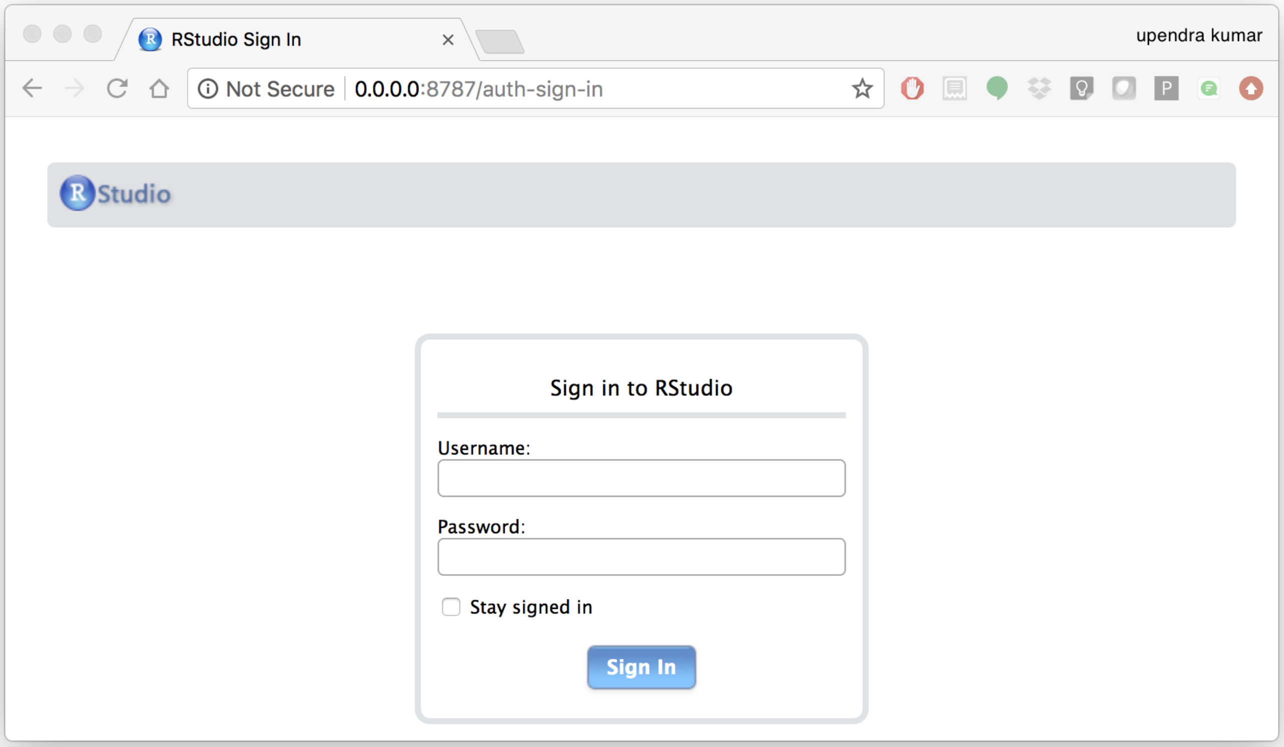
17.2 2. Launch a RStudio container

Next, we will see a Docker image from Rocker which will allow us to run RStudio inside the container and has many useful R packages already installed.

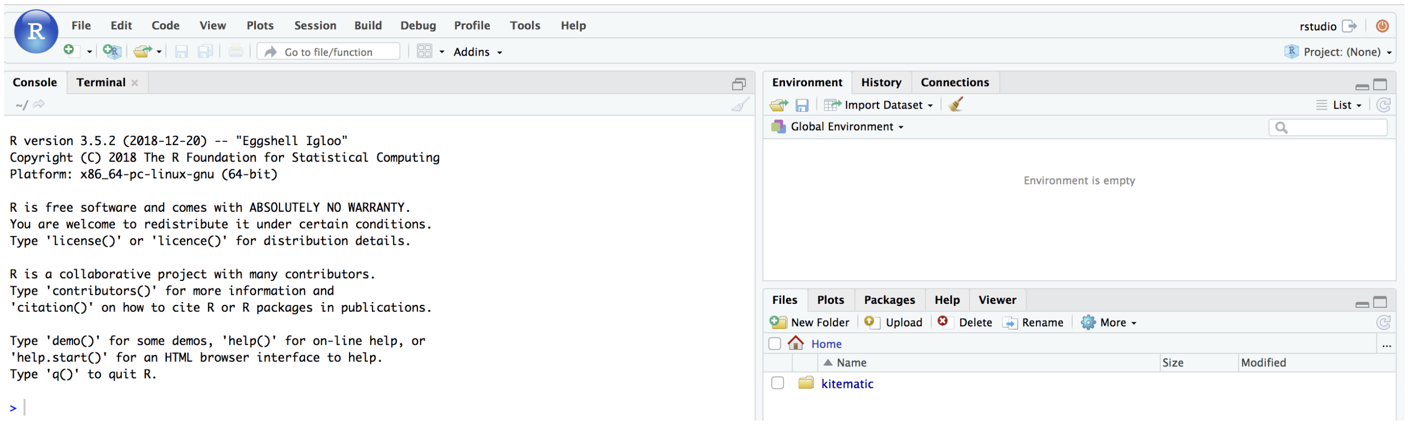

Search - Docker Hub
upendra kumar
Secure | https://hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=rstudio&starCount=0
Docker Store is the new place to discover public Docker content. Check it out →
rstudio
Dashboard
Explore
Organizations
Create
upendradevisetty
Repositories (526)
All
rocker/rstudio
public | automated build
169
STARS
1M+
PULLS
DETAILS

```
$ docker run --rm -d -e PASSWORD=rstudio1 -p 8787:8787 rocker/rstudio:3.5.2
```

The command above will lead RStudio-Server to launch invisibly. To connect to it, open a browser and enter http://localhost:8787, or <ipaddress>:8787 on cloud.



Tip: For the current Rstudio container, the default username is *rstudio* and the password is *rstudio1*. However you can override the disable the log-in with *-e DISABLE_AUTH=true* in place of *-e PASSWORD=rstudio1*.



```
R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

If you want to mount the host directory inside the Rstudio container, you can do as below

```
$ docker run -v $PWD:/data -w /data -p 8787:8787 -e DISABLE_AUTH=true --rm rocker/
↪rstudio:3.5.2
```

And navigate to the */data* inside the container using the file browser option in Rstudio.

An excellent R tutorial for reproducible research can be found here

17.3 3. Machine learning using Docker

In this simple example we'll take a sample dataset of fruits metrics (like size, weight, texture) labelled apples and oranges. Then we can predict the fruit given a new set of fruit metrics using scikit-learn's decision tree

You can find the above code in this github repo

1. Create a directory that consists of all the files

```
$ mkdir scikit_docker && cd scikit_docker
```

2. Create `requirements.txt` file—Contains python modules and has nothing to do with Docker inside the folder - `scikit_docker`.

```
numpy
scipy
scikit-learn
```

3. Create a file called `app.py` inside the folder— `scikit_docker`

```
from sklearn import tree
#DataSet
#[size,weight,texture]
X = [[181, 80, 44], [177, 70, 43], [160, 60, 38], [154, 54, 37],[166, 65, 40], [190,␣
↪90, 47], [175, 64, 39], [177, 70, 40], [159, 55, 37], [171, 75, 42], [181, 85, 43]]

Y = ['apple', 'apple', 'orange', 'orange', 'apple', 'apple', 'orange', 'orange',
↪'orange', 'apple', 'apple']

#classifier - DecisionTreeClassifier
```

(continues on next page)

(continued from previous page)

```
clf_tree = tree.DecisionTreeClassifier();
clf_tree = clf_tree.fit(X,Y);

#test_data
test_data = [[190,70,42],[172,64,39],[182,80,42]];

#prediction
prediction_tree = clf_tree.predict(test_data);

# Write output to a file
with open("output.txt", 'w') as fh_out:
        fh_out.write("Prediction of DecisionTreeClassifier:")
        fh_out.write(str(prediction_tree))
```

4. Create a Dockerfile that contains all the instructions for building a Docker image inside the project directory

```
# Use an official Python runtime as a parent image
FROM python:3.6-slim
MAINTAINER Upendra Devisetty <upendra@cyverse.org>
LABEL Description "This Dockerfile is used to build a scikit-learn's decision tree␣
↪image"

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

5. Create a Docker compose YAML file

```
version: '2'
services:
    datasci:
        build: .
        volumes:
            - .:/app
```

5. Now Build and Run the Docker image using *docker-compose up* command to predict the fruit given a new set of fruit metrics

```
$ docker-compose up
```

Use *docker-compose rm* to remove the container after docker-compose finish running

```
docker-compose rm
Going to remove scikitdocker_datasci_1
Are you sure? [yN] y
Removing scikitdocker_datasci_1 ... done
```

You will find the ouput file in the *scikit_docker* folder with the following contents

17.4 4. jupyter-repo2docker

jupyter-repo2docker is a tool to build, run, and push Docker images from source code repositories that run via a Jupyter server.

repo2docker fetches a repository (from GitHub, GitLab or other locations) and builds a container image based on the configuration files found in the repository. It can be used to explore a repository locally by building and executing the constructed image of the repository, or as a means of building images that are pushed to a Docker registry

In order to run *jupyter-repo2docker*, you first need to install it locally onto your computer

```
pip install jupyter-repo2docker
```

17.4.1 Usage

The core feature of *jupyter-repo2docker* is to fetch a git repository (from GitHub or locally), build a container image based on the specifications found in the repository & optionally launch the container that you can use to explore the repository.

Note: Docker needs to be running on your machine for this to work.

Let's take a simple example:

```
jupyter-repo2docker https://github.com/norvig/pytudes
```

After building (it might take a while!), it should output in your terminal something like:

```
    Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://0.0.0.0:36511/?token=f94f8fabb92e22f5bfab116c382b4707fc2cade56ad1ace0
```

Intermediate example:

Important: *repo2docker* looks for configuration files in the repository being built to determine how to build it. In general, repo2docker uses the same configuration files as other software installation tools, rather than creating new custom configuration files. Here is a list of supported configuration files (roughly in the order of build priority). For this example, I have already created *requirements.txt* file.

```
jupyter-repo2docker https://github.com/upendrak/keras_wine
```

After building (it might take a while for the first time), it should output in your terminal something like this.

```
Or copy and paste one of these URLs:
http://127.0.0.1:55869/?token=103a016fd2e6b04ce7108ce19d078ecef74475dfacce7bd3
```

If you copy paste that URL into your browser you will see a Jupyter Notebook with the contents of the repository you had just built!

For more information on how to use repo2docker, see the usage guide.

17.5 5. Binder

Binder allows you to create custom computing environments that can be shared and used by many remote users. It is powered by BinderHub, which is an open-source tool that deploys the Binder service in the cloud. One-such deployment lives here, at mybinder.org, and is free to use. For more information about the mybinder.org deployment and the team that runs it, see About mybinder.org.

Note: Binder is a research pilot, whose main goal is to understand usage patterns and workloads for future evolution and development. It is not a service that can be relied on for critical operations.

Continuing the *keras_wine* github example, let's create a Binder button to make it easy for the users to launch the notebooks interactively from the README in the github repo of *keras_wine*.

```
[![Binder](https://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/gh/upendrak/
↪keras_wine/master)
```

Clicking the button will make the jupyer notebook interactive.

CHAPTER 18

Booting a CyVerse Atmosphere instance

In this session, we will walk through how to start up a running computer (an "instance") on the CyVerse Atmosphere Cloud service. Here is the Atmosphere manual if you are interested in learning more about CyVerse Atmosphere

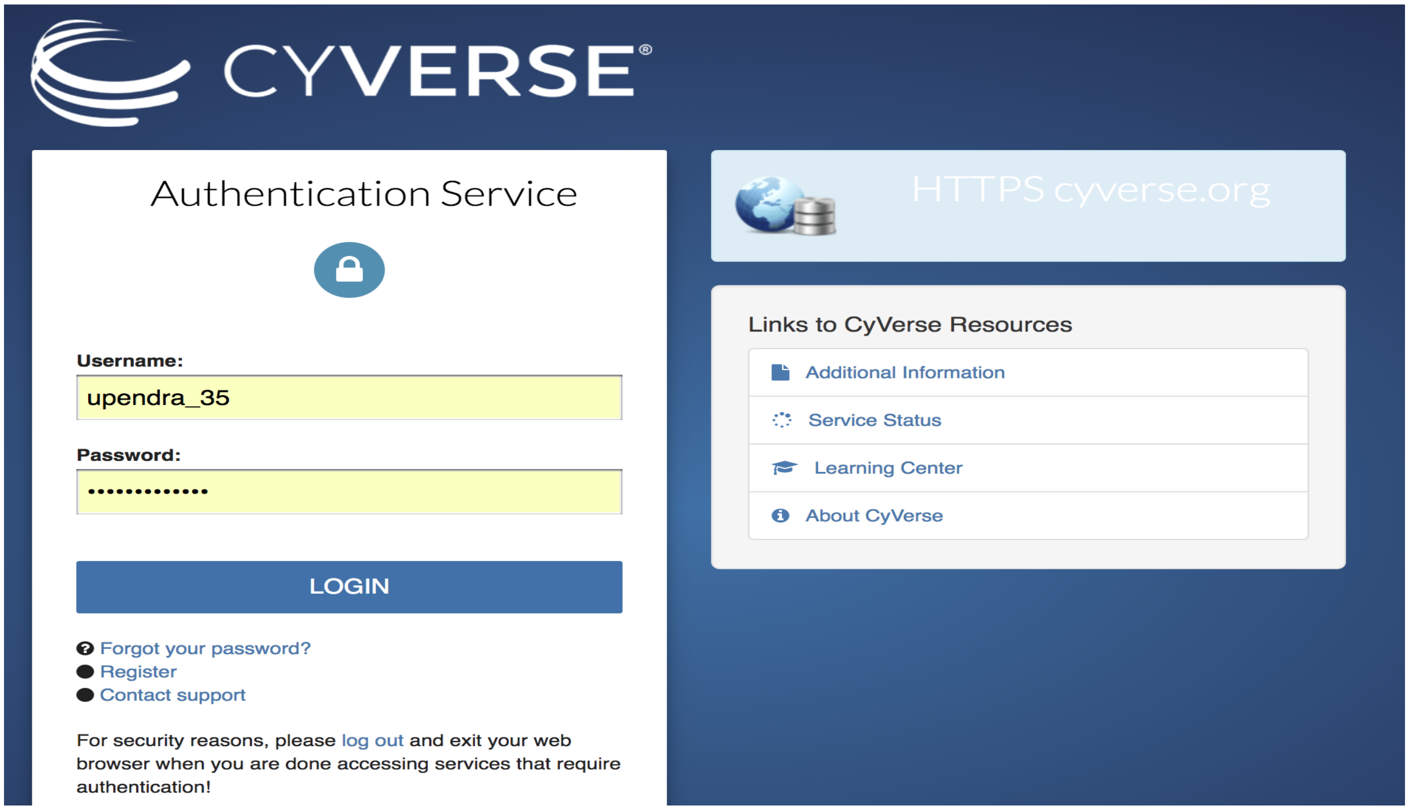
Below, we've provided screenshots of the whole process. You can click on them to zoom in a bit. The important areas to fill in are highlighted.

First, go to the Atmosphere application and then click **login**

Important: You will need to have access to the Atmosphere workshop cloud. If you are unable 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.


Authentication Service

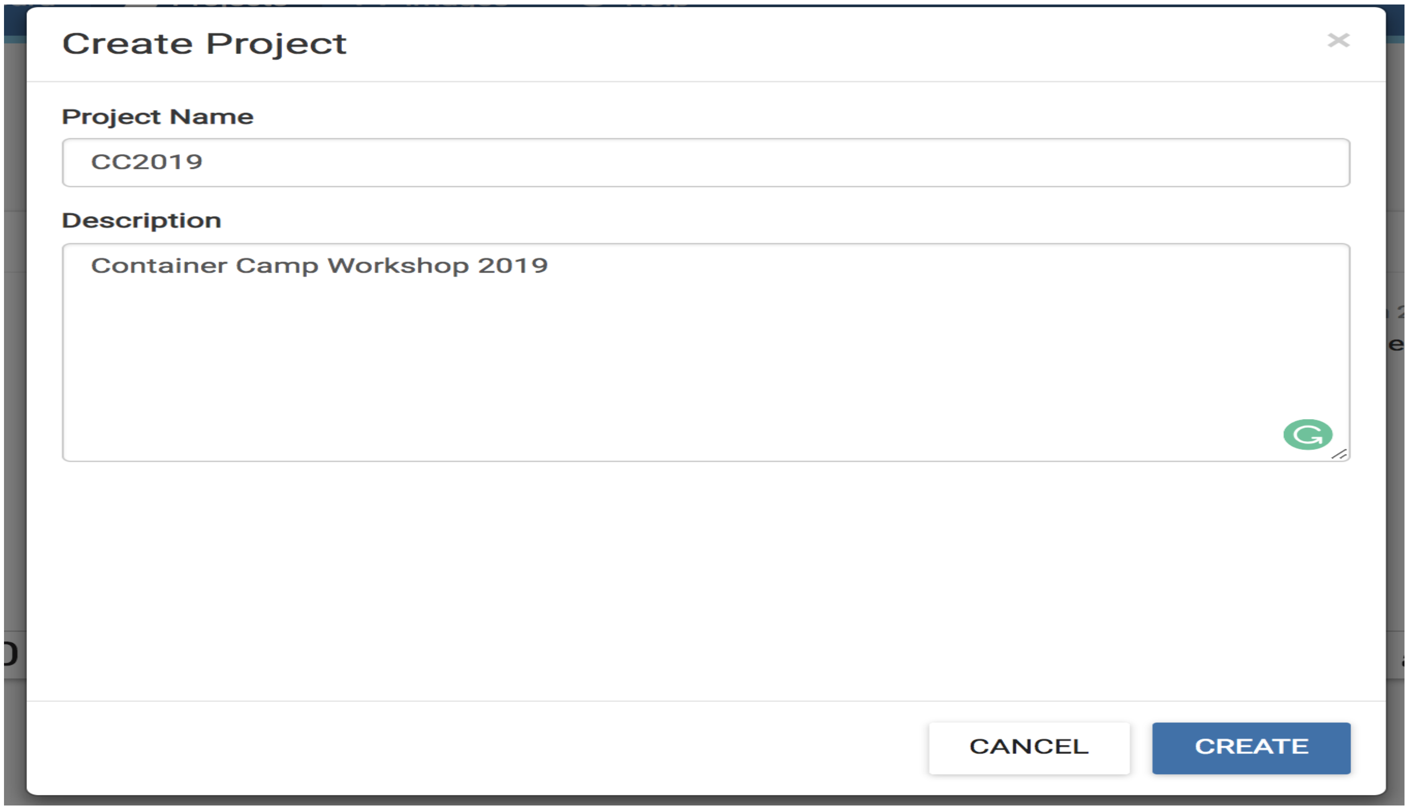
Username: upendra_35

Password: •••••••••••••

For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

2. Select Projects and **Create New Project**

- Now, this is something you only need to do once.
- We'll do this with Projects, which gives you a bit of a workspace in which to keep things that belong to *you*.
- Click on the **Projects** tab on the top and then click **CREATE NEW PROJECT**
- Enter the name **CC2019** into the Project Name box, and something simple like **Container Camp Workshop 2019** into the description. Then click **create**.


Create Project

Project Name

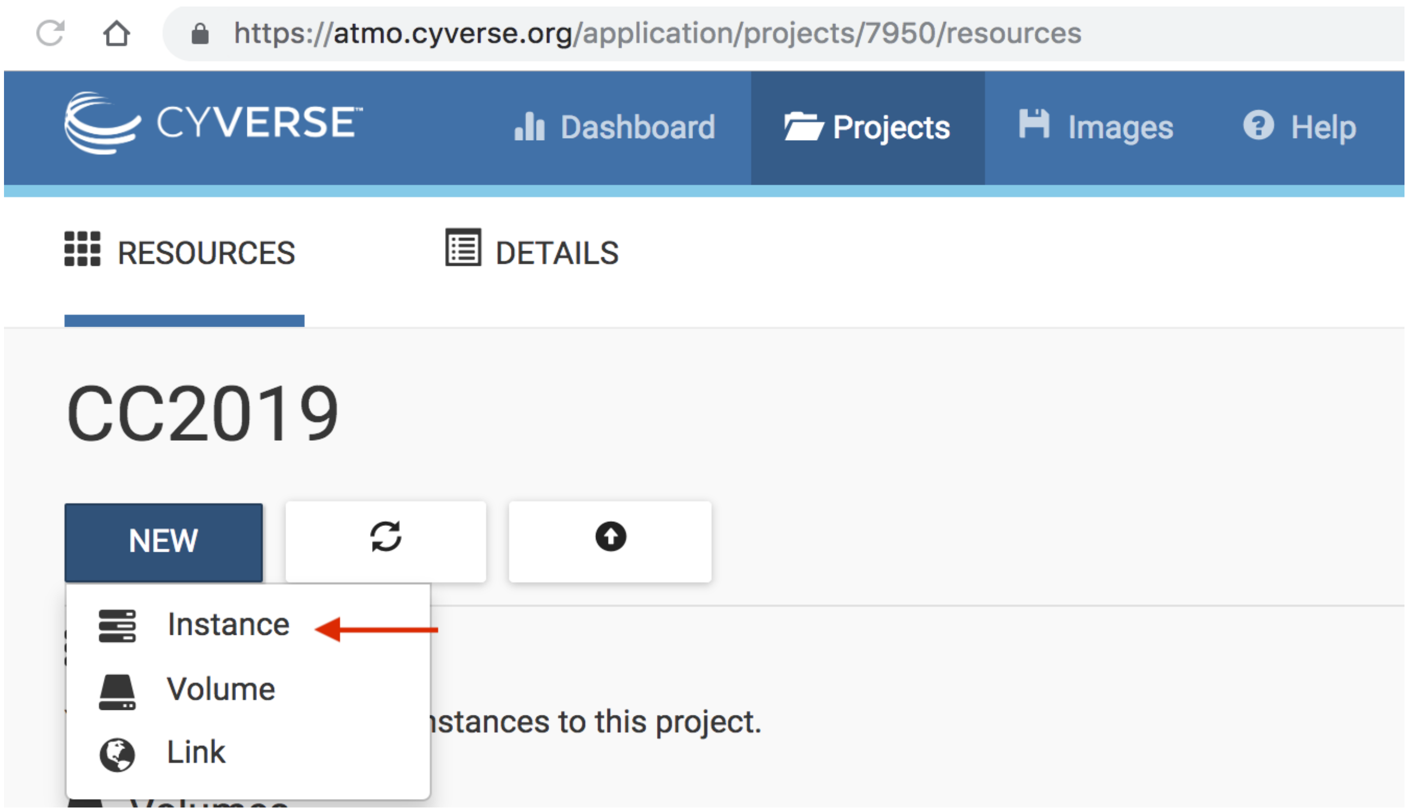
CC2019

Description

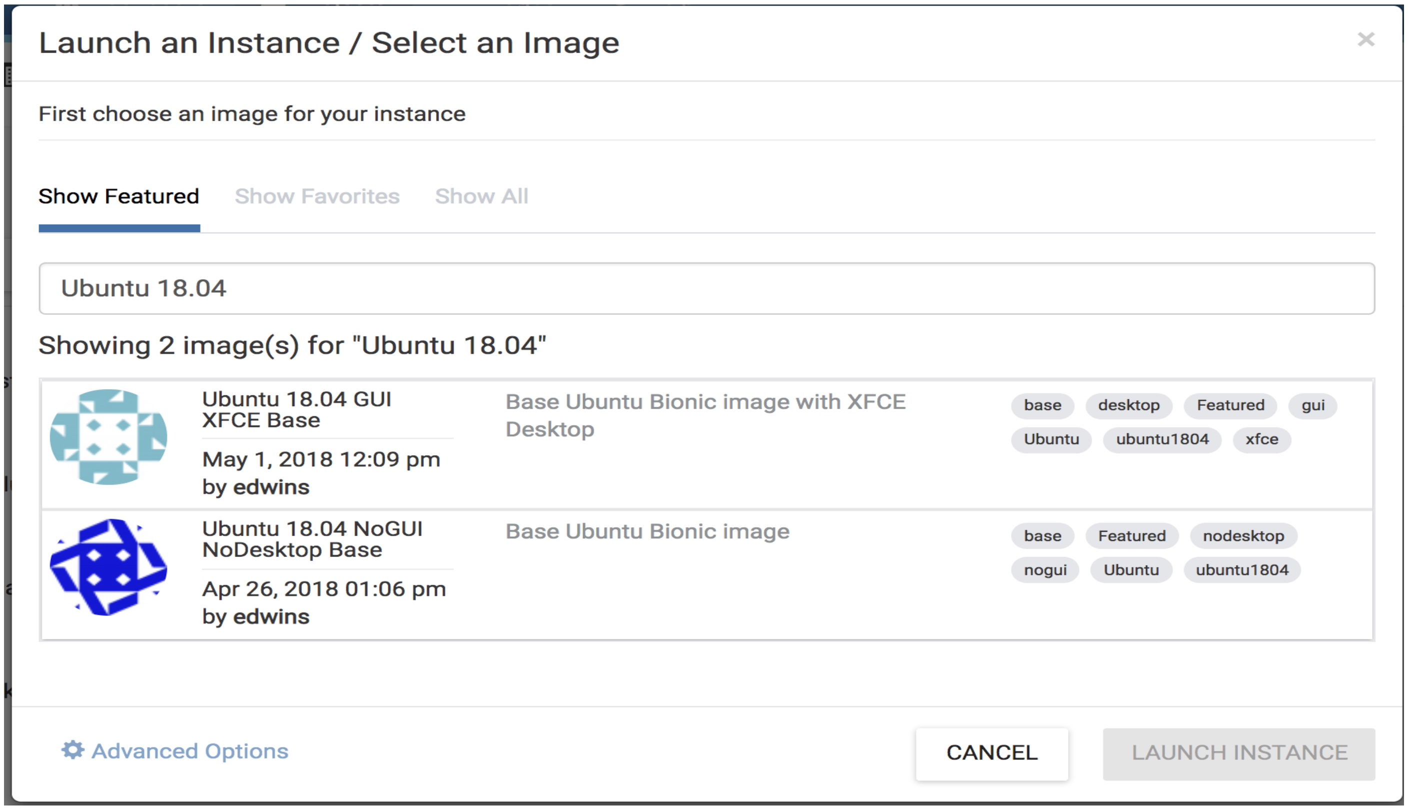
Container Camp Workshop 2019

3. Select the newly created project

- Click on your newly created project!
- Now, click **New** and then **Instance** from the dropdown menu to start up a new virtual machine.


https://atmo.cyverse.org/application/projects/7950/resources
CYVERSE
Dashboard
Projects
Images
Help
RESOURCES
DETAILS
CC2019
NEW
Instance
Volume
Link
nstances to this project.

- Find the **Ubuntu 18.04** image, click on it

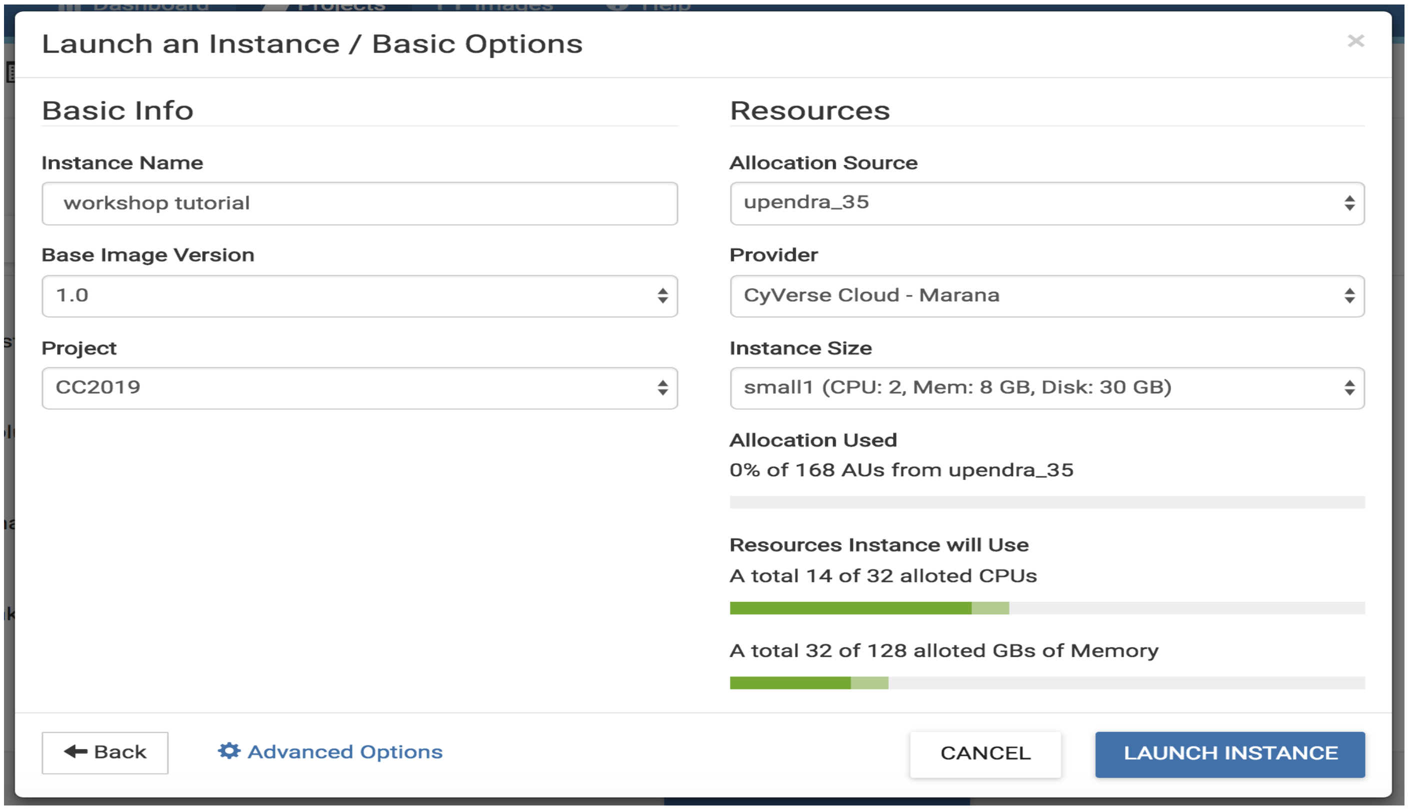


Showing 2 image(s) for "Ubuntu 18.04"

	Ubuntu 18.04 GUI XFCE Base May 1, 2018 12:09 pm by edwins	Base Ubuntu Bionic image with XFCE Desktop	base, desktop, Featured, gui, Ubuntu, ubuntu1804, xfce
	Ubuntu 18.04 NoGUI NoDesktop Base Apr 26, 2018 01:06 pm by edwins	Base Ubuntu Bionic image	base, Featured, nodesktop, nogui, Ubuntu, ubuntu1804

- Name it something simple such as **workshop tutorial** and select **small1 (CPU: 2, Mem: 8GB, Disk: 30GB)**.

- Leave rest of the fields as default.



Launch an Instance / Basic Options

Basic Info

Instance Name: workshop tutorial

Base Image Version: 1.0

Project: CC2019

Resources

Allocation Source: upendra_35

Provider: CyVerse Cloud - Marana

Instance Size: small1 (CPU: 2, Mem: 8 GB, Disk: 30 GB)

Allocation Used

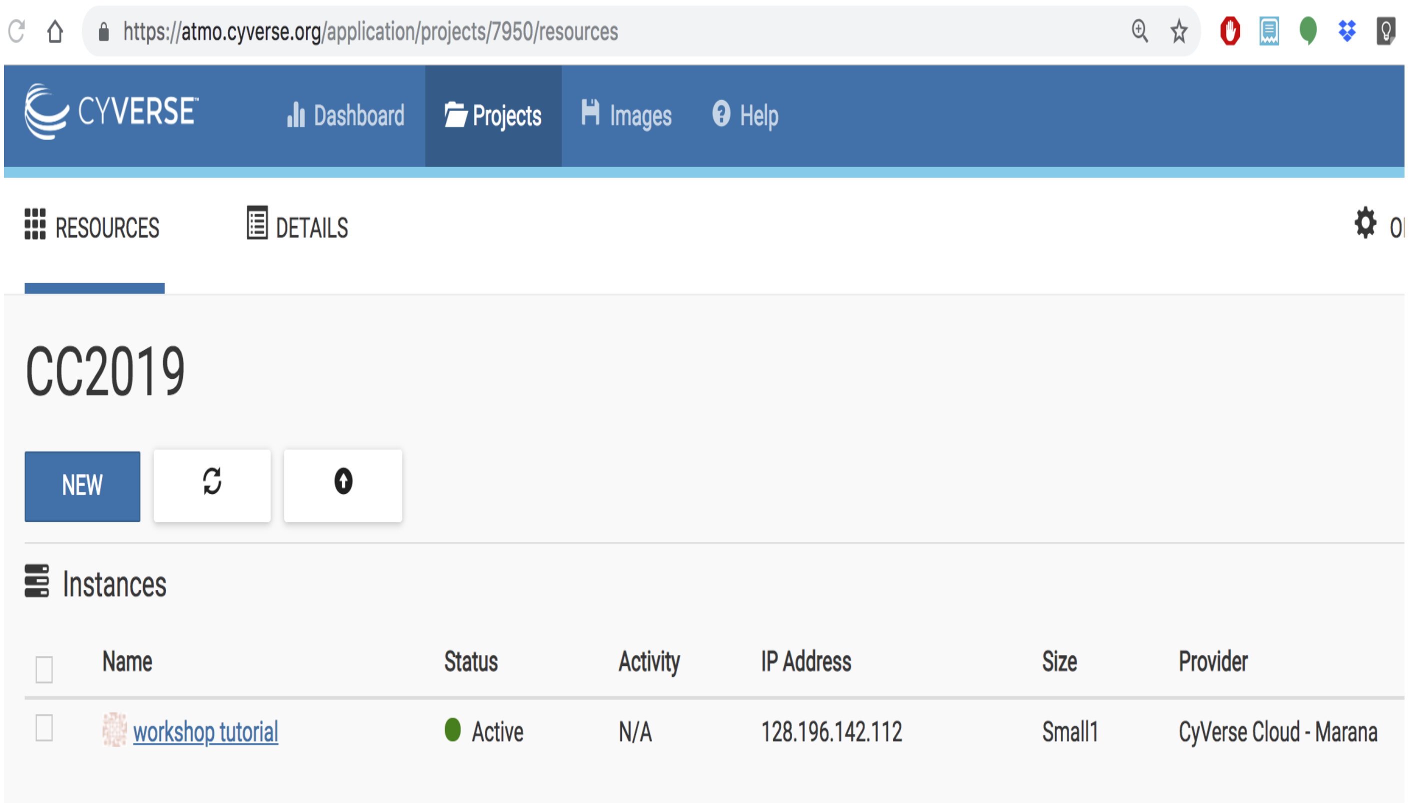
0% of 168 AUs from upendra_35

Resources Instance will Use

A total 14 of 32 alloted CPUs

A total 32 of 128 alloted GBs of Memory

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

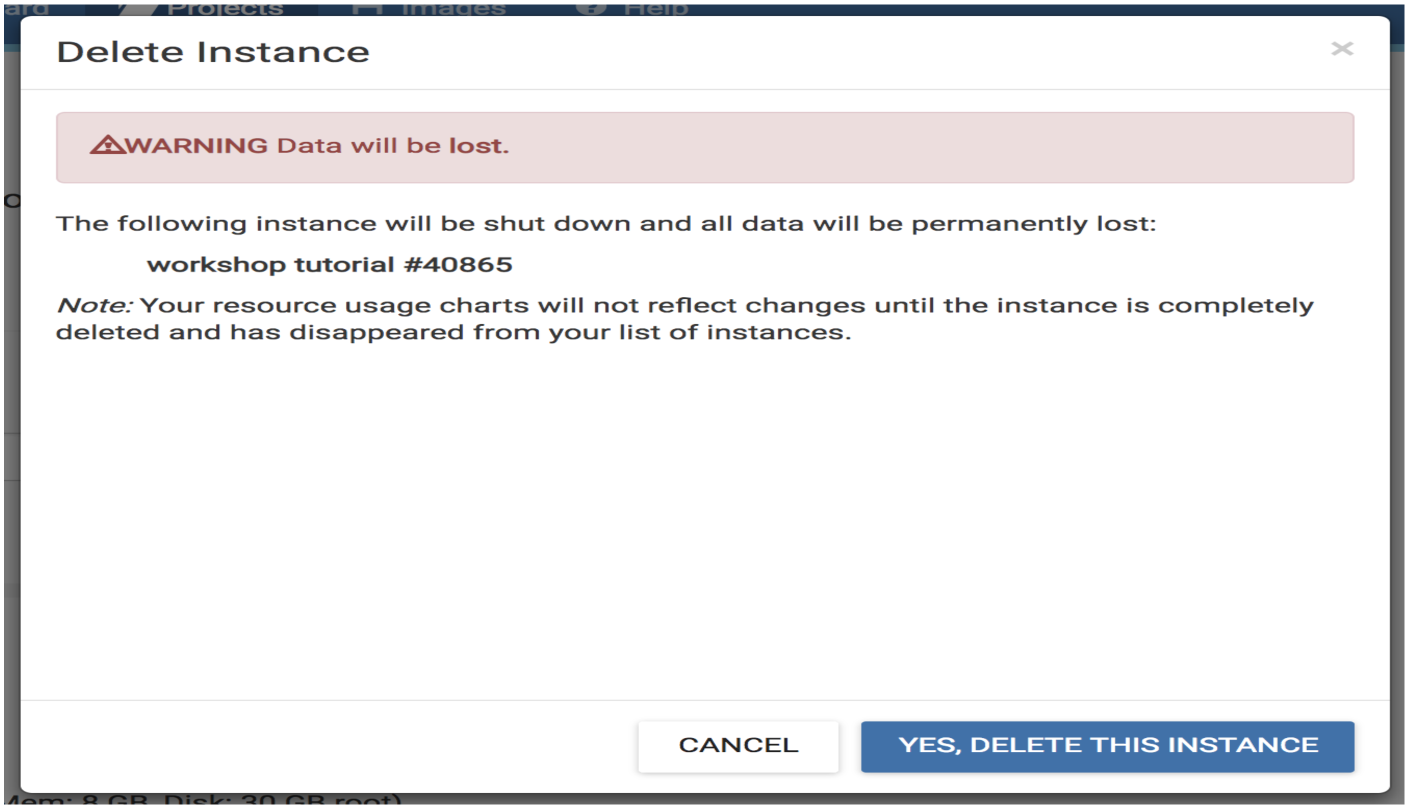


	Name	Status	Activity	IP Address	Size	Provider
	workshop tutorial	● Active	N/A	128.196.142.112	Small1	CyVerse Cloud - Marana

- Click on your new instance to get more information!
- Now, you can either click **Open Web Shell**, *or*, you can ssh in with your CyVerse username on the IP address of the machine. For using **Open Web Shell**, click on the name of the instance and it will take you to the next screen. You'll find the **Open Web Shell** underneath the Actions menu on the right.

Deleting your instance

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


Delete Instance

⚠WARNING Data will be lost.

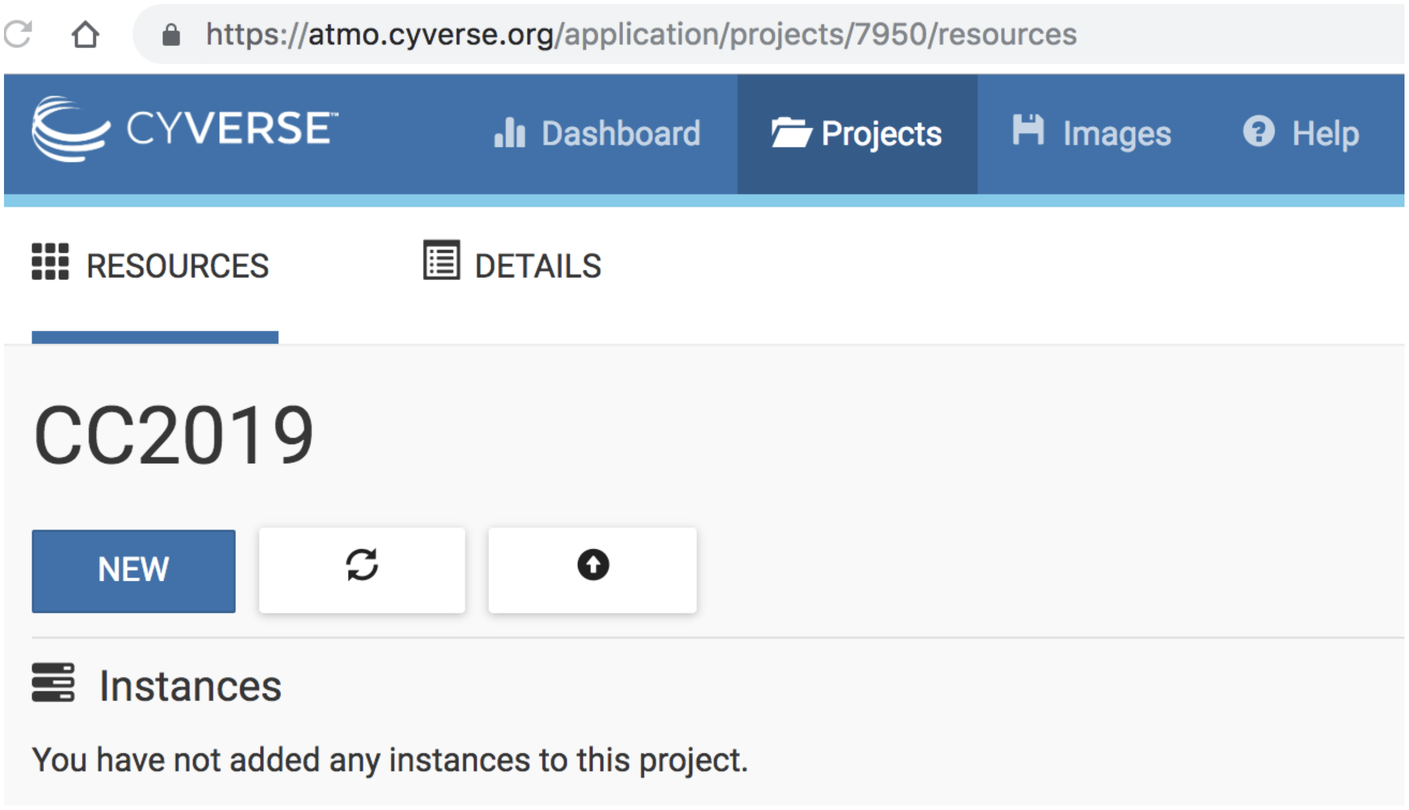
The following instance will be shut down and all data will be permanently lost:

workshop tutorial #40865

Note: Your resource usage charts will not reflect changes until the instance is completely deleted and has disappeared from your list of instances.

Before deleting an instance make sure you backup your data, once the instance is deleted, there is no way you can get the data back. It is recommended to attach the volume to the instance and do your analysis there.

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


https://atmo.cyverse.org/application/projects/7950/resources
CYVERSE
Dashboard
Projects
Images
Help
RESOURCES
DETAILS
NEW

CC2019

Instances

You have not added any instances to this project.

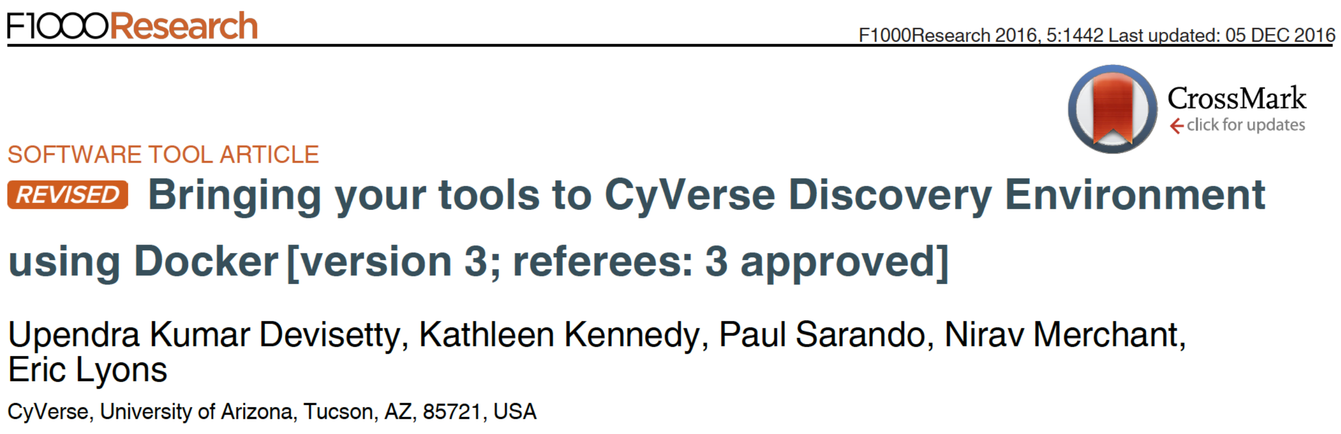
Note: It is advisable to delete the machine if you are not planning to use it in future to save valuable resources. However if you want to use it in future, you can suspend it.

CHAPTER 19

Deploying apps in CyVerse Discovery Environment

The CyVerse Discovery Environment (DE) provides a simple yet powerful web portal for managing data, analyses, and workflows. The DE uses containers (both Docker and Singularity) to support customizable, non-interactive, interactive reproducible workflows using data stored in the CyVerse Data Store.

This paper will guide you to bring your dockerized tools into CyVerse DE.



F1000Research 2016, 5:1442 Last updated: 05 DEC 2016

SOFTWARE TOOL ARTICLE

REVISED Bringing your tools to CyVerse Discovery Environment using Docker [version 3; referees: 3 approved]

Upendra Kumar Devisetty, Kathleen Kennedy, Paul Sarando, Nirav Merchant, Eric Lyons

CyVerse, University of Arizona, Tucson, AZ, 85721, USA

Important: Significant changes have been made as to how you can bring your tools into DE and so we are working on a separate paper that will show all those changes. Meanwhile you can follow the below tutorial for integrating your tools.

Here are the basic steps for deploying Docker images as apps in DE. For this tutorial I am going to show an example of Tensor image classifier

- *Build and test your Docker images*
- *Push your Docker image to Dockerhub*

- *Add Docker images as tool in DE*
- *Create an App UI for the tool in DE*
- *Test the app using appropriate test data in DE*

Warning: If you already have your own Docker image or a Docker image of interest is already hosted on a public registry(s) (Dockerhub or quay.io or some other public repository), then you can skip to Step 3

1. Build and test your Docker images

The first step is to dockerize your tool or software of interest. Detailed steps of how to dockerize your tool and test your dockerized images can be found in sections intro to docker and advanced docker.

For this tutorial I will use the `tensorflow image classifier` docker image that I built using this code.

Building the Docker image from the Dockerfile

```
$ git clone https://github.com/upendrak/tensorflow_image_classifier && cd tensorflow_
↪image_classifier

$ docker build -t tensorflow_up:1.0 .
```

Testing Docker image with test data

```
$ docker run --rm -v $(pwd):/data -w /data tensorflow_up:1.0 sample_data/16401288243_
↪36112bd52f_m.jpg
```

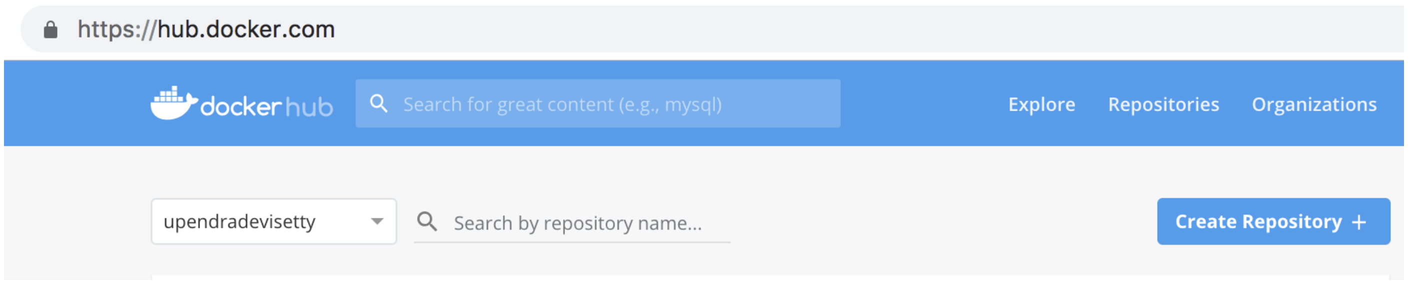
This generates a file called *16401288243_36112bd52f_m.out* that consits of classification percentages such as

```
daisy (score = 0.99785)
bee (score = 0.00009)
speedboat (score = 0.00008)
mitten (score = 0.00006)
sulphur butterfly, sulfur butterfly (score = 0.00004)
```

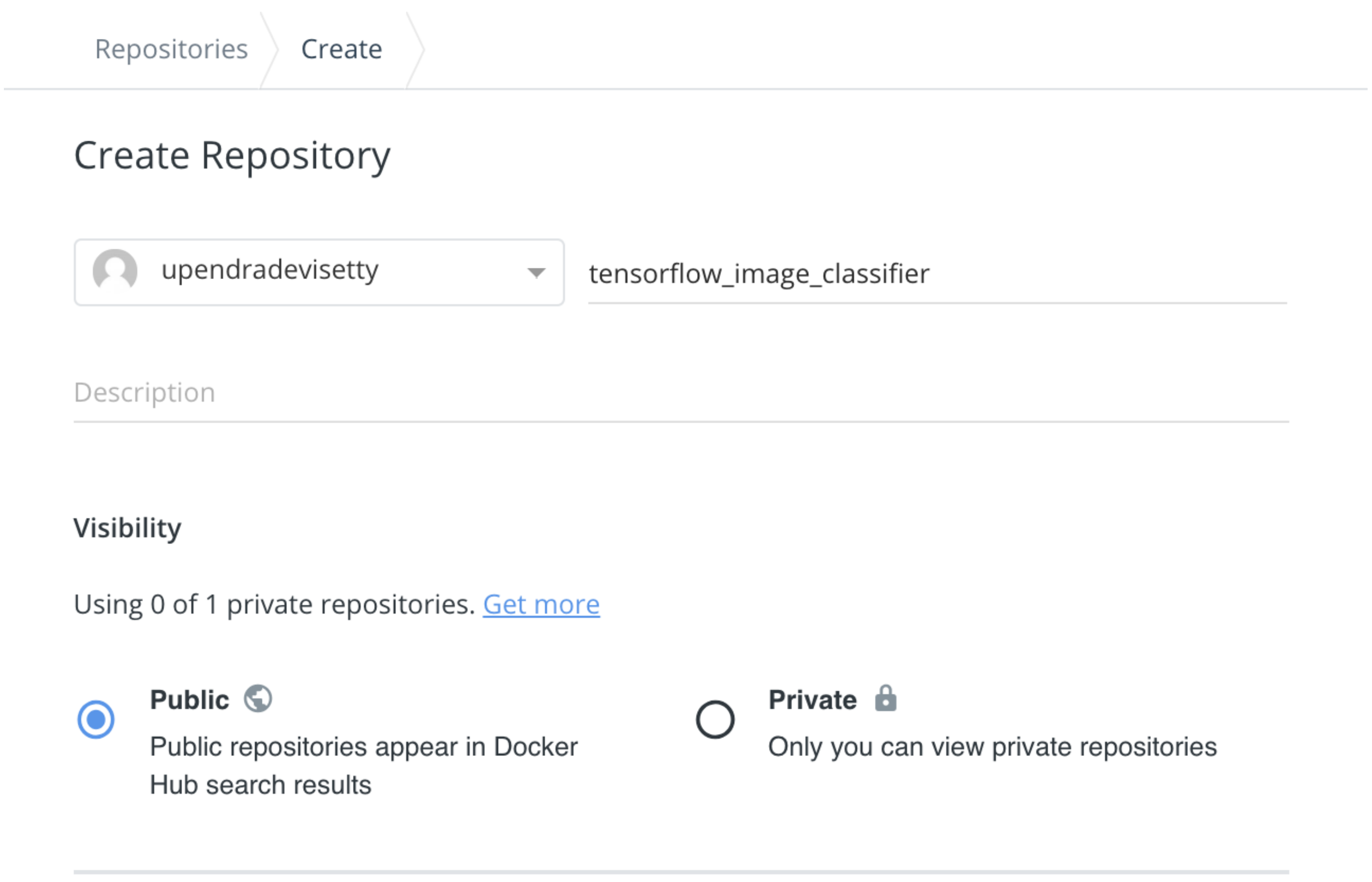
2. Push your Docker image to public repositories

Once the Docker image works as expected then either you set-up an automated build (recommended) or directly push the build Docker image to dockerhub. Here are the brief steps for automated build. See Advanced Docker section for more details.

2.1. Login to hub.docker.com and select Create Repository



2.2. Give a name to the repository. In here, I have given *tensorflow_image_classifier* as the name


Repositories Create

Create Repository

upendradevisetty tensorflow_image_classifier

Description

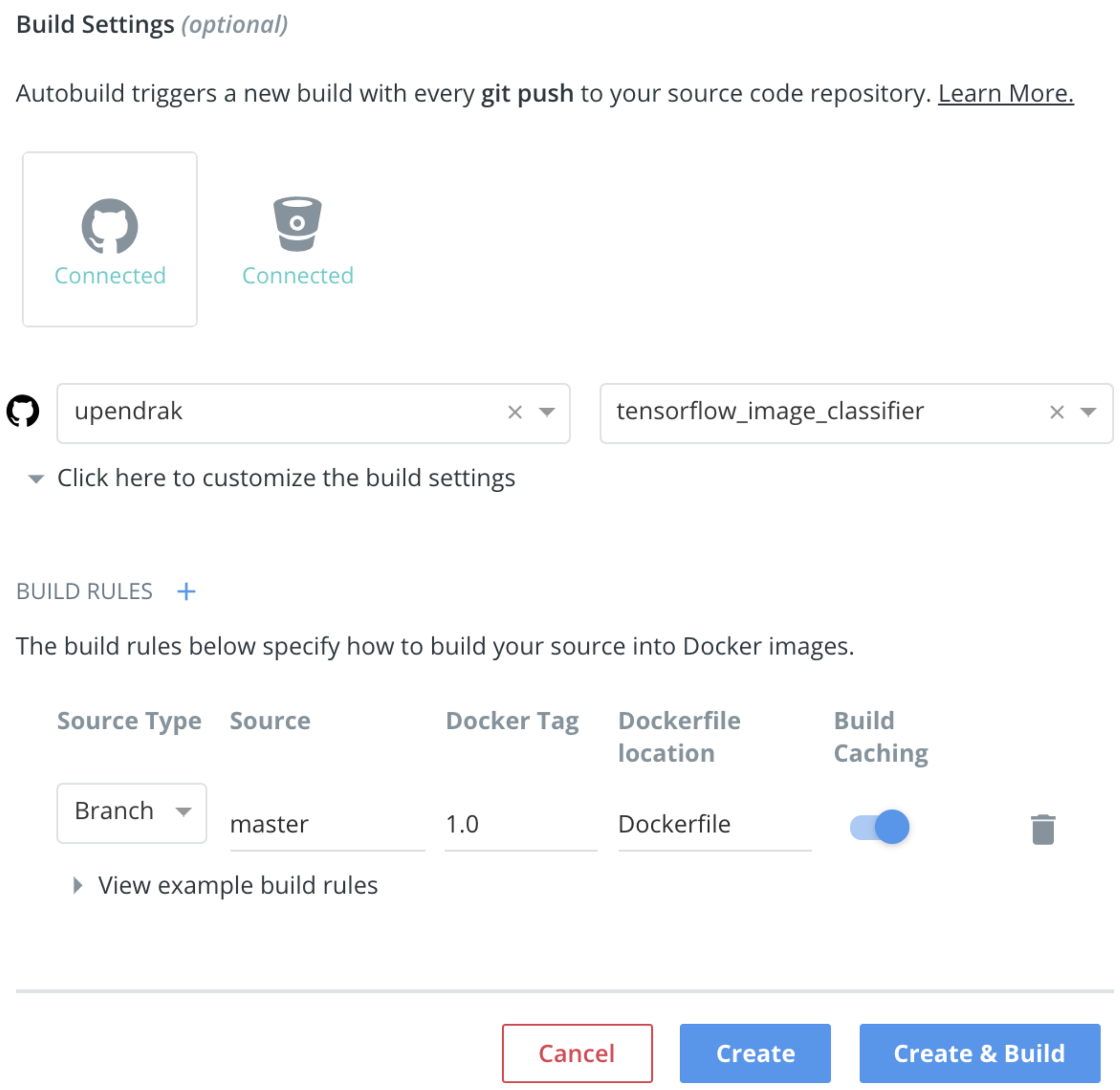
Visibility

Using 0 of 1 private repositories. Get more

[✓ Public] Public repositories appear in Docker Hub search results

Private Only you can view private repositories

2.3. Use the default visibility (Public in this case). Under Build settings, click the github octocat symbol which will ask you to authenticate github. Upon authentication, you'll be able to select the *tensorflow_image_classifier* github repo. Under Build rules, keep the source type as Branch, source as master, Docker Tag as 1.0 and the rest as defaults. Finally click "Create and Build" to start the building process


Build Settings *(optional)*

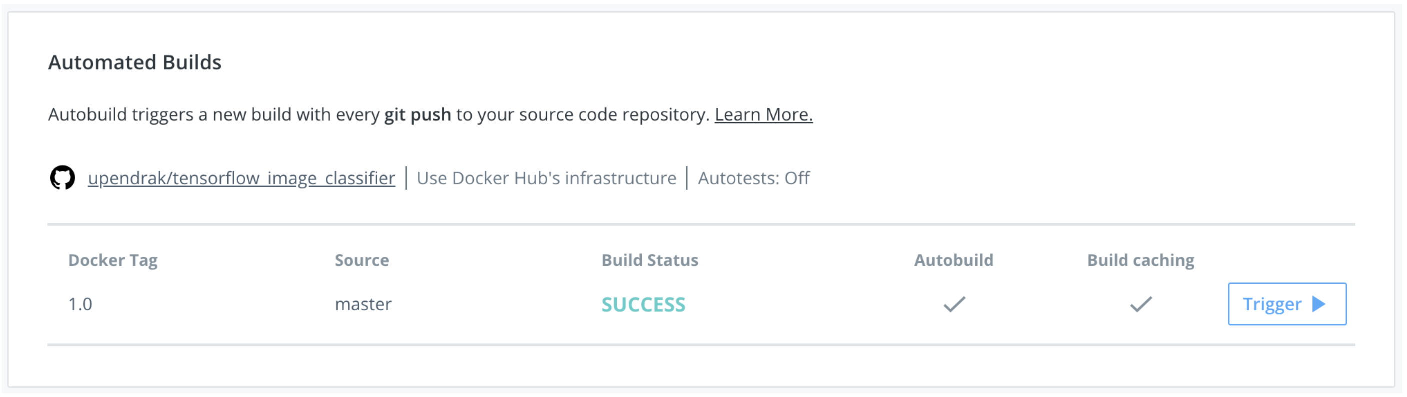
Autobuild triggers a new build with every **git push** to your source code repository. Learn More.

Connected
Connected
upendrak
tensorflow_image_classifier
Click here to customize the build settings
BUILD RULES +

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Caching
Branch	master	1.0	Dockerfile	●

2.4. It takes few minutes to hours (depending on the size of the image) and finally when everything works well, you'll see the **SUCCESS** message as shown here


Automated Builds

Autobuild triggers a new build with every **git push** to your source code repository. Learn More.

upendrak/tensorflow_image_classifier | Use Docker Hub's infrastructure | Autotests: Off

Docker Tag	Source	Build Status	Autobuild	Build caching	
1.0	master	SUCCESS	✓	✓	Trigger ▶

Here is the docker image built using automated build for the tensorflow image classifier on Dockerhub

3. Add Docker images as tool in DE

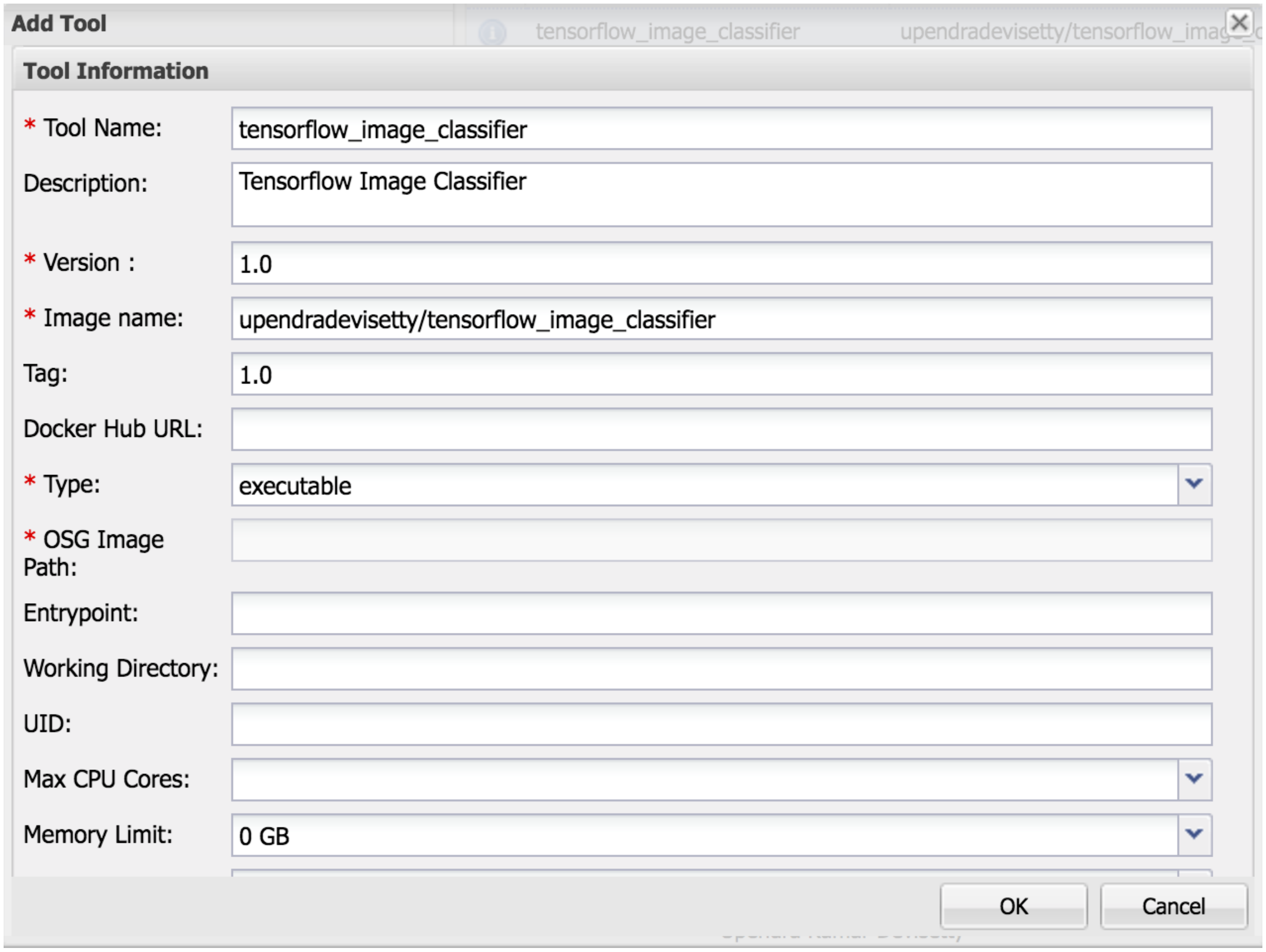
All tools now run installed as Docker images in the DE. Once the software is dockerized and available as Docker images on dockerhub then you can add those docker images as a tool in DE.

Warning: Check if the tool and correct version are already installed in the DE by following the steps below.

- Log in to the Discovery Environment by going to https://de.cyverse.org/de/, entering your CyVerse username and password, and clicking LOGIN. If you have not already done so, you will need to sign up for a CyVerse account.
- Click the `Apps` window to open the Apps window.
- Click the `Manage Tools` button on the top-right of the Apps window.
- In the search tools field, enter the first few letters of the tool name and then click enter.
- If the tool is available then you can skip to skip to step 3 for creating a UI for that tool.

If the tool is not available in DE then do the following:

- Click open the `Tools` tab in `Manage Tools` window and then click `Add tools` button
- Then enter the fields about your tool and then click "Ok".
  - Tool Name: It should be the name of the tool. For example "tensorflow_image_classifier".
  - Description: A short Description about the tool. For example "Tensorflow image classifier".
  - Version: What is the version number of the tool. For example "1.0".
  - Image name: Name of the Docker image on dockerhub or quay.io. For example "upendradevisetty/tensorflow_image_classifier".
  - Tag: What is the tag of your Docker image. This is optional but is highly recommended. If non specified, it will pull the default tag `latest`. If the `latest` tag is not avaiable the tool integration will fail. For example "1.0"
  - Entrypoint: Do you want a entrypoint for your Docker image? This optional.
  - Docker Hub URL: URL of the Dockerhub docker image. Option but is recommended. In this example "".


Add Tool

Tool Information

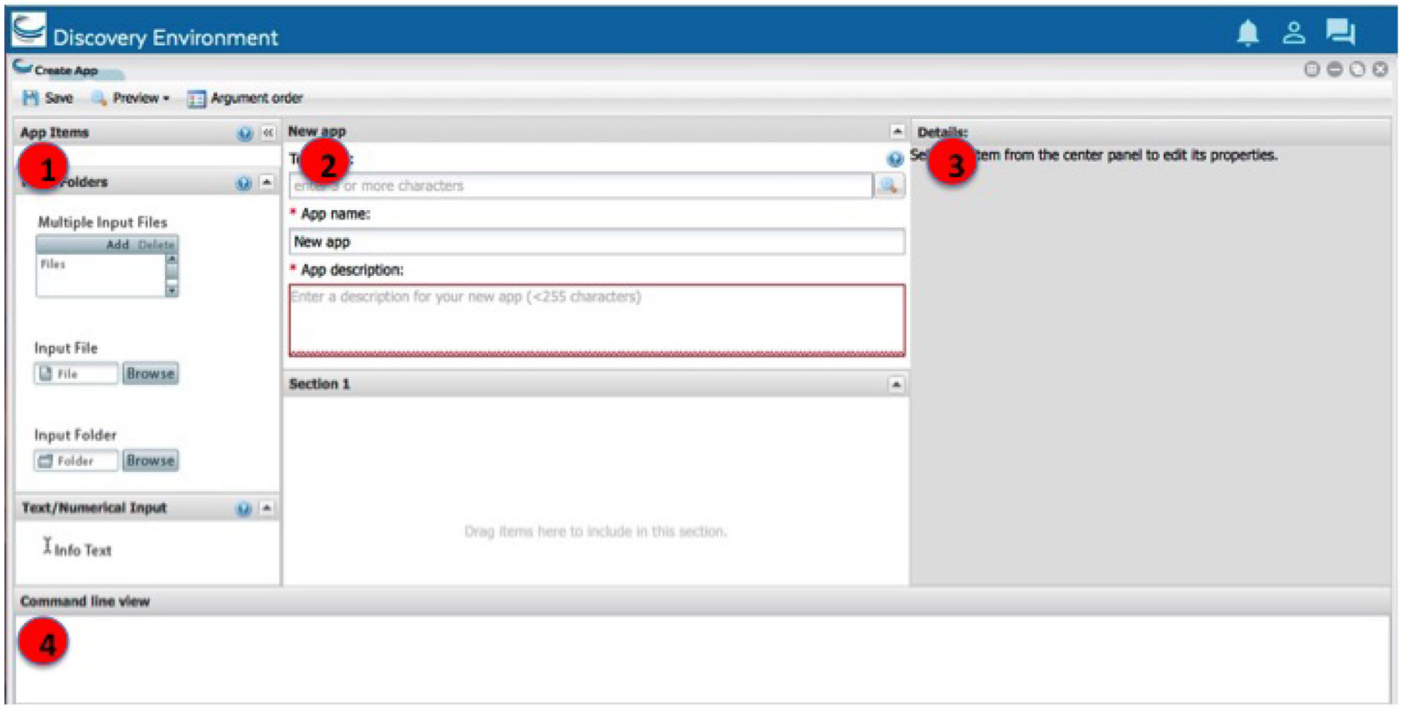
* Tool Name:	tensorflow_image_classifier
Description:	Tensorflow Image Classifier
* Version :	1.0
* Image name:	upendradevisetty/tensorflow_image_classifier
Tag:	1.0
Docker Hub URL:	
* Type:	executable
* OSG Image Path:	
Entrypoint:	
Working Directory:	
UID:	
Max CPU Cores:	
Memory Limit:	0 GB

- If there is no error message, you have successfully integrated the tool.

4. Create an App UI for the tool in DE

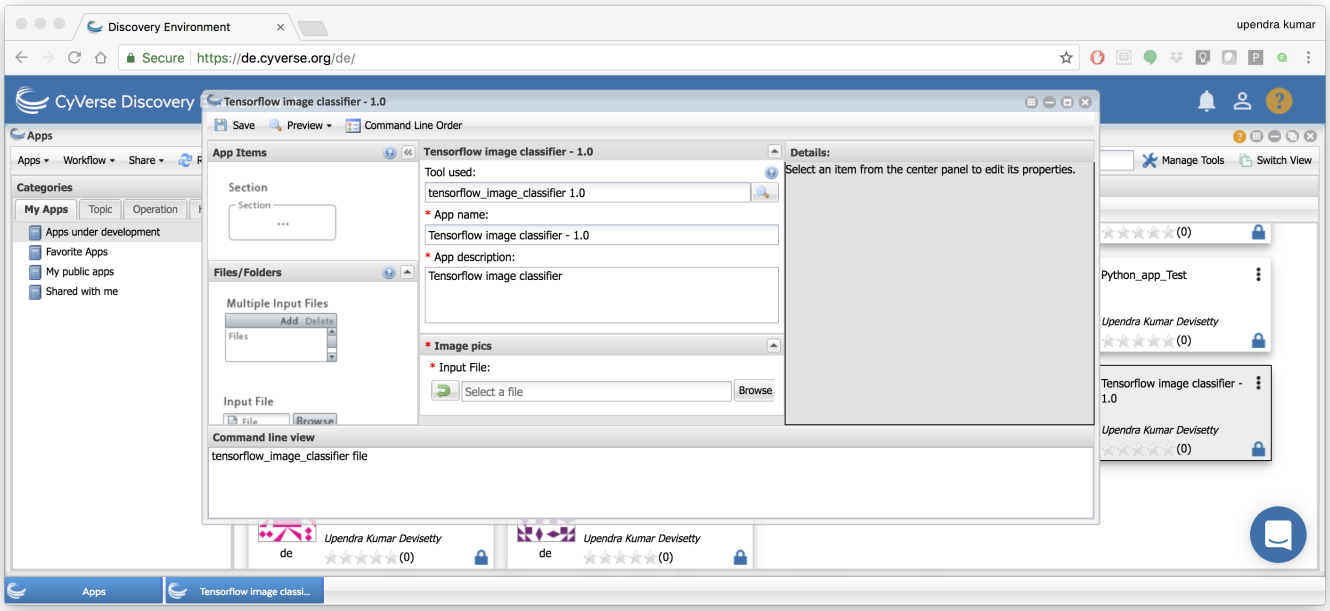
Once the Dockerized tool is added, you can create the app UI for the tool. The `Create App` window consists of four distinct sections:

- The first section contains the different app items that can be added to your interface. To add an app item, select the one to use (hover over the object name for a brief description) and drag it into position in the middle section.
- The second section is the landing place for the objects you dragged and dropped from the left section, and it updates to display how the app will look when presented to a user.
- The third section (Details) displays all of the available properties for the selected item. As you customize the app in this section, the middle section updates dynamically so you can see how it will look and act.
- Finally, the fourth section at the bottom (Command line view) contains the command-line commands for the current item's properties. As you update the properties in the Details section, the command-line view updates as well to let you make sure that you are passing the correct arguments in the correct order.



Note: Creating a new app interface requires that you know how to use the tool. With that knowledge, you create the interface according to how you want options to be displayed to a user.

Here is an example of the `Tensorflow image classifier - 1.0` app UI in DE



Tool used: tensorflow_image_classifier 1.0

App name: Tensorflow image classifier - 1.0

App description: Tensorflow image classifier

Command line view:

```
tensorflow_image_classifier file
```

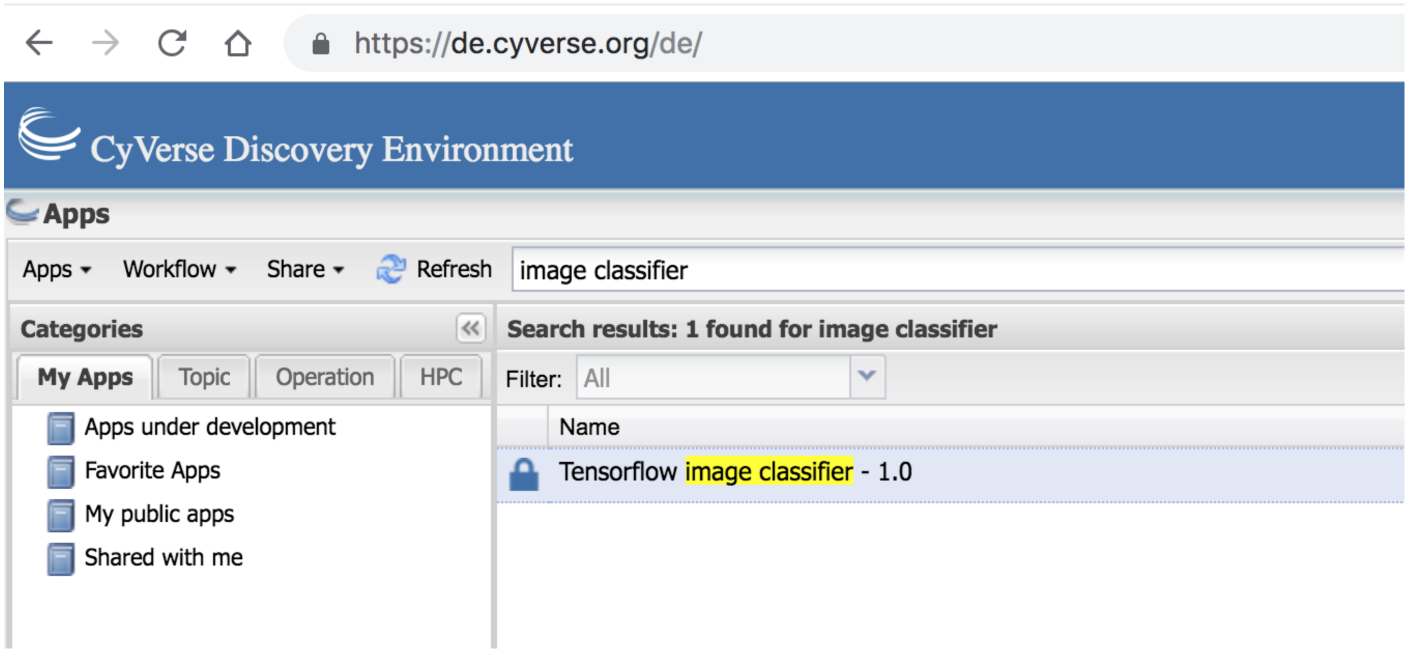
5. Test the app using appropriate test data in DE

After creating the new app according to your design, test your app in the Your Apps under development folder in the

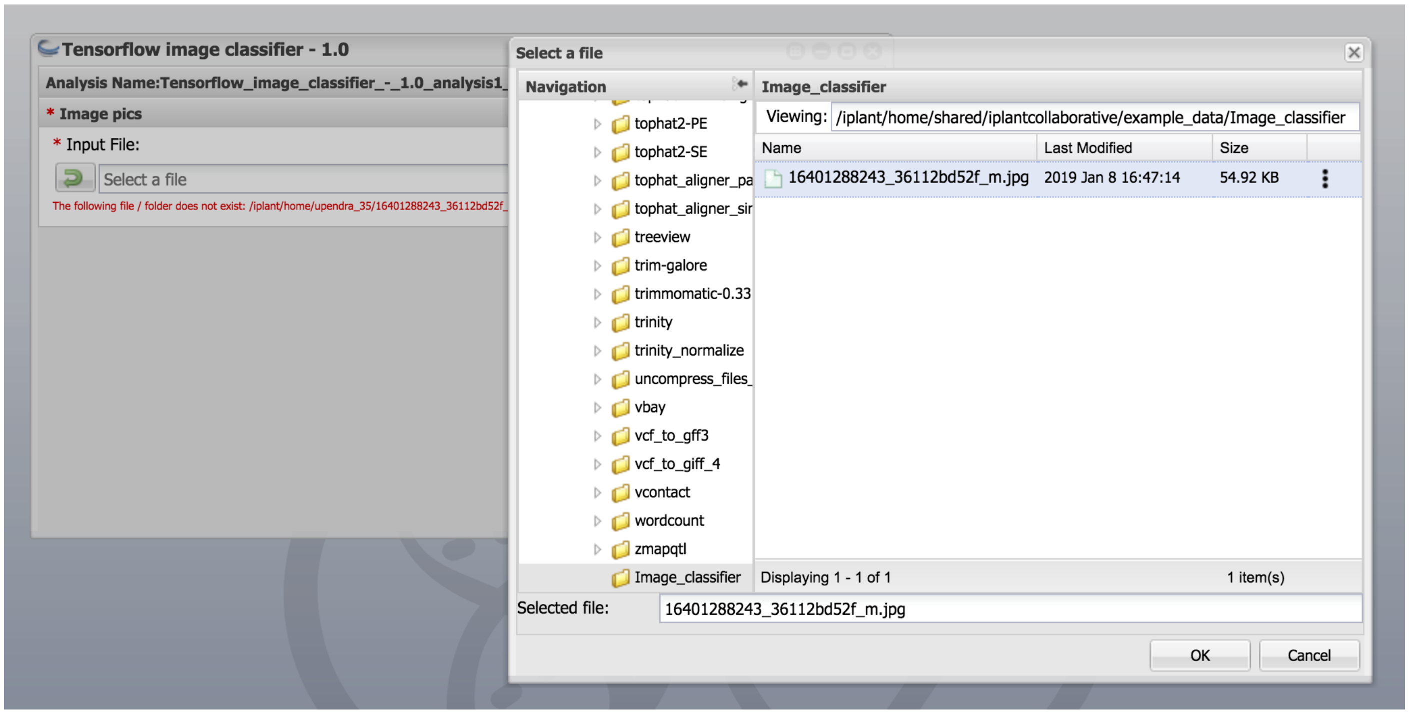
DE using appropriate test data to make sure it works properly.

For testing, we'll use the the same image that we used earlier.

1. First open the `Tensorflow image classifier - 1.0` app in the app window



2. Next browse the test file in the app and click launch analysis



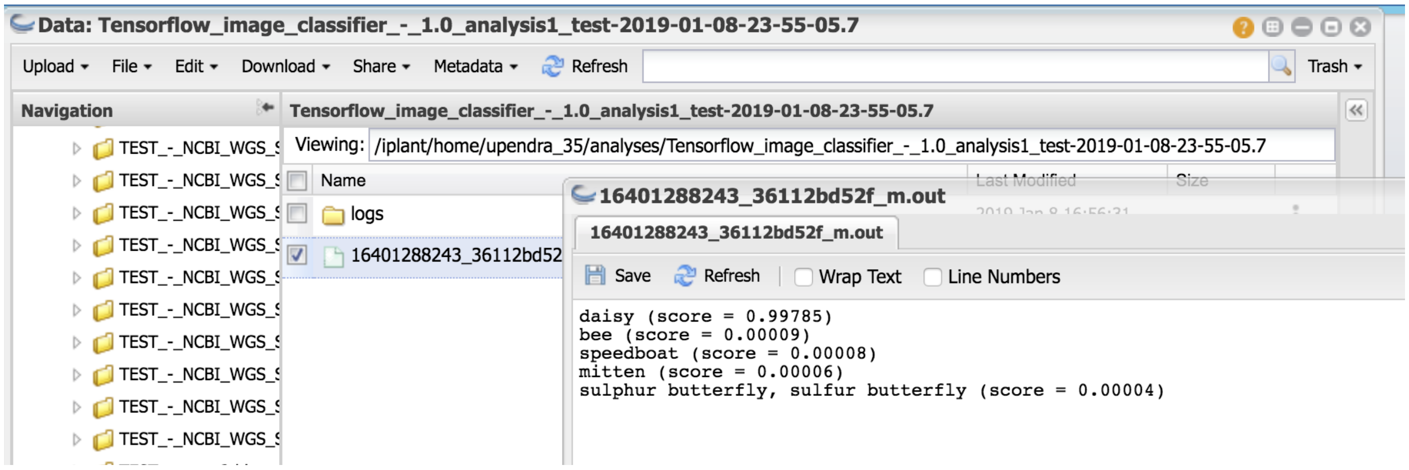
The following file / folder does not exist: /iplant/home/upendra_35/16401288243_36112bd52f_

Viewing: /iplant/home/shared/iplantcollaborative/example_data/Image_classifier

Name	Last Modified	Size
16401288243_36112bd52f_m.jpg	2019 Jan 8 16:47:14	54.92 KB

Selected file: 16401288243_36112bd52f_m.jpg

3. After the analysis is completed, open the folder and check to see if the image classifier correctly predicts


Data: Tensorflow_image_classifier_-_1.0_analysis1_test-2019-01-08-23-55-05.7
Upload
File
Edit
Download
Share
Metadata
Refresh
Trash
Navigation
TEST_-_NCBI_WGS_
Tensorflow_image_classifier_-_1.0_analysis1_test-2019-01-08-23-55-05.7
Name
logs
16401288243_36112bd52
16401288243_36112bd52f_m.out
Save
Refresh
Wrap Text
Line Numbers

Viewing: /iplant/home/upendra_35/analyses/Tensorflow_image_classifier_-_1.0_analysis1_test-2019-01-08-23-55-05.7

```
daisy (score = 0.99785)
bee (score = 0.00009)
speedboat (score = 0.00008)
mitten (score = 0.00006)
sulphur butterfly, sulfur butterfly (score = 0.00004)
```

Congrats!!! It works. The image classifier correctly predicts that the image is a daisy..

- If your app works the way you expect it to you can share your app or make the app public
- If your app doesn't work, then you may need to make changes to the app UI or you need to make changes to your Docker image. If you make changes to the Docker image, then you don't need to create a new app UI again as the Docker image updates will be propagated automatically.

CHAPTER 20

Deploying interactive apps in CyVerse Discovery Environment

The current apps in the DE are non-interactive, meaning the user selects parameters and data for a particular analysis, and submits the job for execution on platforms (Condor, HPC via Agave). When the process completes, the user is notified and they can view their analysis results in a folder. Any desired changes in results requires the user to change analysis parameters and run the job again to full completion. But exploratory data analysis (EDA) requires user to click and interact with running applications (i.e Data Scientists need a Workbench). Availability of computational notebooks (Jupyter, Zepplin) and Rstudio's Shiny allow users to readily share analysis in a reproducible manner and technologies like Javascript, WebGL, and others are making the web browser an extremely capable workbench

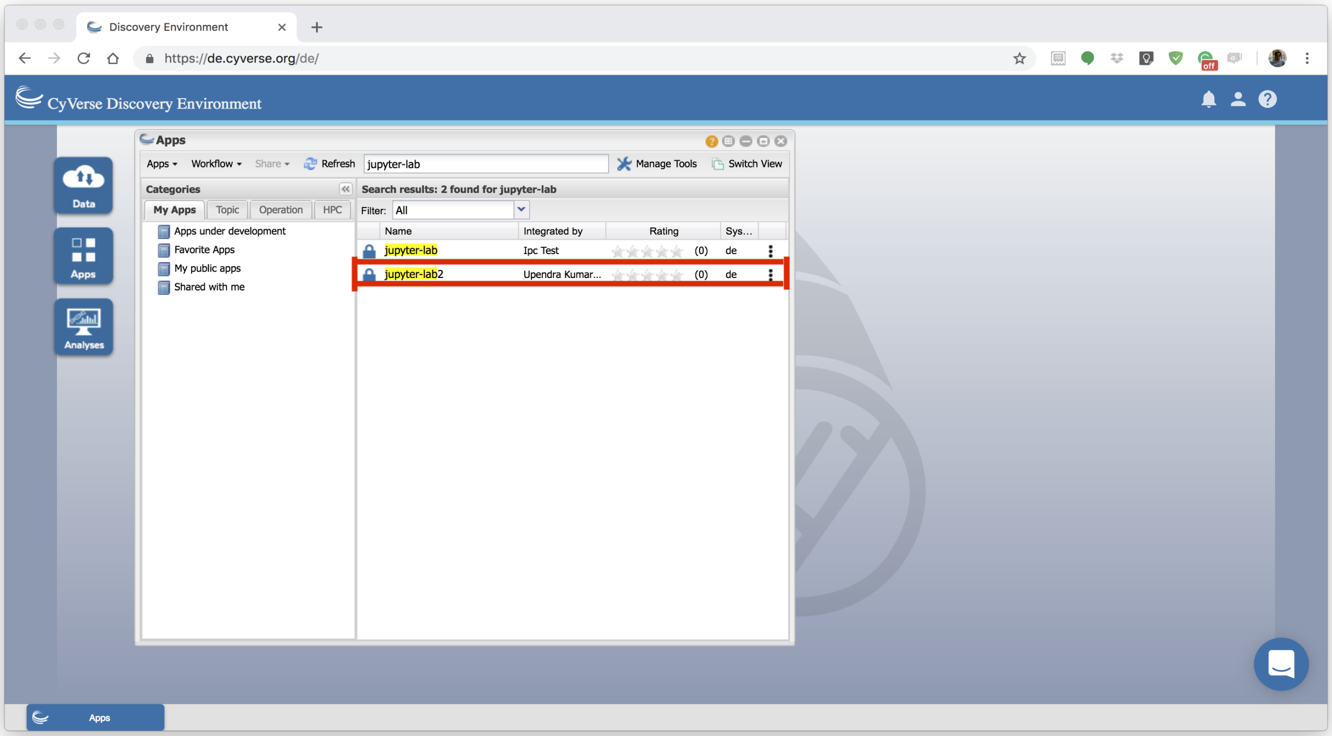
VICE (Visual Interactive Computing Environment) lets users interact with their data and do analyses in their favorite programming language in one place in an iterative way. Researchers can now explore their datasets interactively by easily changing parameters of selected analysis applications without having to download data from storage to an active workspace.

Here are the basic steps for deploying Docker images as interactive apps (VICE) in DE. For this tutorial I am going to show an example of Keras wine classifier

First log-in CyVerse DE

20.1 1. Search JupyterLab App

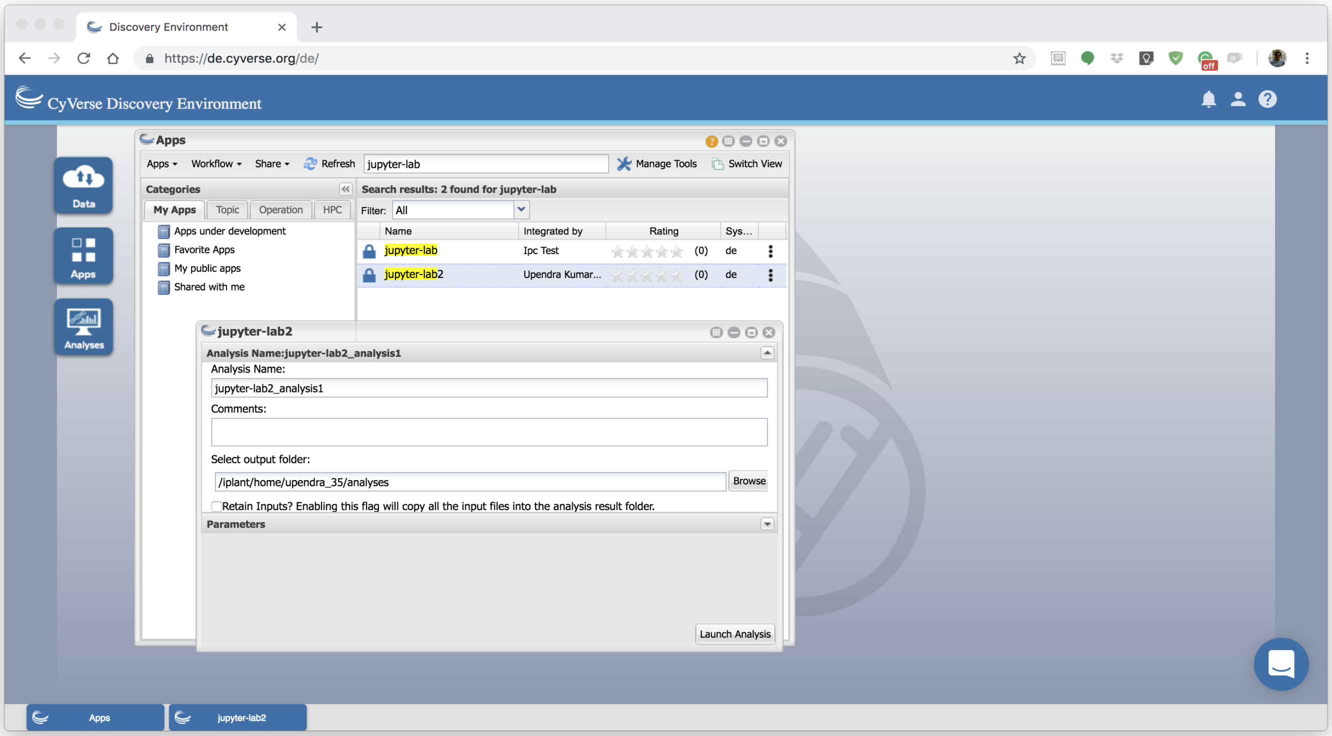
After you login to DE, open the Apps window and search the JupyterLab with key word *JupyterLab*.



Name	Integrated by	Rating	Sys...
jupyter-lab	Ipc Test	(0)	de
jupyter-lab2	Upendra Kumar...	(0)	de

20.2 2. Launch analysis

Launch the JupyterLab app by clicking **launch analysis**. Before you launch, you can either drag and drop or browse the files that you want to use with Jupyter-lab. There is currently no restriction of how many files and size of the files that can be launched along with JupyterLab app.



jupyter-lab

Search results: 2 found for jupyter-lab

Filter: All

Name	Integrated by	Rating		Sys...
jupyter-lab	Ipc Test	☆☆☆☆☆	(0)	de
jupyter-lab2	Upendra Kumar...	☆☆☆☆☆	(0)	de

Analysis Name:jupyter-lab2_analysis1

Analysis Name:

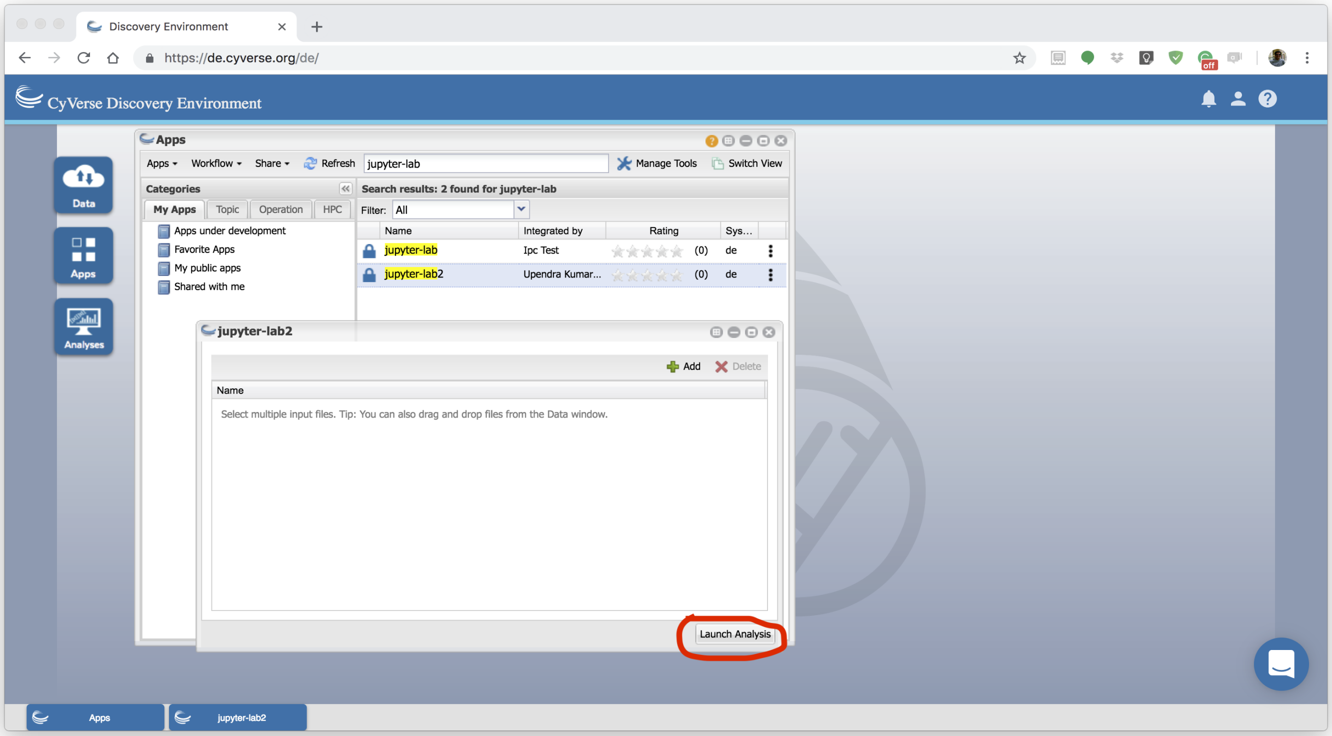
jupyter-lab2_analysis1

Comments:

Select output folder:

/iplant/home/upendra_35/analyses

Retain Inputs? Enabling this flag will copy all the input files into the analysis result folder.



jupyter-lab

Search results: 2 found for jupyter-lab

Filter: All

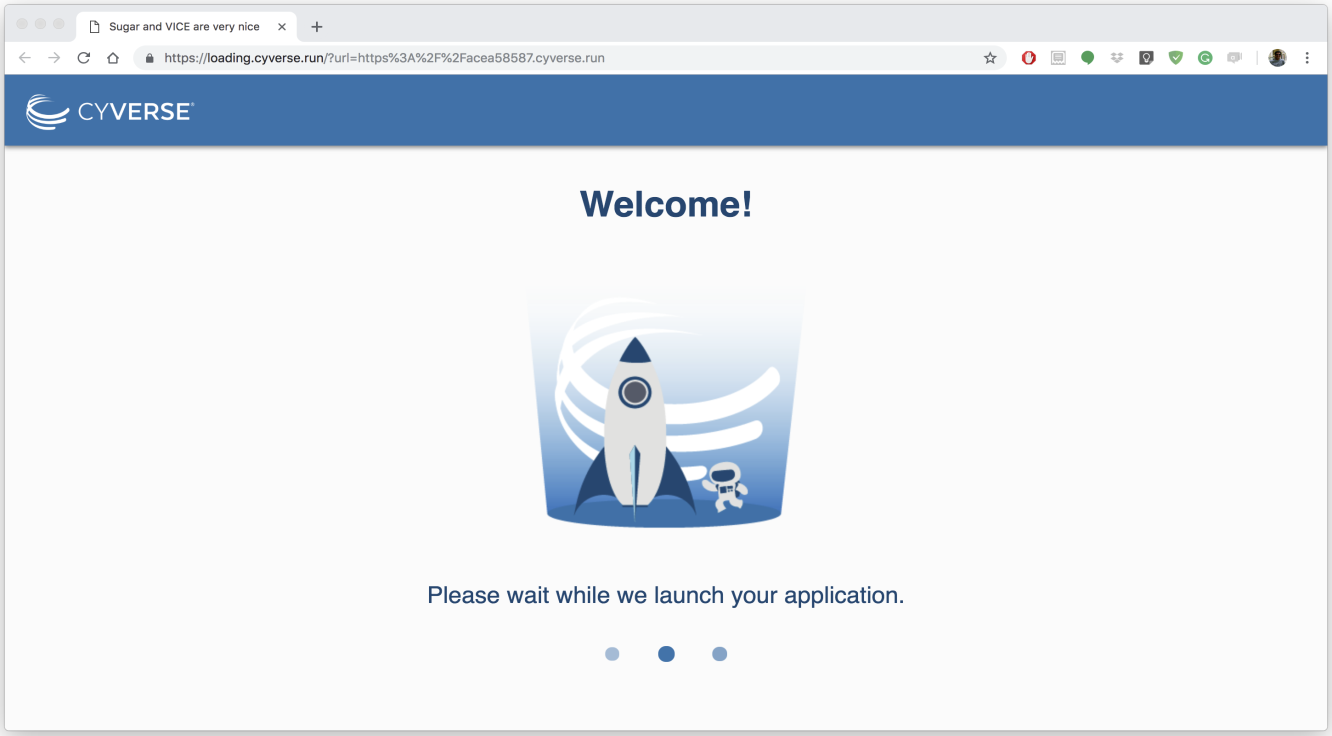
Name	Integrated by	Rating		Sys...
jupyter-lab	Ipc Test	☆☆☆☆☆	(0)	de
jupyter-lab2	Upendra Kumar...	☆☆☆☆☆	(0)	de

Select multiple input files. Tip: You can also drag and drop files from the Data window.

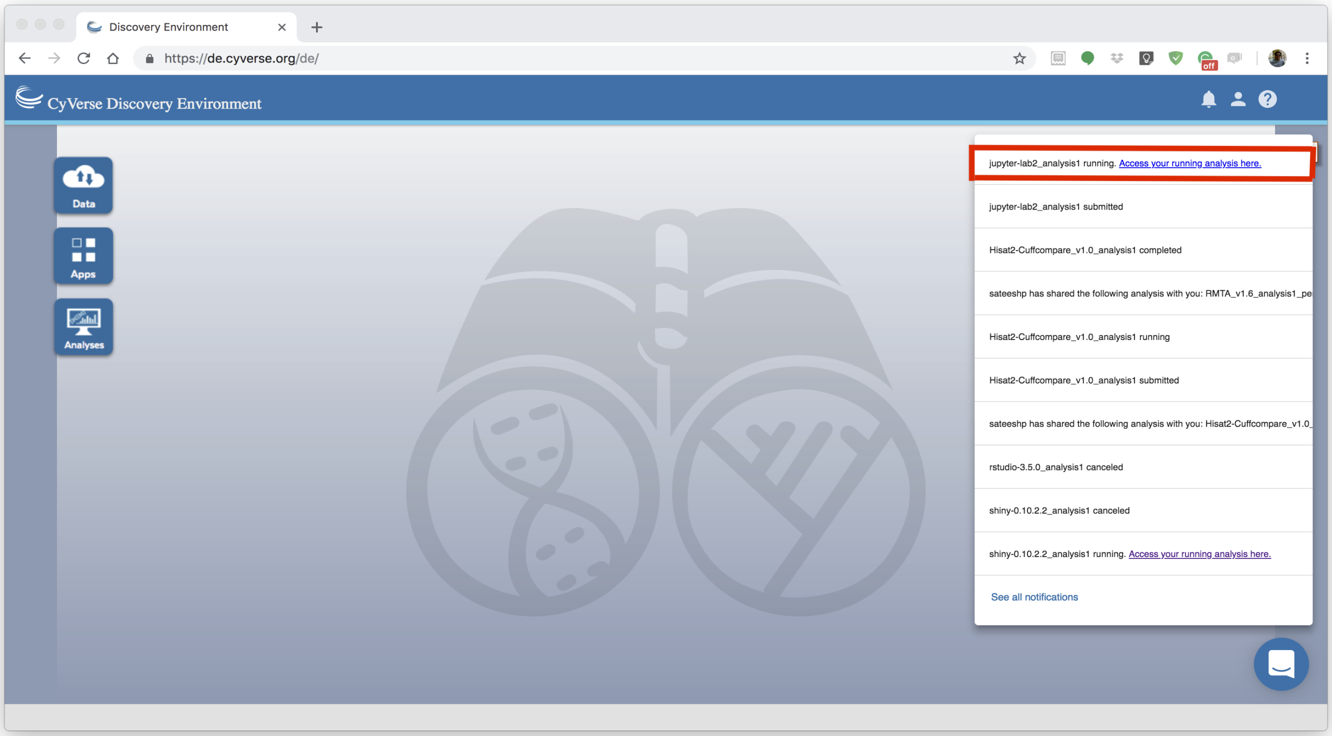
Note: The first two steps of launching apps are same as with other DE apps.

20.3 3. Navigate to JupyterLab url

Unlike regular DE apps once the analysis starts running you will get url. Clicking on the "Access your running Analysis here" url will redirect you to a page with a welcome message


Sugar and VICE are very nice
https://loading.cyverse.run/?url=https%3A%2F%2Facea58587.cyverse.run
CYVERSE
Welcome!
Please wait while we launch your application.

After it finished loading your app, the JupyterLab Interface automatically appears in your browser.



jupyter-lab2_analysis1 running. Access your running analysis here.

jupyter-lab2_analysis1 submitted

Hisat2-Cuffcompare_v1.0_analysis1 completed

sateeshp has shared the following analysis with you: RMTA_v1.6_analysis1_pe

Hisat2-Cuffcompare_v1.0_analysis1 running

Hisat2-Cuffcompare_v1.0_analysis1 submitted

sateeshp has shared the following analysis with you: Hisat2-Cuffcompare_v1.0_

rstudio-3.5.0_analysis1 canceled

shiny-0.10.2.2_analysis1 canceled

shiny-0.10.2.2_analysis1 running. Access your running analysis here.

See all notifications

The JupyterLab Interface: JupyterLab provides flexible building blocks for interactive, exploratory computing. While JupyterLab has many features found in traditional integrated development environments (IDEs), it remains focused on interactive, exploratory computing. The JupyterLab interface consists of a main work area containing tabs of documents and activities, a collapsible left sidebar, and a menu bar. The left sidebar contains a file browser, the list of running kernels and terminals, the command palette, the notebook cell tools inspector, and the tabs list.

More information about the JupyterLab can be found here

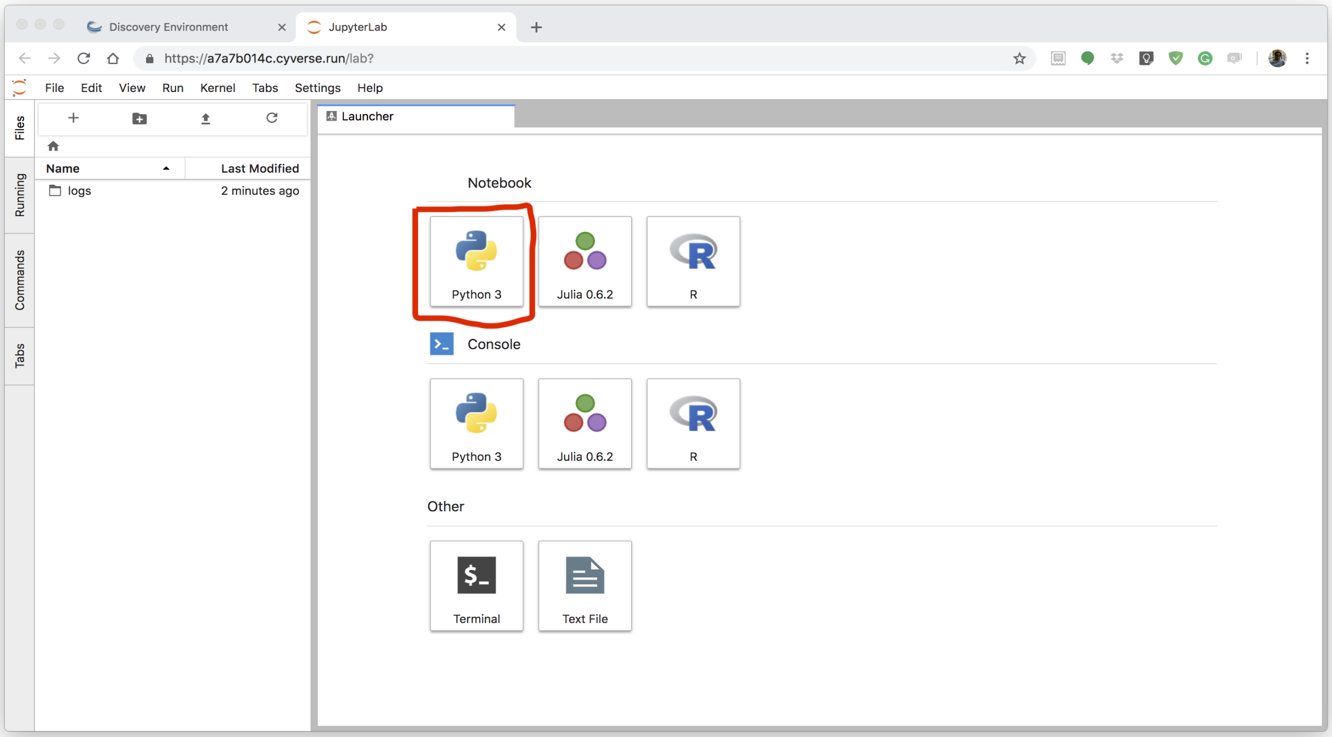
20.4 4. Create Jupyter notebook

Jupyter notebooks are documents that combine live runnable code with narrative text (Markdown), equations (LaTeX), images, interactive visualizations and other rich output. Jupyter notebooks (.ipynb files) are fully supported in JupyterLab

If you want to create a notebook, you can do so by clicking the + button in the file browser and then selecting a kernel in the new Launcher tab. Currently there are 3 different notebooks available - Python3, Julia and R. Click on *Python 3* under Notebook section in the JupyterLab Interface, which will open a new Jupyter Notebook. A new file is created with a default name. Rename a file by right-clicking on its name in the file browser and selecting "Rename" from the context menu.

To know more about notebooks in JupyterLab click here

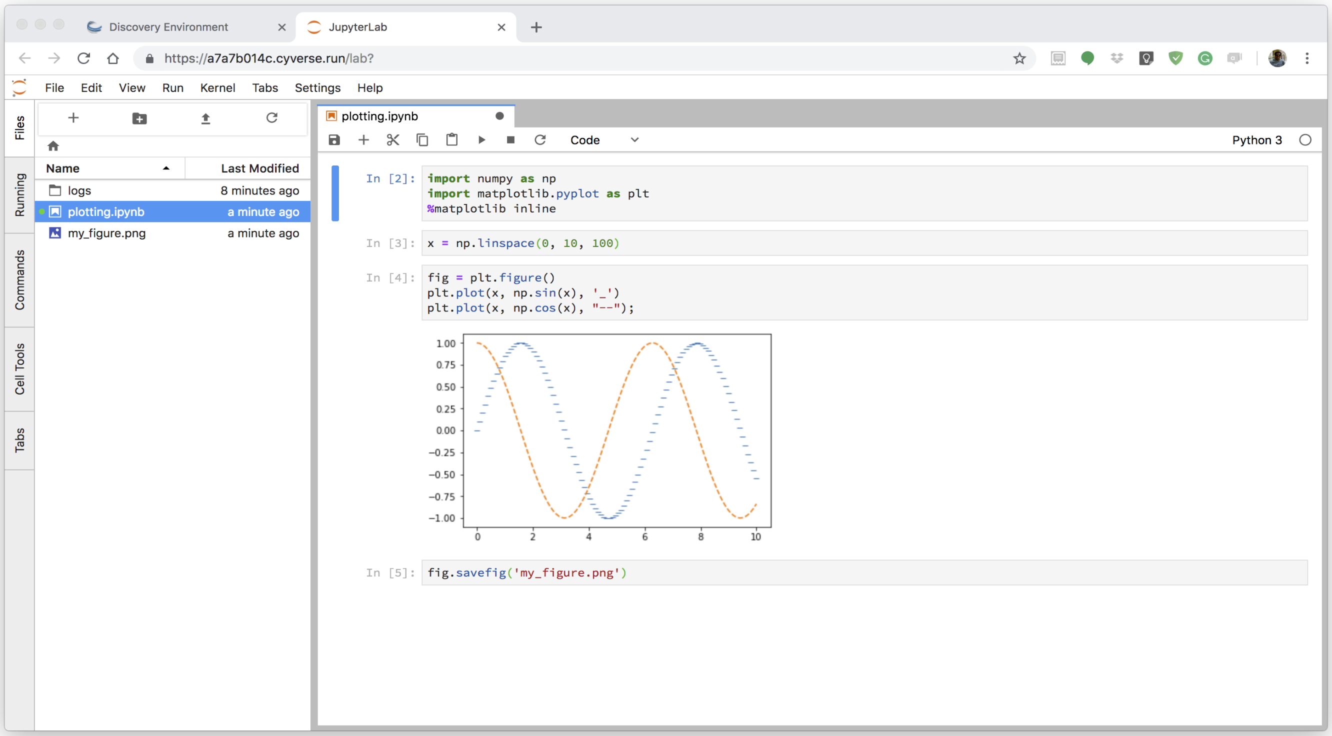
Tip: To open the classic Notebook from JupyterLab, select "Launch Classic Notebook" from the JupyterLab Help menu.



Note: There are plenty other cool stuff that you can do in JupyterLab such as using consoles, using terminal and using text editor

20.5 5. Write your code

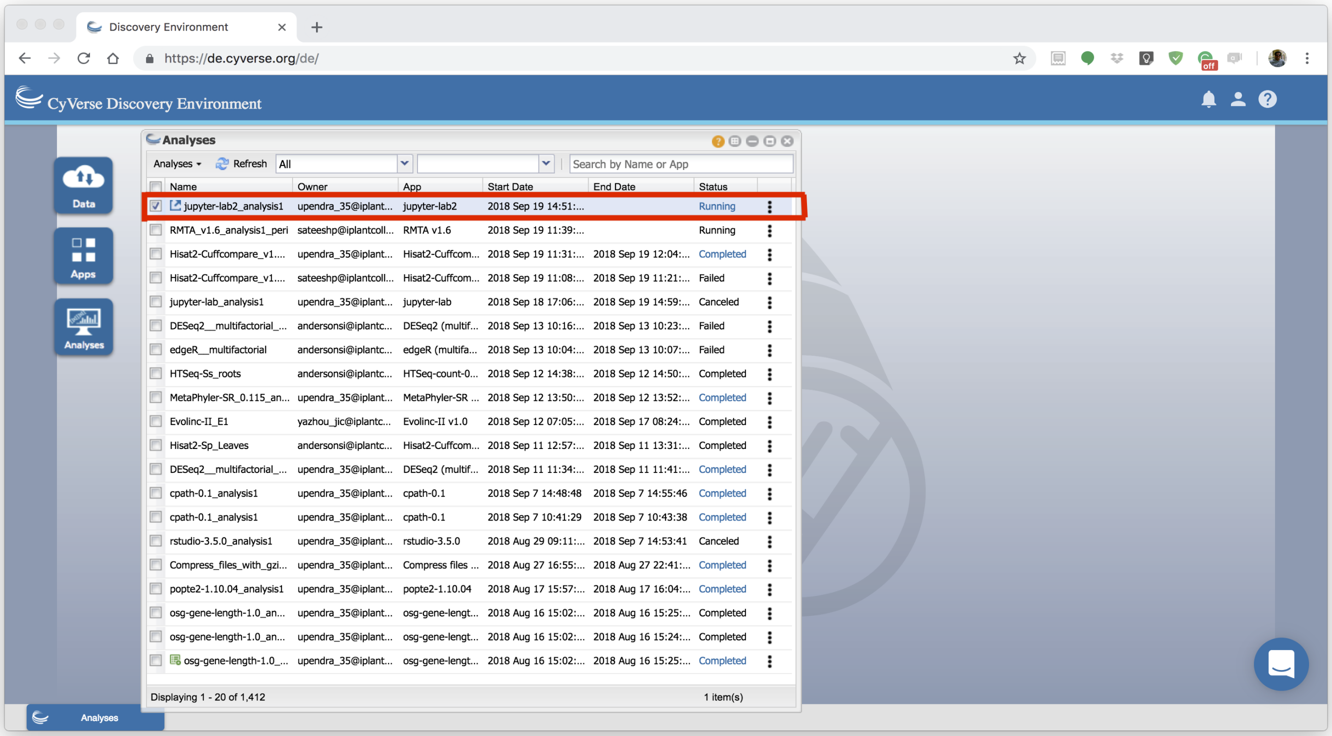
Once you open a new notebook, you can start writing your code, put markdown text, generate plots, save plots etc.


Discovery Environment
JupyterLab
https://a7a7b014c.cyverse.run/lab?
File Edit View Run Kernel Tabs Settings Help
Files
Running
Commands
Cell Tools
Tabs
Name
Last Modified
logs 8 minutes ago
plotting.ipynb a minute ago
my_figure.png a minute ago
plotting.ipynb
Code
Python 3

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

In [3]: x = np.linspace(0, 10, 100)

In [4]: fig = plt.figure()
        plt.plot(x, np.sin(x), '_')
        plt.plot(x, np.cos(x), "--");

In [5]: fig.savefig('my_figure.png')
```

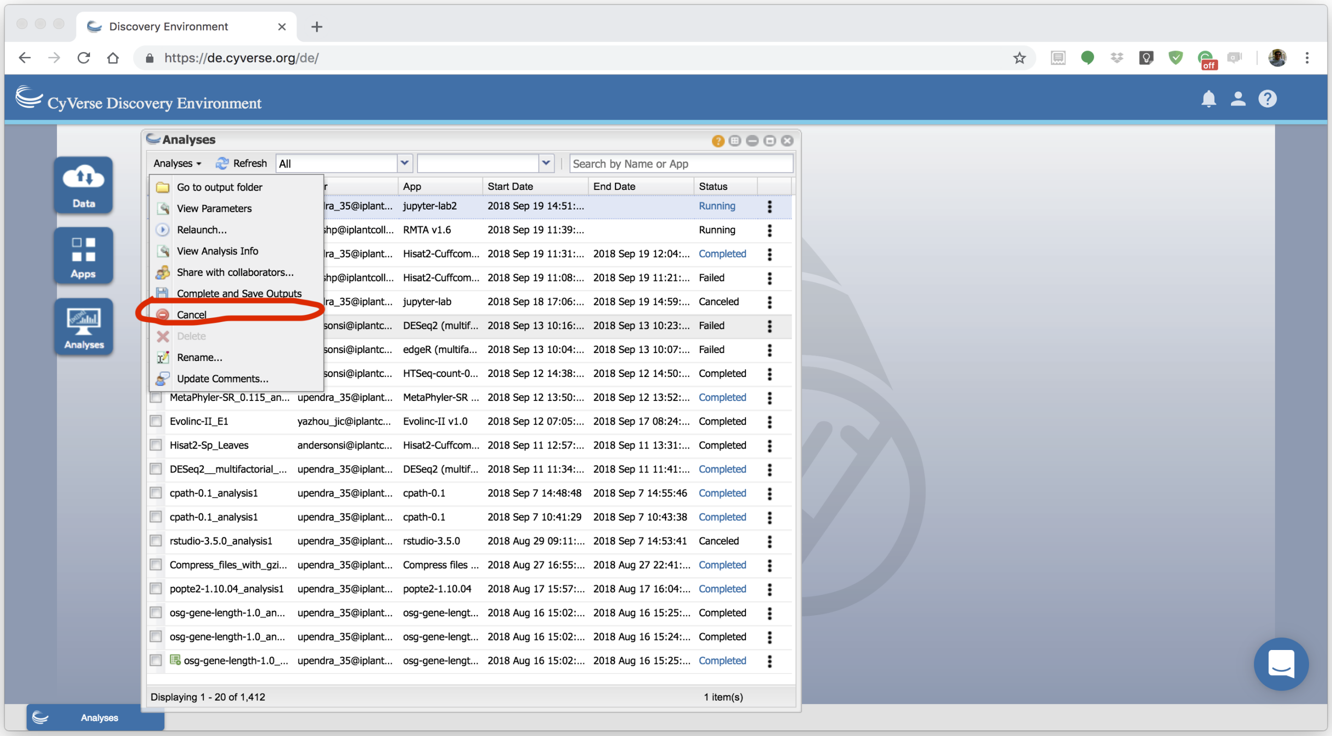
20.6 6. Complet and Save Outputs

After finishing your analysis, you can save outputs to data store by clicking the Analysis window, then select the sshiny analysis that you are running and then selecting *Complete and Save Outputs* under "Analyses" button.



Name	Owner	App	Start Date	End Date	Status
jupyter-lab2_analysis1	upendra_35@iplant...	jupyter-lab2	2018 Sep 19 14:51:...		Running
RMTA_v1.6_analysis1_peri	sateeshp@iplantcoll...	RMTA v1.6	2018 Sep 19 11:39:...		Running
Hisat2-Cuffcompare_v1....	upendra_35@iplant...	Hisat2-Cuffcom...	2018 Sep 19 11:31:...	2018 Sep 19 12:04:...	Completed
Hisat2-Cuffcompare_v1....	sateeshp@iplantcoll...	Hisat2-Cuffcom...	2018 Sep 19 11:08:...	2018 Sep 19 11:21:...	Failed
jupyter-lab_analysis1	upendra_35@iplant...	jupyter-lab	2018 Sep 18 17:06:...	2018 Sep 19 14:59:...	Canceled
DESeq2__multifactorial_...	andersonsi@iplantc...	DESeq2 (multif...	2018 Sep 13 10:16:...	2018 Sep 13 10:23:...	Failed
edgeR__multifactorial	andersonsi@iplantc...	edgeR (multifa...	2018 Sep 13 10:04:...	2018 Sep 13 10:07:...	Failed
HTSeq-Ss_roots	andersonsi@iplantc...	HTSeq-count-0...	2018 Sep 12 14:38:...	2018 Sep 12 14:50:...	Completed
MetaPhyler-SR_0.115_an...	upendra_35@iplant...	MetaPhyler-SR ...	2018 Sep 12 13:50:...	2018 Sep 12 13:52:...	Completed
Evolinc-II_E1	yazhou_jic@iplantc...	Evolinc-II v1.0	2018 Sep 12 07:05:...	2018 Sep 17 08:24:...	Completed
Hisat2-Sp_Leaves	andersonsi@iplantc...	Hisat2-Cuffcom...	2018 Sep 11 12:57:...	2018 Sep 11 13:31:...	Completed
DESeq2__multifactorial_...	upendra_35@iplant...	DESeq2 (multif...	2018 Sep 11 11:34:...	2018 Sep 11 11:41:...	Completed
cpath-0.1_analysis1	upendra_35@iplant...	cpath-0.1	2018 Sep 7 14:48:48	2018 Sep 7 14:55:46	Completed
cpath-0.1_analysis1	upendra_35@iplant...	cpath-0.1	2018 Sep 7 10:41:29	2018 Sep 7 10:43:38	Completed
rstudio-3.5.0_analysis1	upendra_35@iplant...	rstudio-3.5.0	2018 Aug 29 09:11:...	2018 Sep 7 14:53:41	Canceled
Compress_files_with_gzi...	upendra_35@iplant...	Compress files ...	2018 Aug 27 16:55:...	2018 Aug 27 22:41:...	Completed
popte2-1.10.04_analysis1	upendra_35@iplant...	popte2-1.10.04	2018 Aug 17 15:57:...	2018 Aug 17 16:04:...	Completed
osg-gene-length-1.0_an...	upendra_35@iplant...	osg-gene-lengt...	2018 Aug 16 15:02:...	2018 Aug 16 15:25:...	Completed
osg-gene-length-1.0_an...	upendra_35@iplant...	osg-gene-lengt...	2018 Aug 16 15:02:...	2018 Aug 16 15:24:...	Completed
osg-gene-length-1.0_...	upendra_35@iplant...	osg-gene-lengt...	2018 Aug 16 15:02:...	2018 Aug 16 15:25:...	Completed

Displaying 1 - 20 of 1,412 1 item(s)



Name	Owner	App	Start Date	End Date	Status
	ra_35@iplant...	jupyter-lab2	2018 Sep 19 14:51:...		Running
	hp@iplantcoll...	RMTA v1.6	2018 Sep 19 11:39:...		Running
	ra_35@iplant...	Hisat2-Cuffcom...	2018 Sep 19 11:31:...	2018 Sep 19 12:04:...	Completed
	hp@iplantcoll...	Hisat2-Cuffcom...	2018 Sep 19 11:08:...	2018 Sep 19 11:21:...	Failed
	ra_35@iplant...	jupyter-lab	2018 Sep 18 17:06:...	2018 Sep 19 14:59:...	Canceled
	sonsi@iplantc...	DESeq2 (multif...	2018 Sep 13 10:16:...	2018 Sep 13 10:23:...	Failed
	sonsi@iplantc...	edgeR (multifa...	2018 Sep 13 10:04:...	2018 Sep 13 10:07:...	Failed
	sonsi@iplantc...	HTSeq-count-0...	2018 Sep 12 14:38:...	2018 Sep 12 14:50:...	Completed
MetaPhyler-SR_0.115_an...	upendra_35@iplant...	MetaPhyler-SR ...	2018 Sep 12 13:50:...	2018 Sep 12 13:52:...	Completed
Evolinc-II_E1	yazhou_jic@iplantc...	Evolinc-II v1.0	2018 Sep 12 07:05:...	2018 Sep 17 08:24:...	Completed
Hisat2-Sp_Leaves	andersonsi@iplantc...	Hisat2-Cuffcom...	2018 Sep 11 12:57:...	2018 Sep 11 13:31:...	Completed
DESeq2__multifactorial_...	upendra_35@iplant...	DESeq2 (multif...	2018 Sep 11 11:34:...	2018 Sep 11 11:41:...	Completed
cpath-0.1_analysis1	upendra_35@iplant...	cpath-0.1	2018 Sep 7 14:48:48	2018 Sep 7 14:55:46	Completed
cpath-0.1_analysis1	upendra_35@iplant...	cpath-0.1	2018 Sep 7 10:41:29	2018 Sep 7 10:43:38	Completed
rstudio-3.5.0_analysis1	upendra_35@iplant...	rstudio-3.5.0	2018 Aug 29 09:11:...	2018 Sep 7 14:53:41	Canceled
Compress_files_with_gzi...	upendra_35@iplant...	Compress files ...	2018 Aug 27 16:55:...	2018 Aug 27 22:41:...	Completed
popte2-1.10.04_analysis1	upendra_35@iplant...	popte2-1.10.04	2018 Aug 17 15:57:...	2018 Aug 17 16:04:...	Completed
osg-gene-length-1.0_an...	upendra_35@iplant...	osg-gene-lengt...	2018 Aug 16 15:02:...	2018 Aug 16 15:25:...	Completed
osg-gene-length-1.0_an...	upendra_35@iplant...	osg-gene-lengt...	2018 Aug 16 15:02:...	2018 Aug 16 15:24:...	Completed
osg-gene-length-1.0_...	upendra_35@iplant...	osg-gene-lengt...	2018 Aug 16 15:02:...	2018 Aug 16 15:25:...	Completed

Displaying 1 - 20 of 1,412 1 item(s)

After you had done this, you can find the outputs that you generated (if any) in the analysis of the JupyterLab.

Warning: Currently, VICE can run for 48 hrs beyond which the apps will be terminated. So make sure you run your analysis before 48 hrs.

CHAPTER 21

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

CHAPTER 22

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*

22.1 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 23

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
- Banner UMC Cafeteria

CHAPTER 24

For instructors!

Coordinating Web site work

Please create a pull request (PR) 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 Container Camp workshop tutorials are stored on GitHub.

We will use GitHub Flow for updates: from the command line,

- fork the container camp repository;
- edit, change, add, etc;
- submit a PR;
- when ready to review & merge, say 'ready for review & merge @cc2019'.

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/container_camp_workshop_2019.
↪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.

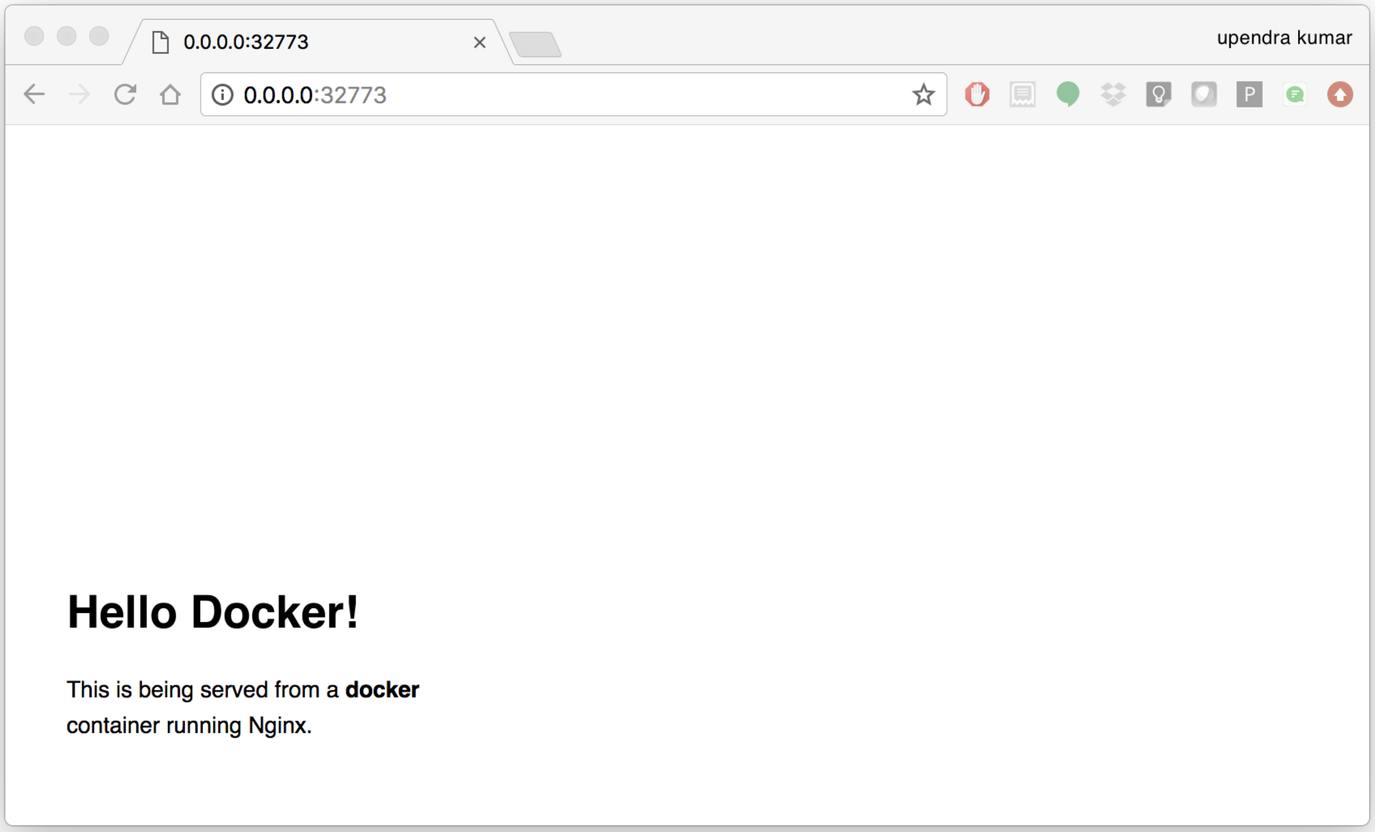
(Note that you can go visit the github repo and it will helpfully render *.rst* files for you if you click on them! They just won't have the full site template.)

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-2019.readthedocs-hosted.com/_static/filename

Images

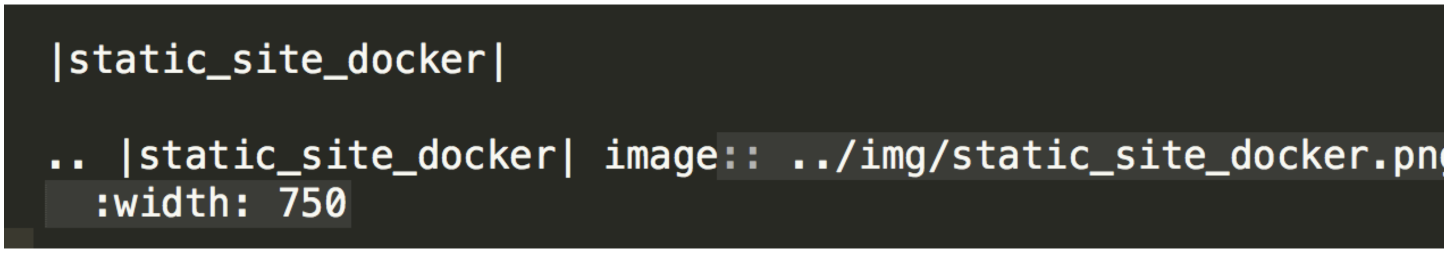
Image formatting in Restructured text is pretty straightforward. Here is an example



Hello Docker!

This is being served from a **docker** container running Nginx.

The code that generates this image is this



```
|static_site_docker|

.. |static_site_docker| image:: ../img/static_site_docker.pn
   :width: 750
```

CHAPTER 25

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!

Fix or improve this documentation

- On Github: Repo link
- Send feedback: support@cyverse.org