CyVerse Documentation

Release 0.1.0

CyVerse

Sep 18, 2020

Cybercarpentry workshop On Containers

CHAPTER 1

Workshop Code of Conduct

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

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

FAIR principles

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

Learning objectives

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

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

Workshop level

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

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

Need help?

Couldn't find what you were looking for?

- You can reach the lead instructor, Upendra Devisetty, at upendra at cyverse dot org.
- You can also talk to any of the instructors or TAs if you need immediate help.
- 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 TRIPODS Mini Course on Containers at the Biosphere2, which will run on May 21st, 2018.

Prerequisite	Notes	Links
Wi-Fi-enabled laptop with up to date web-browser	You should be able to use any laptop using Windows/MacOS/Linux. We **strongly recommend** Firefox or Chrome browser; **We do not recommend Microsoft Edge Browser**. It is recommended that you have administrative/install permissions on your laptop.	• Download FireFox • Download Chrome
Github Account	Please ensure that you have a Github account	Register for your Github account at https://github.com/.
Docker	Install Docker on your computer (Optional since we will be using Jetstream for this workshop)	• Docker for Mac • Docker for Windows • Docker for Linux
Singularity (Optional since we will be using Jetstream for this workshop)	Install Singularity on your computer	• Singularity for Mac • Singularity for Windows • Singularity for Linux
Dockerhub Account	Please ensure that you have a Dockerhub account	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/.
Jetstream Cloud	The Jetstream Cloud service is part of XSEDE and so it required allocation.	The Jetstream Cloud service should be listed in this webapge once you log in to CyVerse.
HPC Account (Optional)	Please ensure that you have a HPC account.	Link here

CHAPTER 3

Agenda

Below are the schedule and classroom materials for Cybercarpentry container workshop, which will run on July 18th

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

Day	Time	Topic/Activity	Notes/Links
07/18/18 (Wednesday)	9:00am-12:00pm	Introduction to Docker (Upendra Devisetty)	• Docker intro slides • Docker demo
07/18/18 (Wednesday)	12:00pm-1:00pm	Lunch	Location
07/18/18 (Wednesday)	1:00pm-05:30pm	General overview of Singularity (Upendra Devisetty)	• Singularity Overview Slides • Singularity demo

CHAPTER 4

About CyVerse

CyVerse Vision: Transforming science through data-driven discovery.

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

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

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

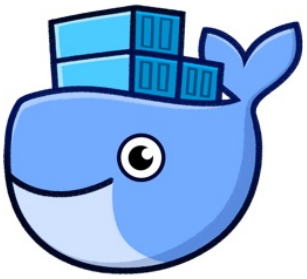
Funding and Citations

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

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

CHAPTER 5

Introduction to Docker



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

5.1 1. Prerequisites

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

5.2 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 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.

5.3 2.1 XSEDE Jetstream / CyVerse Atmosphere Clouds

CyVerse staff have deployed an Ansible playbooks called `ez` installation which includes Docker that only requires you to type a short line of code.

Start a featured instance on Jetstream [needs to insert link to Jetstream's launching instance].

Type in the following:

```
$ ezd

* Updating ez docker and installing docker (this may take a few minutes, coffee break!
↪)
Cloning into '/opt/cyverse-ez-docker'...
remote: Counting objects: 38, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 38 (delta 1), reused 0 (delta 0), pack-reused 31
Unpacking objects: 100% (38/38), done.
* docker was updated successfully

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

To test docker, type: docker run hello-world
```

Let's test docker by running *docker run hello-world*

```
$ docker run hello-world
```

You should see an error like this

```
Got permission denied while trying to connect to the Docker daemon socket at unix:///
↪var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.35/containers/json?
↪all=1: dial unix /var/run/docker.sock: connect: permission denied
```

This is because Docker needs root access to run Docker commands and so either can append *sudo* infront of each command (and you have sudo permissions on Jetstream instance) or add yourself to the Docker group using following instructions..

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

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

Rerun the `docker run hello-world` command to make sure the everything is working fine

```
$ 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.
```

(continues on next page)

(continued from previous page)

```
 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.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/
```

5.4 3. Running Docker containers

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

But wait, what exactly is a Container and Image?

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

Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run `docker inspect hello-world` and `docker history hello-world`.

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

```
$ docker run alpine
```

What happened there? Similar to `docker run hello-world` command in the demo above, `docker run alpine` command fetches the `alpine:latest` image from the Docker registry first, saves it in our system. However unlike *hello-world* run, here we don't see any output on the screen. This is because *alpine* and *ubuntu* (which we see later) are base images and doesn't have any executable that produces any kind of output. They however launches a container and since the container doesn't have executable, they quickly exits. You can check the status of container by running the command `docker ps -l`

```
$ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ␣
↪STATUS                     PORTS               NAMES
b4219971d6e5        alpine              "/bin/sh"           7 minutes ago       ␣
↪Exited (0) 7 minutes ago                       vigilant_goodall
```

`docker ps -l` command shows the status of last container run and as you can see the container has been exited from the `alpine` run.

Let's add some commands to the container run this time.

```
$ 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!

Let's say if you wanted to see files and folders inside the `alpine` container, then you can run the following

```
$ 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
........
```

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

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`, `whoami` and others.

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

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

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

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

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

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

```
$ docker run -it --name alpine_test alpine sh
/ # whoami
root
```

Running the `run` command with the `-it` flags attaches us to an interactive `tty` in the container and `--name` is the name of the container (otherwise docker will name the container with some random name). Now you can run as many commands in the container as you want. Take some time to run your favorite commands.

Tip: You don't really have to use *sh* as by default when you enter into interactive model, it puts you into *sh* shell

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

```
/ # exit
```

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

```
$ docker ps -l
CONTAINER ID        IMAGE                 COMMAND                  CREATED         ␣
↪    STATUS                          PORTS                    NAMES
de4bbc3eeaec        alpine                "/bin/sh"                3 minutes ago   ␣
↪    Exited (0) About a minute ago                            alpine_test
```

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

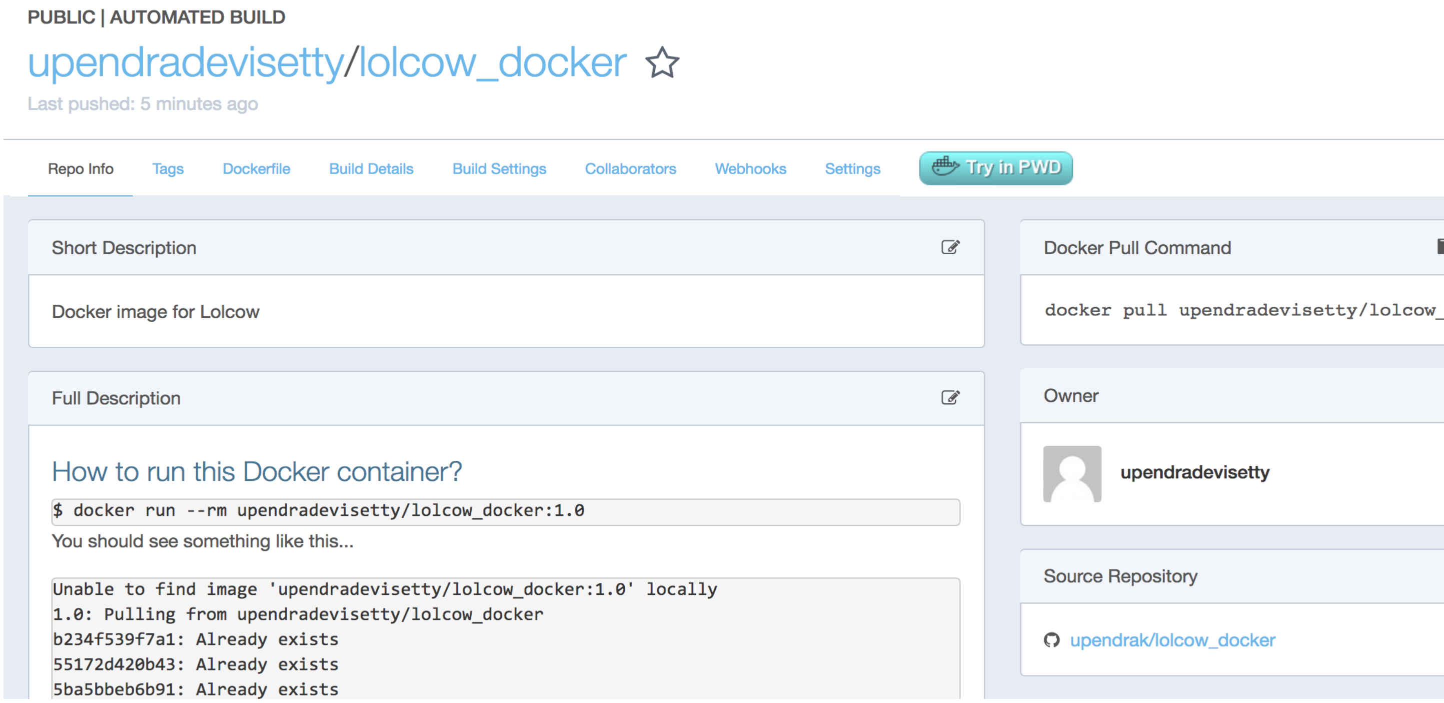
```
$ docker ps -l
CONTAINER ID        IMAGE                 COMMAND                  CREATED         ␣
↪    STATUS                         PORTS                    NAMES
0db38ea51a48        alpine                "sh"                     3 minutes ago   ␣
↪    Up 3 minutes                                            alpine_test
```

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 alpine_test
```

5.4.1 3.1 Running Docker container from a custom Docker image

Let's run a Docker container from a custom Docker image and we will learn how Docker images are built using Dockerfile. But first pull a *lolcow* Docker image from Dockerhub



PUBLIC | AUTOMATED BUILD

upendradevisetty/lolcow_docker ☆

Last pushed: 5 minutes ago

Short Description

Docker image for Lolcow

Full Description

How to run this Docker container?

```
$ docker run --rm upendradevisetty/lolcow_docker:1.0
```

You should see something like this...

```
Unable to find image 'upendradevisetty/lolcow_docker:1.0' locally
1.0: Pulling from upendradevisetty/lolcow_docker
b234f539f7a1: Already exists
55172d420b43: Already exists
5ba5bbeb6b91: Already exists
```

Docker Pull Command

```
docker pull upendradevisetty/lolcow_
```

Owner

upendradevisetty

Source Repository

upendrak/lolcow_docker

```
$ docker run --rm --name lolcow upendradevisetty/lolcow_docker:1.0
Unable to find image 'upendradevisetty/lolcow_docker:1.0' locally
1.0: Pulling from upendradevisetty/lolcow_docker
b234f539f7a1: Already exists
55172d420b43: Already exists
5ba5bbeb6b91: Already exists
43ae2841ad7a: Already exists
f6c9c6de4190: Already exists
8e55b0aed266: Already exists
c2c7ed47afbc: Already exists
Digest: sha256:37062c6724b6746ea75c9d11ca4a335d5590702368039ada7ff8801c8cc6620f
Status: Downloaded newer image for upendradevisetty/lolcow_docker:1.0
 ____________________________________
/ Q: What is orange and goes "click, \
\ click?" A: A ball point carrot.    /
 ------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Tip: If you have noticed, I am running *docker run* with an extra flag *–rm*. This flag is quite useful and it removes the container after the container exits. So you don't have to manually remove them

Let's take a look at the Dockerfile now for this image. Before that what exactly is a Dockerfile?

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

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

FROM

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

MAINTAINER

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

LABEL

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

RUN

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

ENV

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

CMD

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

5.5 4. Docker images

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

```
$ docker images
REPOSITORY     TAG       IMAGE ID        CREATED            SIZE
hello-world    latest    690ed74de00f    5 months ago       960 B
alpine         latest    3fd9065eaf02    3 months ago       4.15MB
.........
```

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

# Exercise

- Build a Docker image for this tool - Diamond blast

Bonus:

- Can you build from the source instead of binaries for the same tool?

5.6 5. Docker Registry

You've built a Docker image. Now what?

What exactly is a registry?

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

What is Docker registry?

A Docker registry is a place to store and distribute Docker images. It serves as a target for your docker push and docker pull commands. Before we get any further, let's cover some of the basic terminology related to Docker registries. 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.

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.

There are several things you can do with Docker registries:

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

Why Do I Need a Docker Registry?

Imagine a workflow where you push a commit that triggers a build on your CI provider which in turn pushes a new image into your registry. Your registry can then fire off a webhook and trigger a deployment. All without a human having to step and manually do anything. Registries make a fully automated workflow like this much easier.

As the previous example demonstrates, you will likely want to have a private registry for storing your proprietary images. A public registry such as the one on Docker Hub is hugely helpful for publicly available and open-source images. However, for your company's private images, a private registry is what you need.

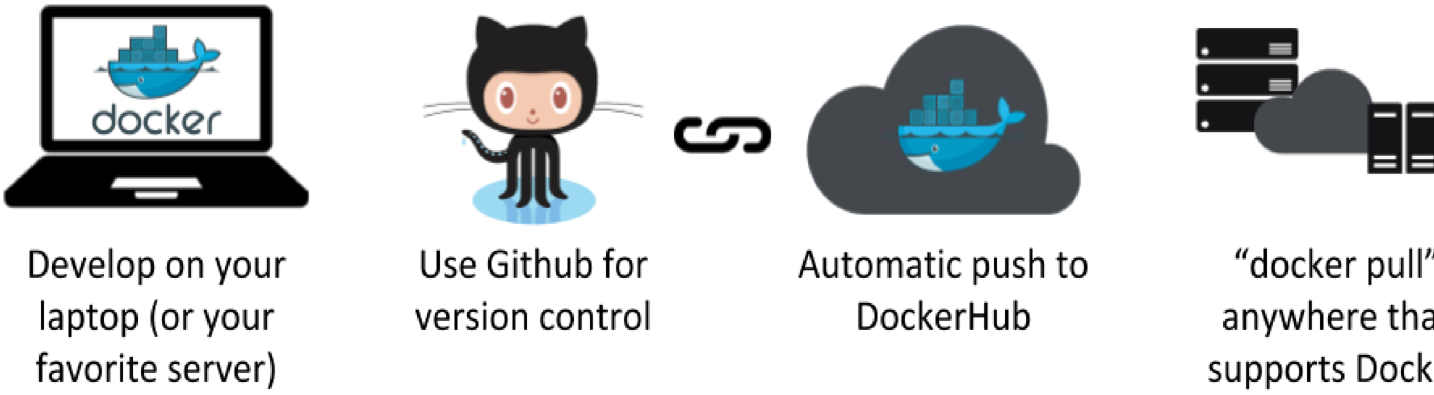
The question then is: Public or Private?

The decision to host your own private registry or to go with a public option is much like any other question of building it yourself versus outsourcing to a third party. Price, control, flexibility and maintenance all come into play. Instead of rehashing these arguments though, let's instead just run through what your options are.

Some example of public registries include Docker cloud, Docker hub, quay.io, AWS EC2 Container Registry (ECR), *Google Container Registry (GCR)* <https://cloud.google.com/container-registry/>'_ etc.,

5.6.1 5.1.2 Push the iamge to Dockerhub registry

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.


docker
Develop on your laptop (or your favorite server)
Use Github for version control
Automatic push to DockerHub
"docker pull" anywhere tha
supports Dock

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.

5.1.2.1 Prerequisites

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

Note:

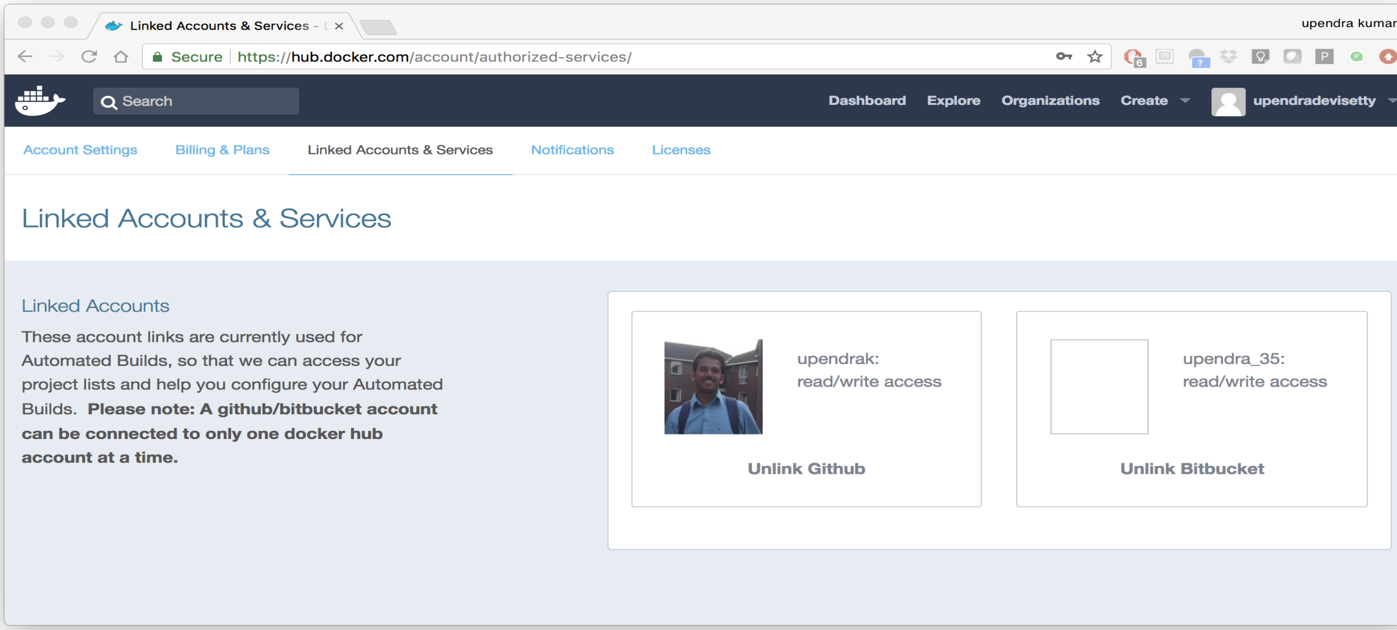
- If you have previously linked your Github or Bitbucket account, you must have chosen the Public and Private connection type. To view your current connection settings, log in to Docker Hub and choose Profile > Settings > Linked Accounts & Services.
- Building Windows containers is not supported.

5.1.2.2 Link your Docker Hub account to GitHub

1. Log into Docker Hub.
2. Navigate to Profile > Settings > Linked Accounts & Services.

3. Click the `Link GitHub`. The system prompts you to choose between **Public and Private** and **Limited Access**. The **Public** and **Private** connection type is required if you want to use the Automated Builds.
4. Press `Select` under **Public and Private** connection type. If you are not logged into GitHub, the system prompts you to enter GitHub credentials before prompting you to grant access. After you grant access to your code repository, the system returns you to Docker Hub and the link is complete.

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



Linked Accounts & Services

Linked Accounts

These account links are currently used for Automated Builds, so that we can access your project lists and help you configure your Automated Builds. **Please note: A github/bitbucket account can be connected to only one docker hub account at a time.**

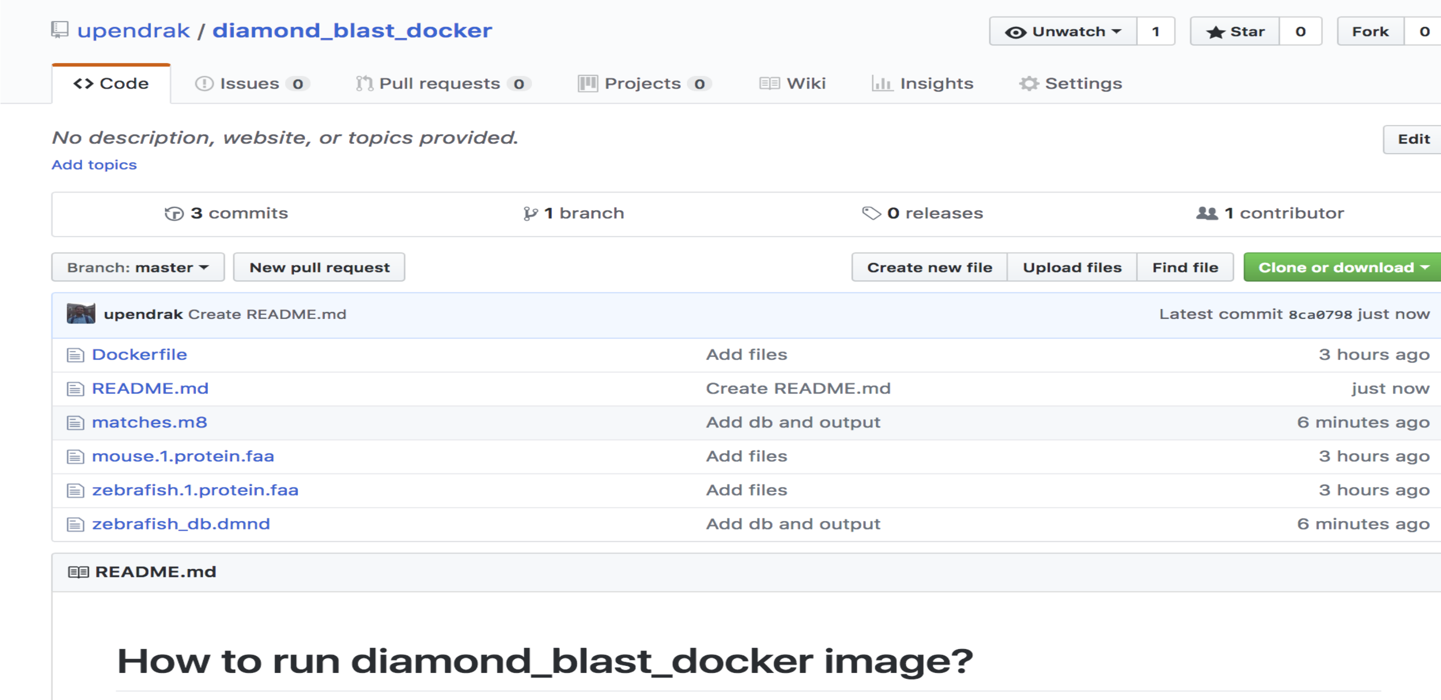
upendrak: read/write access

upendra_35: read/write access

Let's look at an example for Automated build.

Let's create an automatic build for our `diamond_blast_docker` using the instructions below.

Note: Hopefully you guys have a working Docker image from the exercies in a github repo, if you don't have a `diamond_blast_docker` github repo then fork my *repo <https://github.com/upendrak/diamond_blast_docker>*_ :



No description, website, or topics provided.

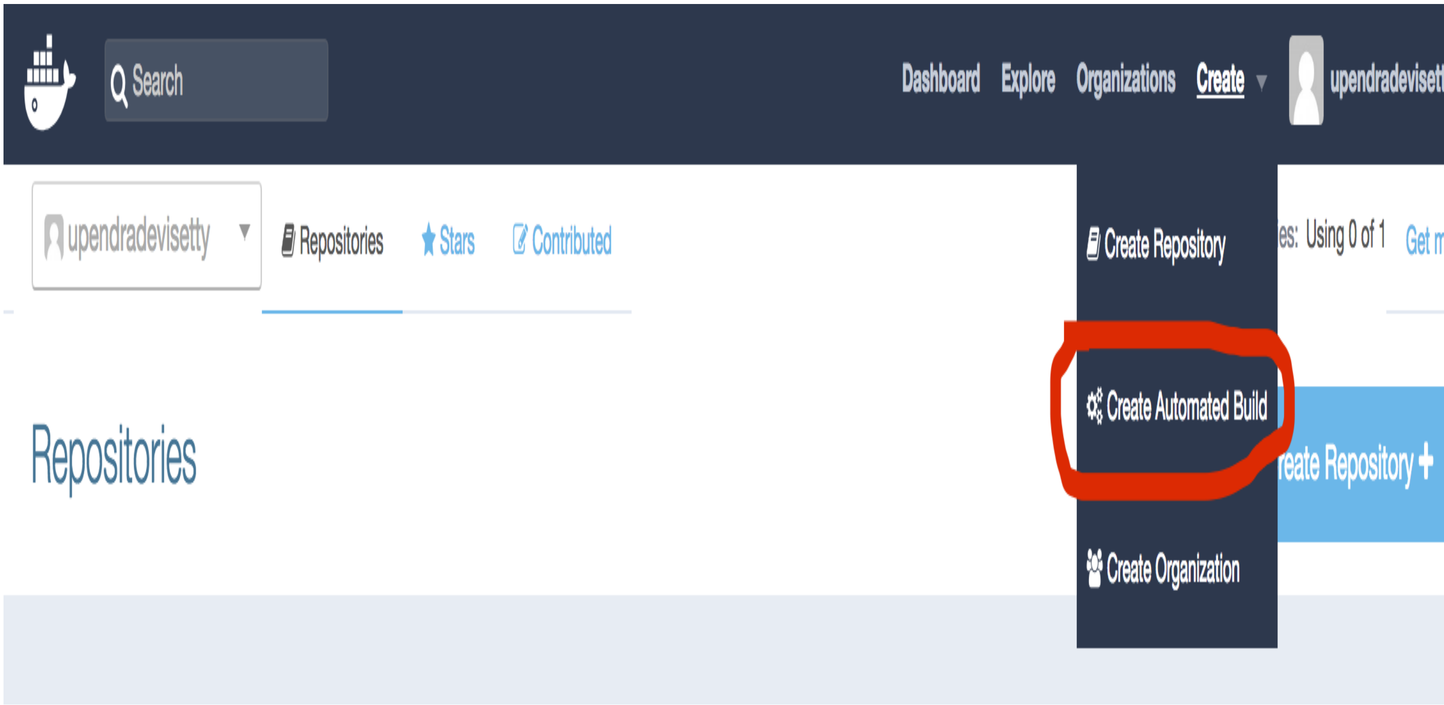
upendrak Create README.md — Latest commit 8ca0798 just now

Dockerfile	Add files	3 hours ago
README.md	Create README.md	just now
matches.m8	Add db and output	6 minutes ago
mouse.1.protein.faa	Add files	3 hours ago
zebrafish.1.protein.faa	Add files	3 hours ago
zebrafish_db.dmnd	Add db and output	6 minutes ago

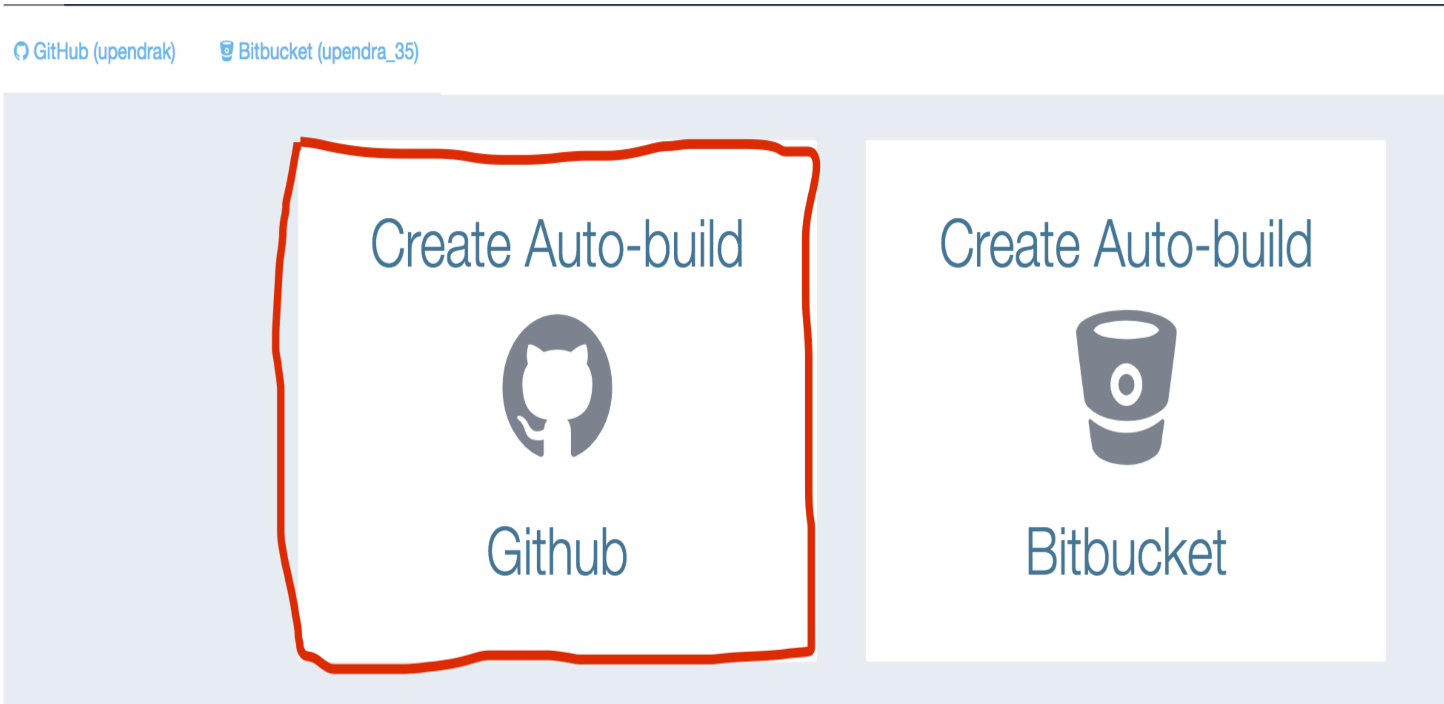
README.md

How to run diamond_blast_docker image?

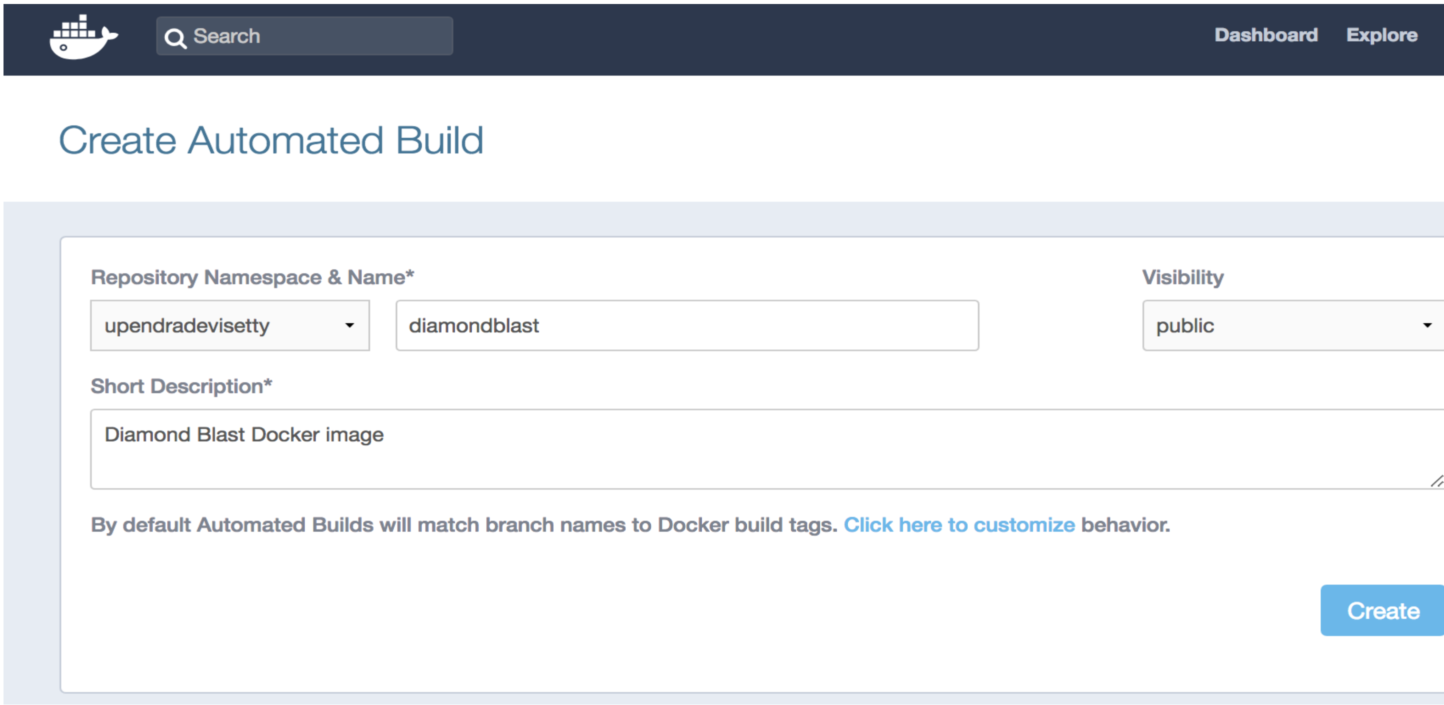
5. Select `Create` > `Create Automated Build` from Docker Hub.



- 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. 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.
- Pick the project to build. In this case `diamond_blast_docker`. Type in "diamondblast" in the Short Description box.



Create Automated Build

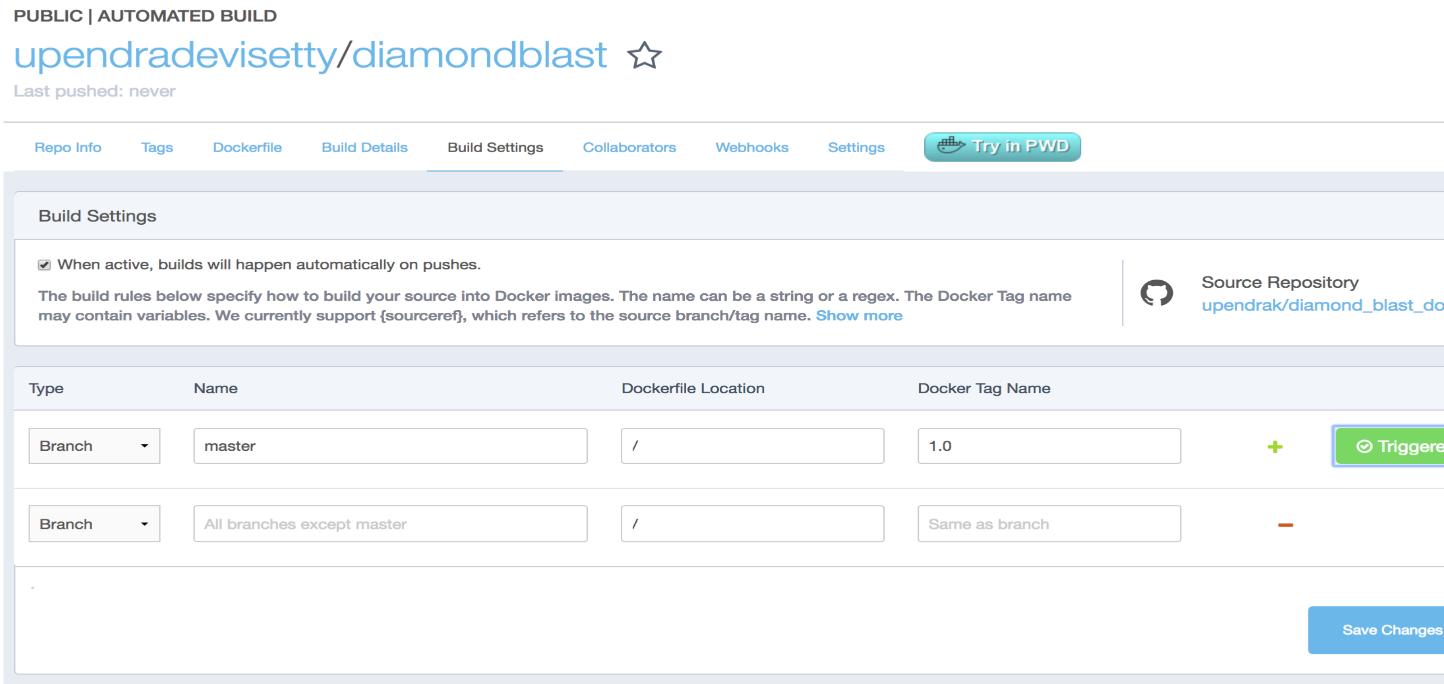
Repository Namespace & Name*: upendradevisetty / diamondblast

Visibility: public

Short Description*: Diamond Blast Docker image

By default Automated Builds will match branch names to Docker build tags. Click here to customize behavior.

- Next click on the `Build Settings` tab to customize the automated build.



PUBLIC | AUTOMATED BUILD

upendradevisetty/diamondblast ☆

Last pushed: never

Build Settings

☑ When active, builds will happen automatically on pushes.

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

Source Repository
upendrak/diamond_blast_do

Type	Name	Dockerfile Location	Docker Tag Name
Branch	master	/	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.

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

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.

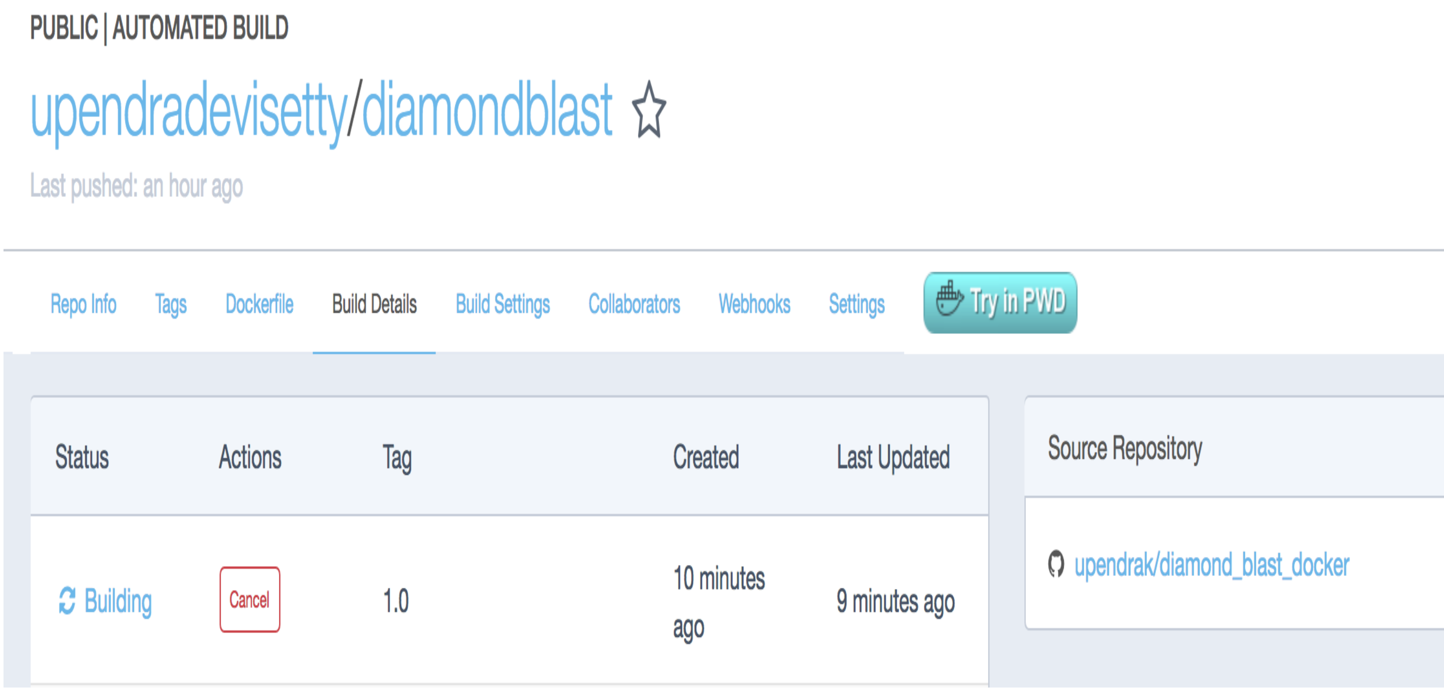
6. Review the build results

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

Navigate to the `Build Details` page.

Wait until your image build is done.

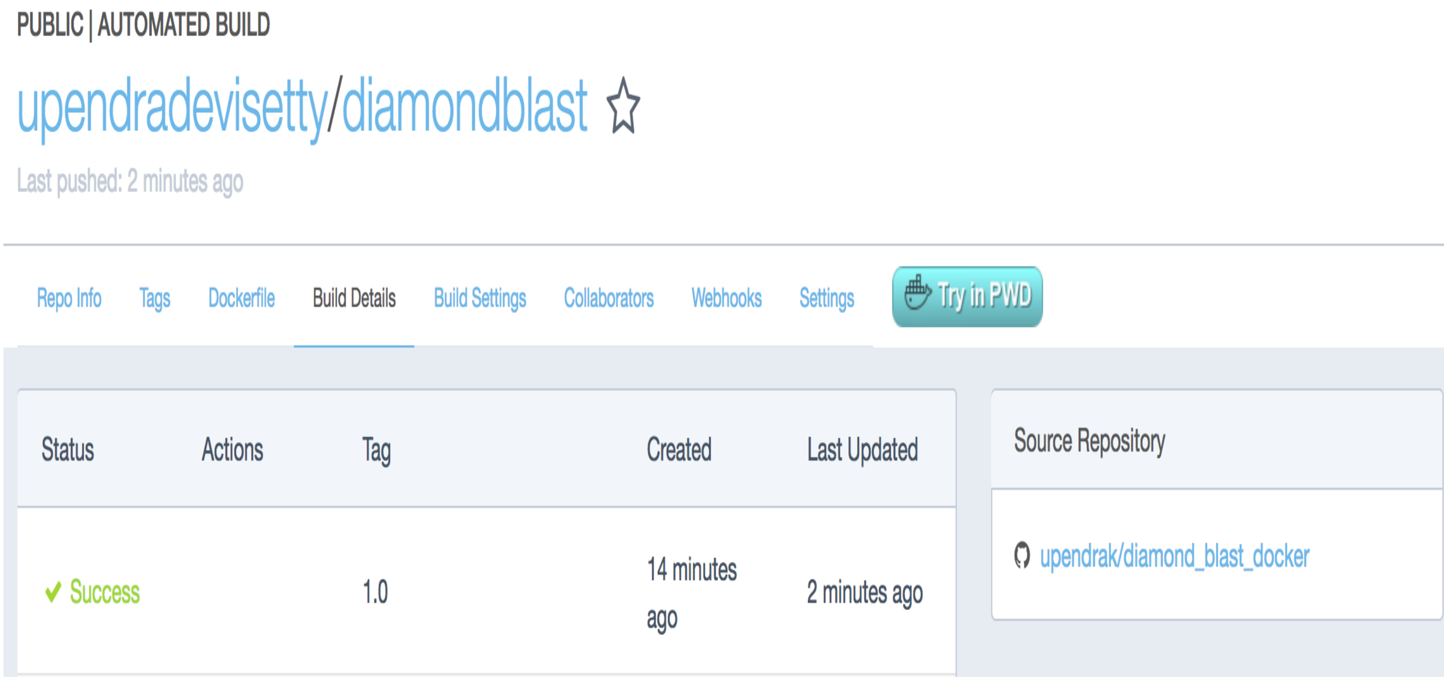
You may have to manually refresh the page and your build may take several minutes to complete.



Status	Actions	Tag	Created	Last Updated
Building	Cancel	1.0	10 minutes ago	9 minutes ago

Source Repository

upendrak/diamond_blast_docker



PUBLIC | AUTOMATED BUILD

upendradevisetty/diamondblast ☆

Last pushed: 2 minutes ago

Status	Actions	Tag	Created	Last Updated
✓ Success		1.0	14 minutes ago	2 minutes ago

Source Repository

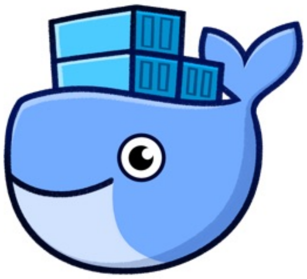
upendrak/diamond_blast_docker

5.6.2 5.1.1 Pull the image from Dockerhub registry

You have already seen couple of examples of pulling the image earlier. The basic syntax is `docker pull <docker image>`. We will see more examples of this in the advanced docker session

CHAPTER 6

Advanced Docker



This is the advanced session of Docker. The topics include Managing data in Docker containers, Docker compose for building multiple containers, Docker for Data science, etc.

6.1 1. Managing data in Docker

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

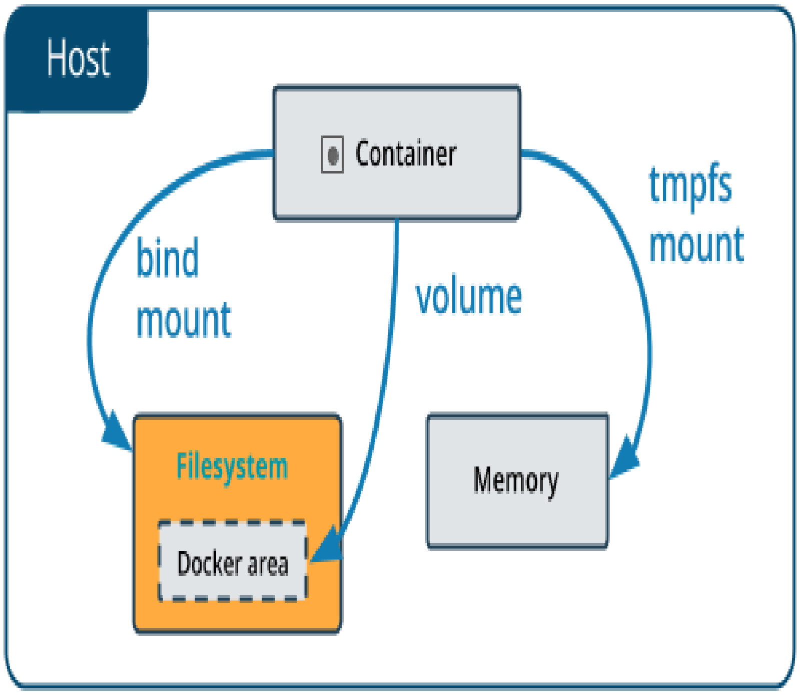
- The data doesn't persist when that container is no longer running, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere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

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

Note: If you're using Docker for Windows make sure you have shared your drive.

6.1.1 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:**
>
> 1. 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.
> 2. If you use 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. So make sure that file or directory exists before mounting it to the container.

1.1.1 Start a container with a bind mount

To demonstrate how bind mount works, we will use the `diamondblast:1.0` image that we created in docker intro section

- Download some test data for testing `diamondblast:1.0` image using bind mount method

```
$ mkdir diamond_test && cd diamond_test
$ wget https://raw.githubusercontent.com/upendrak/diamond_blast_docker/master/mouse.1.
↪protein.faa
$ wget https://raw.githubusercontent.com/upendrak/diamond_blast_docker/master/
↪zebrafish.1.protein.faa
```

- Making the blast database

In order to set up a reference database for DIAMOND, the makedb command needs to be executed with the following command line:

```
$ docker run --rm -v `pwd`:/data upendradevisetty/diamondblast:1.0 makedb --in /data/
↪zebrafish.1.protein.faa -d /data/zebrafish_db
```

This will create a binary DIAMOND database file with the specified name `zebrafish_db.dmnd` in the `diamond_test` direcotry

- Running the alignment

The alignment task may then be initiated using the blastp command like this:

```
$ docker run -v `pwd`:/data upendradevisetty/diamondblast:1.0 blastp --db /data/
↪zebrafish_db -q /data/mouse.1.protein.faa -o /data/matches.m
```

The output file here is specified with the `-o` option and named `matches.m8`. By default, it is generated in BLAST tabular format.

Note: `-v` Consists of three fields, separated by colon characters (:). - The first field is the path of the directory or file. - 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`.

You can use `docker inspect $(docker ps -lq) | grep -A 9 Mounts` to verify that the bind mount was created correctly. Look for the "Mounts" section

```
$ docker inspect $(docker ps -lq) | grep -A 9 Mounts
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/upendra/diamond_test",
        "Destination": "/data",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

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

6.2 2. Docker Compose for multi container apps

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see the list of features

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 and Redis. We will end up with a Flask container and a Redis container all on one host.

2.1.1 Clone the repo that container Docker compose code

```
$ git clone https://github.com/upendrak/compose_flask.git
$ cd compose_flask
```

In the github repo you will find 4 files - Dockerfile, requirements.txt file, app.py and finally docker-compose.yml file

Using Compose is a three-step process, which is best explained with a short example.

1. First, you define your app's environment with a `Dockerfile` so it can run anywhere
2. Next, you define the components that make up your app in `docker-compose.yml` so they can be run together in an isolated environment

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.

3. Lastly, run docker-compose up and Compose will start and run your entire app, determining the right order to start everything in, and building and pulling any images as necessary

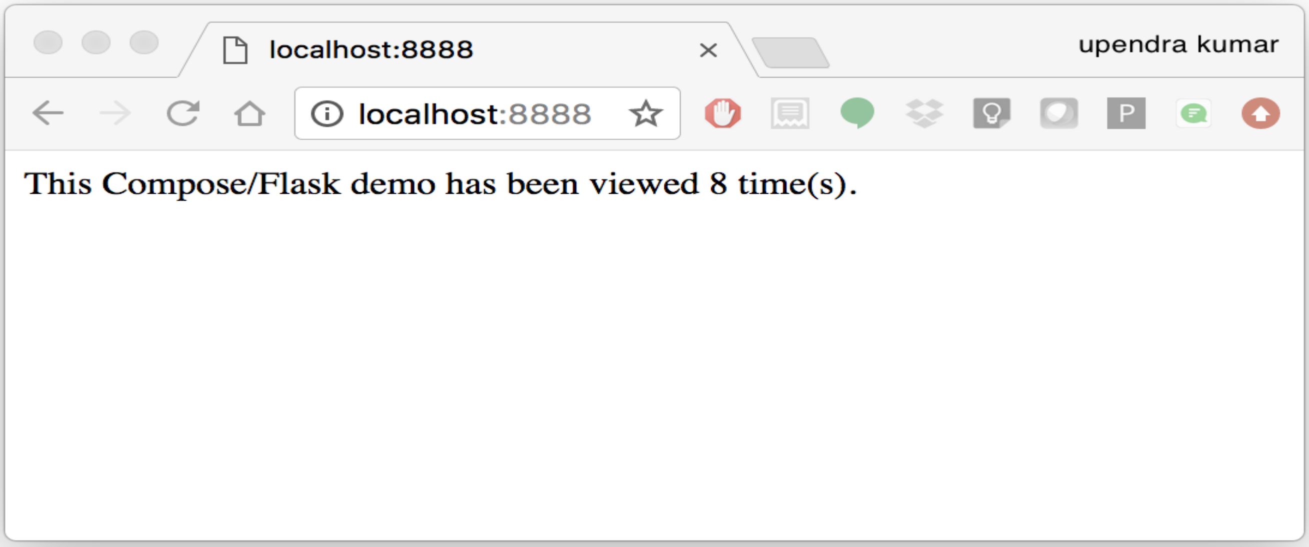
```
$ docker-compose up -d

Building web
Step 1/5 : FROM python:2.7
2.7: Pulling from library/python
f49cf87b52c1: Already exists
7b491c575b06: Already exists
b313b08bab3b: Already exists
51d6678c3f0e: Already exists
09f35bd58db2: Already exists
f7e0c30e74c6: Pull complete
c308c099d654: Pull complete
339478b61728: Pull complete
Digest: sha256:8cb593cb9cd1834429f0b4953a25617a8457e2c79b3e111c0f70bffd21acc467
Status: Downloaded newer image for python:2.7
 ---> 9e92c8430ba0
Step 2/5 : ADD . /code
 ---> 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)
```

(continues on next page)

(continued from previous page)

```
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 `<ipaddress>:8888`



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

6.3 3. Improving your data science workflow using Docker containers (Containerized Data Science)

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. We want data scientists using our platform to launch a Jupyter or RStudio or TensorFlow session in minutes, not hours. We also want them to have that fast user experience while still working in a governed, central architecture (rather than on their local machines).
- Containerization benefits both data science and IT/technical operations teams.
- Ultimately, 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. To start using containers and our library of curated images to do collaborative data science work, request a demo of our platform today.
- 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 thse in the following sections. . .

Note: The above code can be found in this github

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:

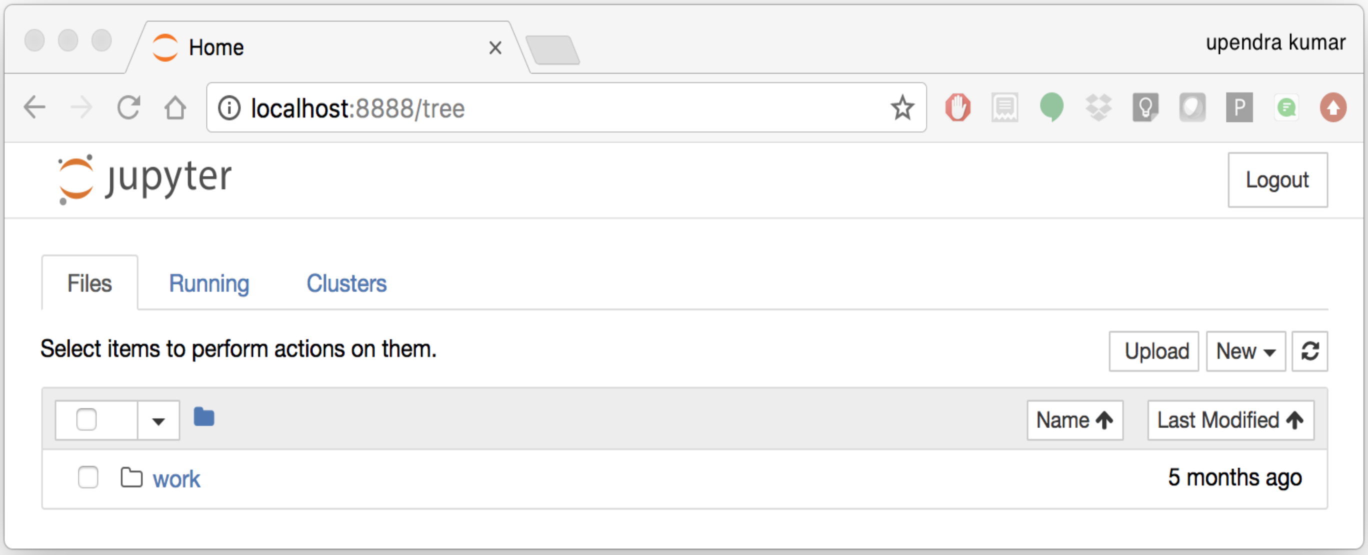
```
$ ezj
/usr/bin/python3
DEBUG: using python version 3
DEBUG: downloading anaconda binary, may take a few minutes
DEBUG: install Anaconda
PREFIX=/opt/anaconda3
installing: python-3.6.4-hc3d631a_1 ...
```

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

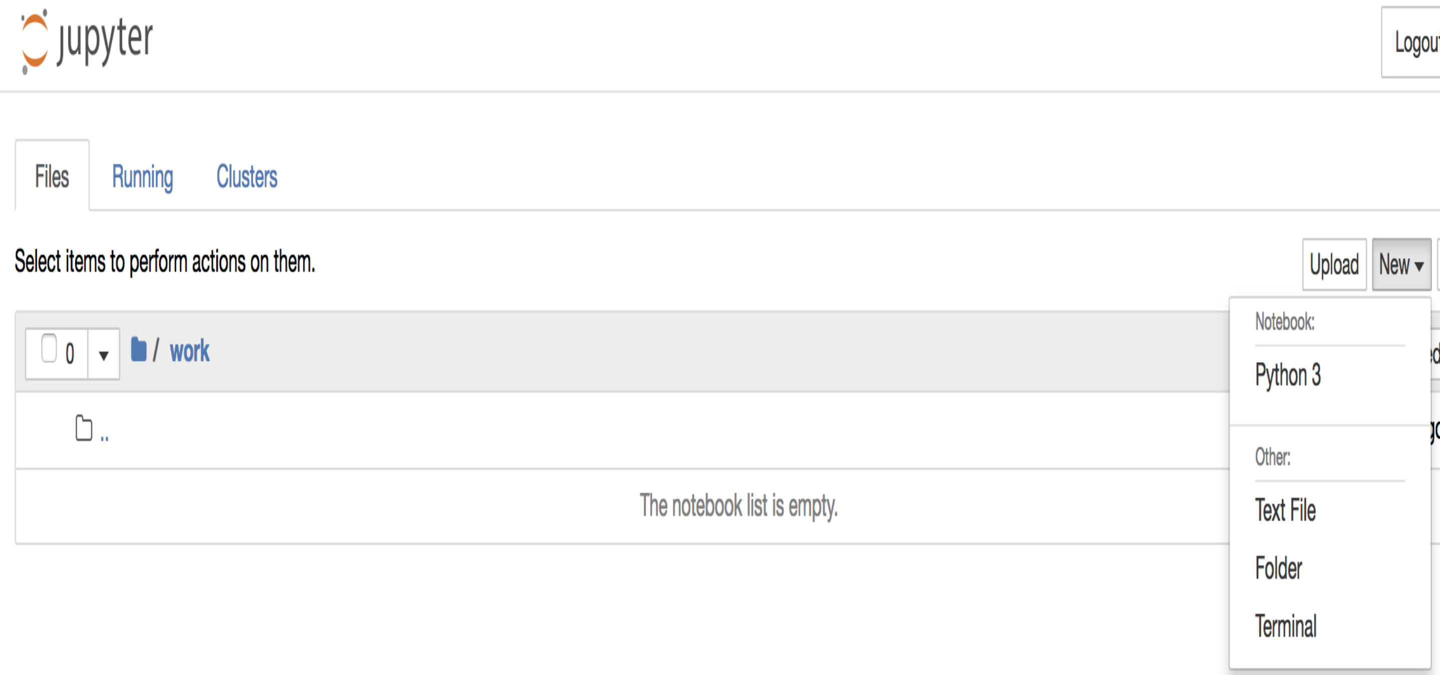
```
    Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://149.165.169.252:8888/?
↪token=13fffde91fc7006441e4e68656bb1385945fc26216b446be
```

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

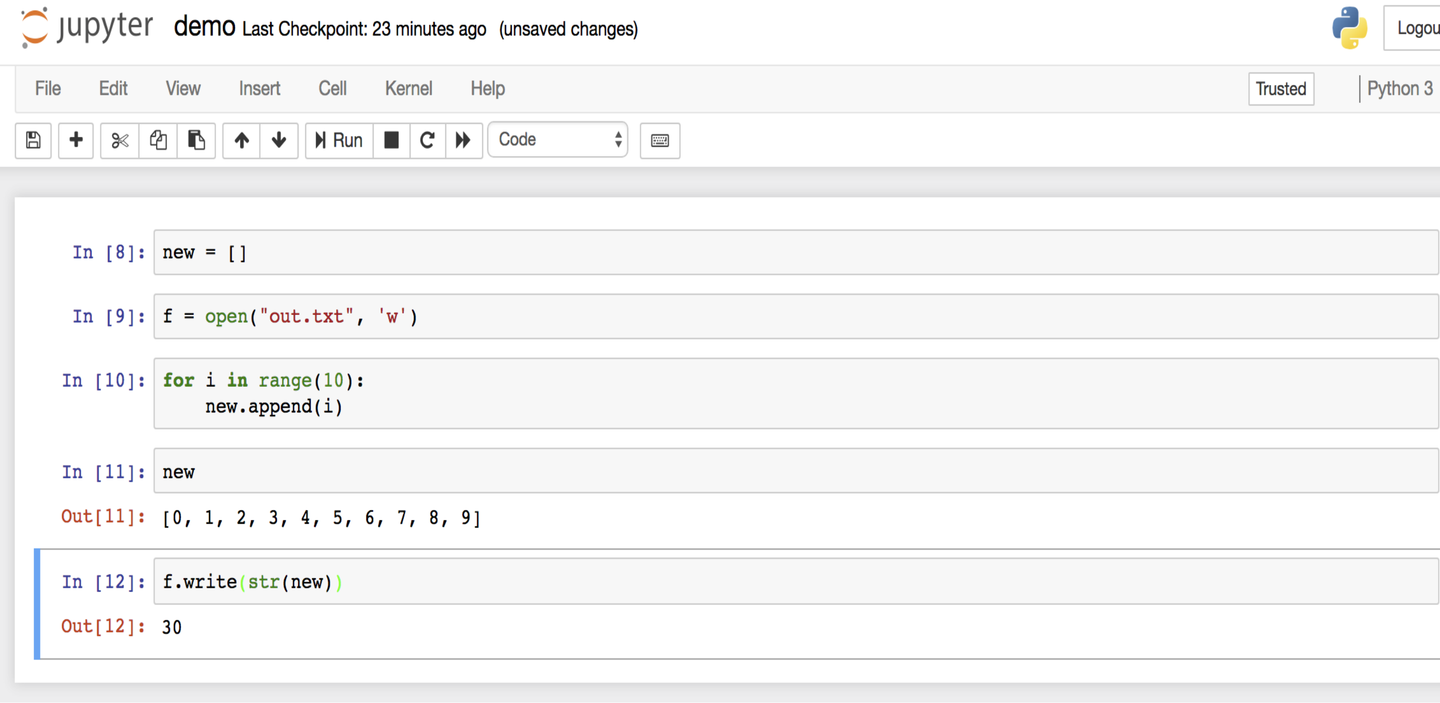
Once you've done that you should be greeted by your very own containerised Jupyter service!



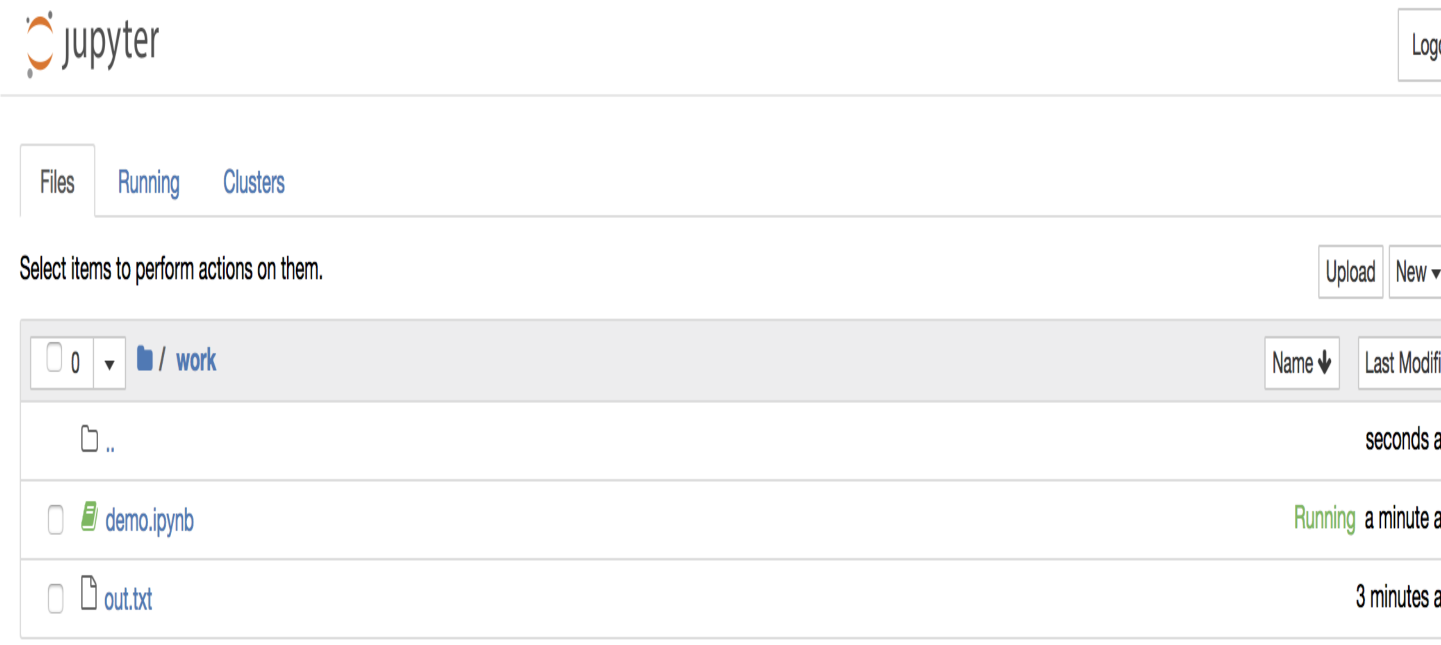
To create your first notebook, drill into the work directory and then click on the 'New' button on the right hand side and choose 'Python 3' to create a new Python 3 based Notebook.



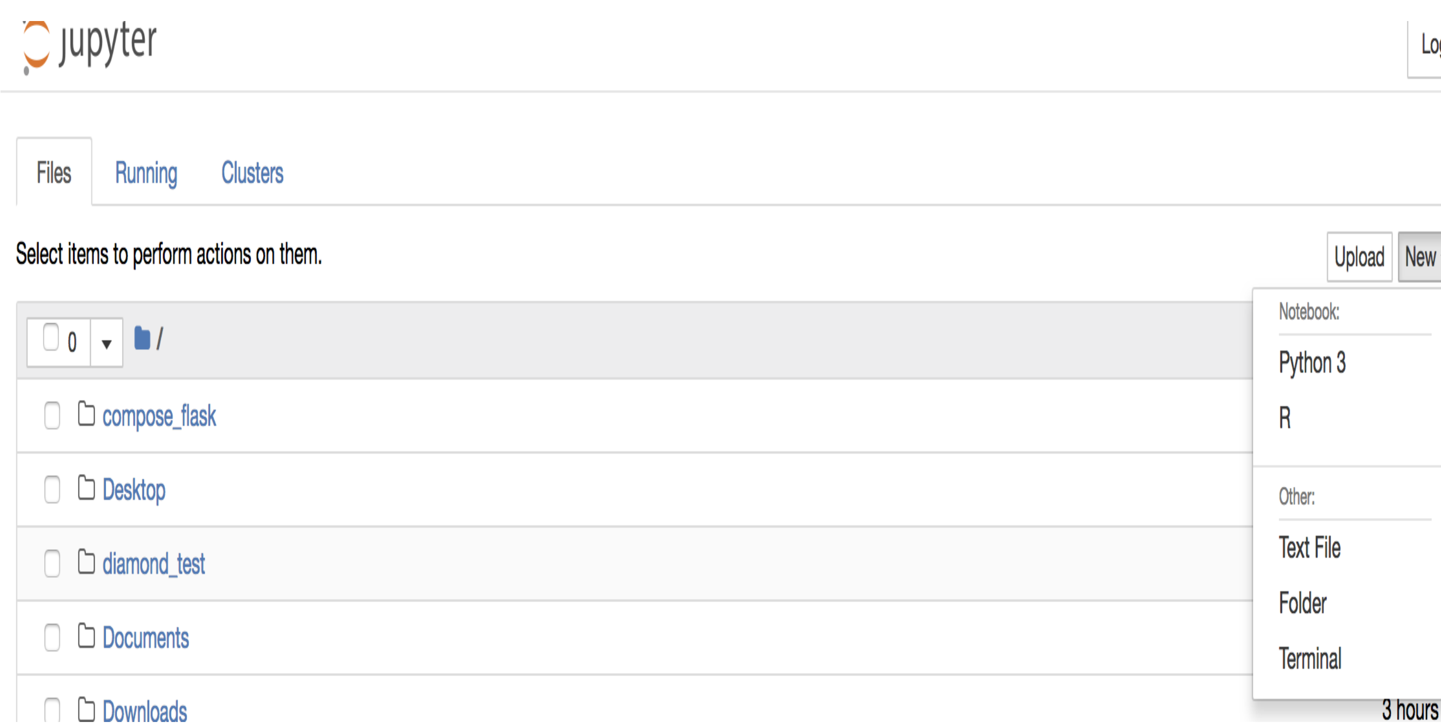
Now you can write your python code. Here is an example



```
In [8]: new = []

In [9]: f = open("out.txt", 'w')

In [10]: for i in range(10):
             new.append(i)

In [11]: new

Out[11]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [12]: f.write(str(new))

Out[12]: 30
```



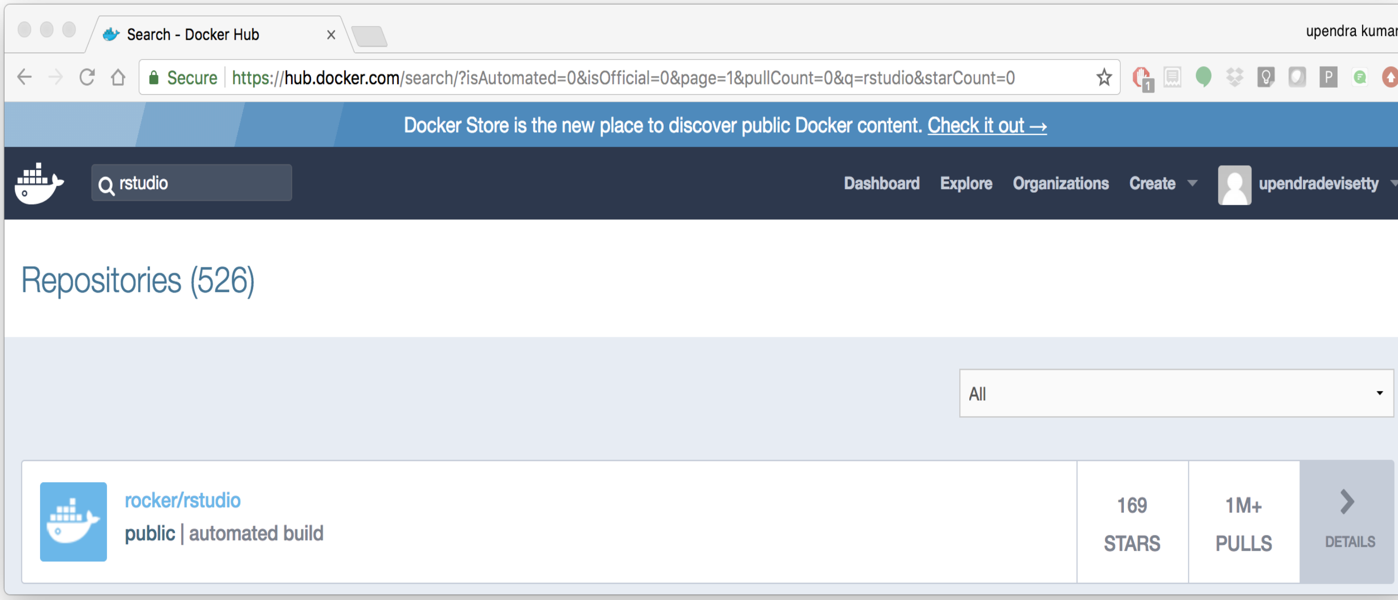
Note: If you want a *R* kernel in Jupyter notebook, then you should run it as `ezj -R`



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.

2. Launch a R-Studio container

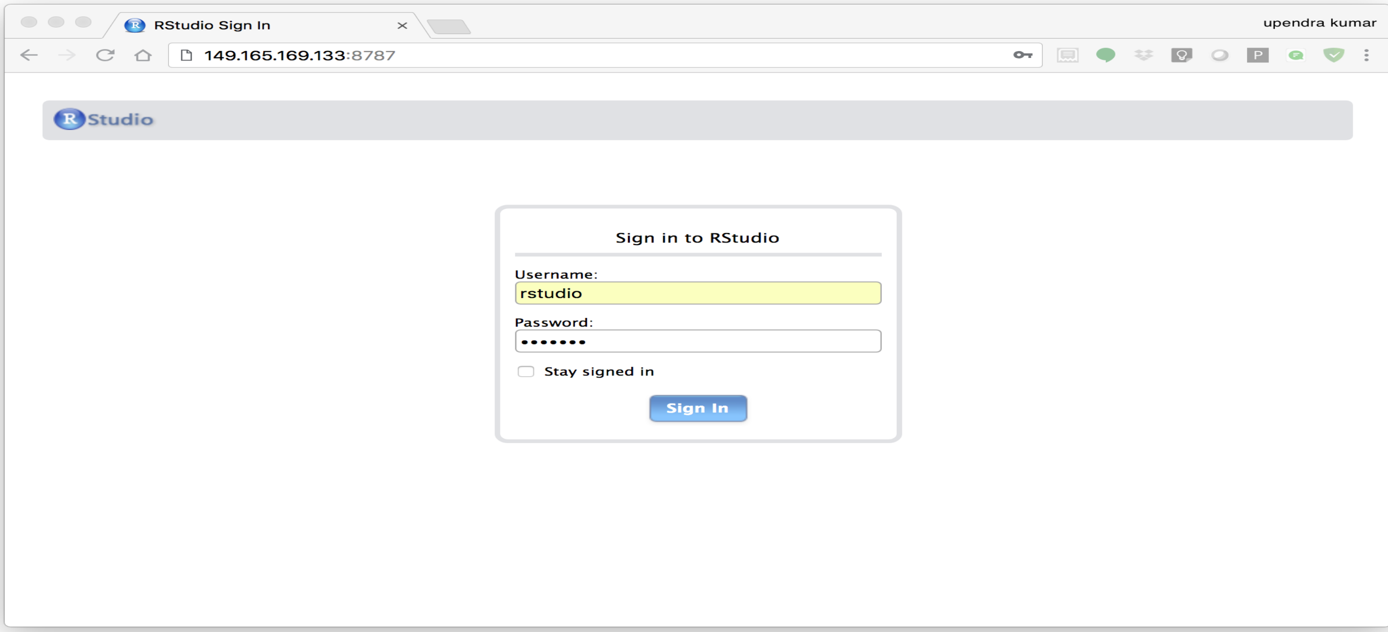
Next, we will see a Docker image from Rocker which will allow us to run RStudio inside the container and has many useful R packages already installed.


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 -v `pwd`:/data -p 8787:8787 rocker/rstudio:3.5.1
330242b370fa38c2158f3ba31b09fb5ef41bc64763baeebdeff10008f3d37186
```

Note: `-rm` ensures that when we quit the container, the container is deleted. If we did not do this, everytime we run a container, a version of it will be saved to our local computer. This can lead to the eventual wastage of a lot of disk space until we manually remove these containers.

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



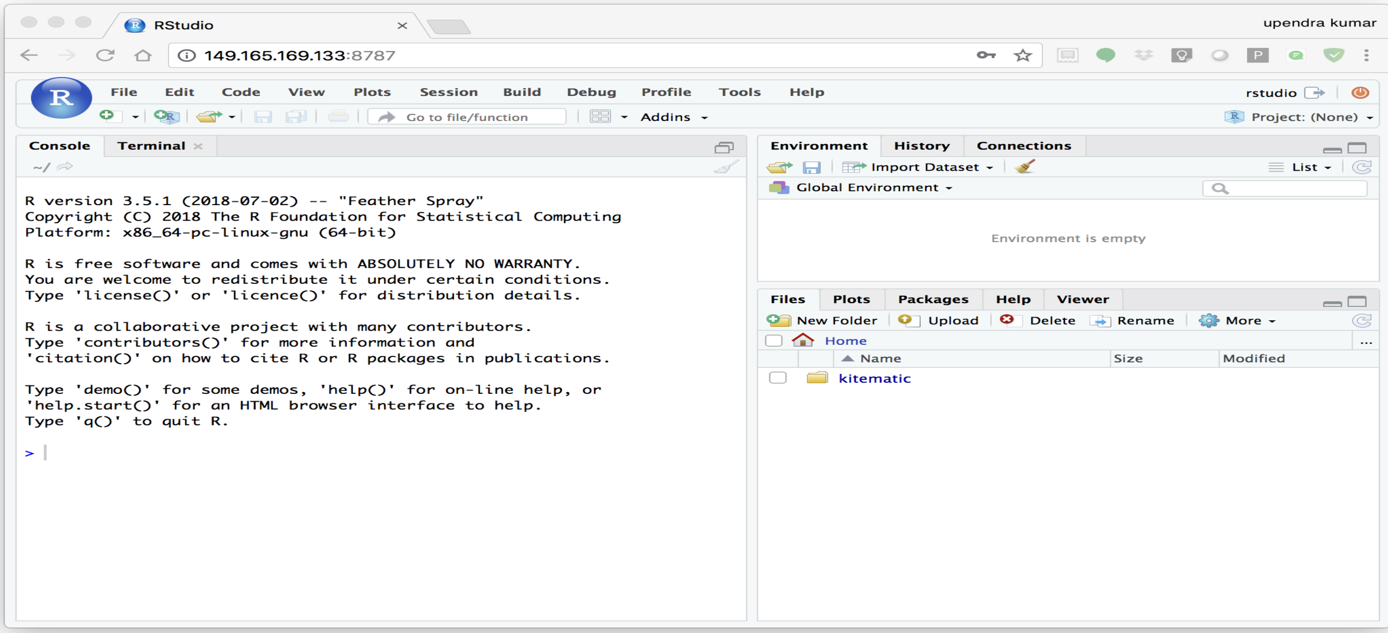
Sign in to RStudio

Username: rstudio

Password: •••••••

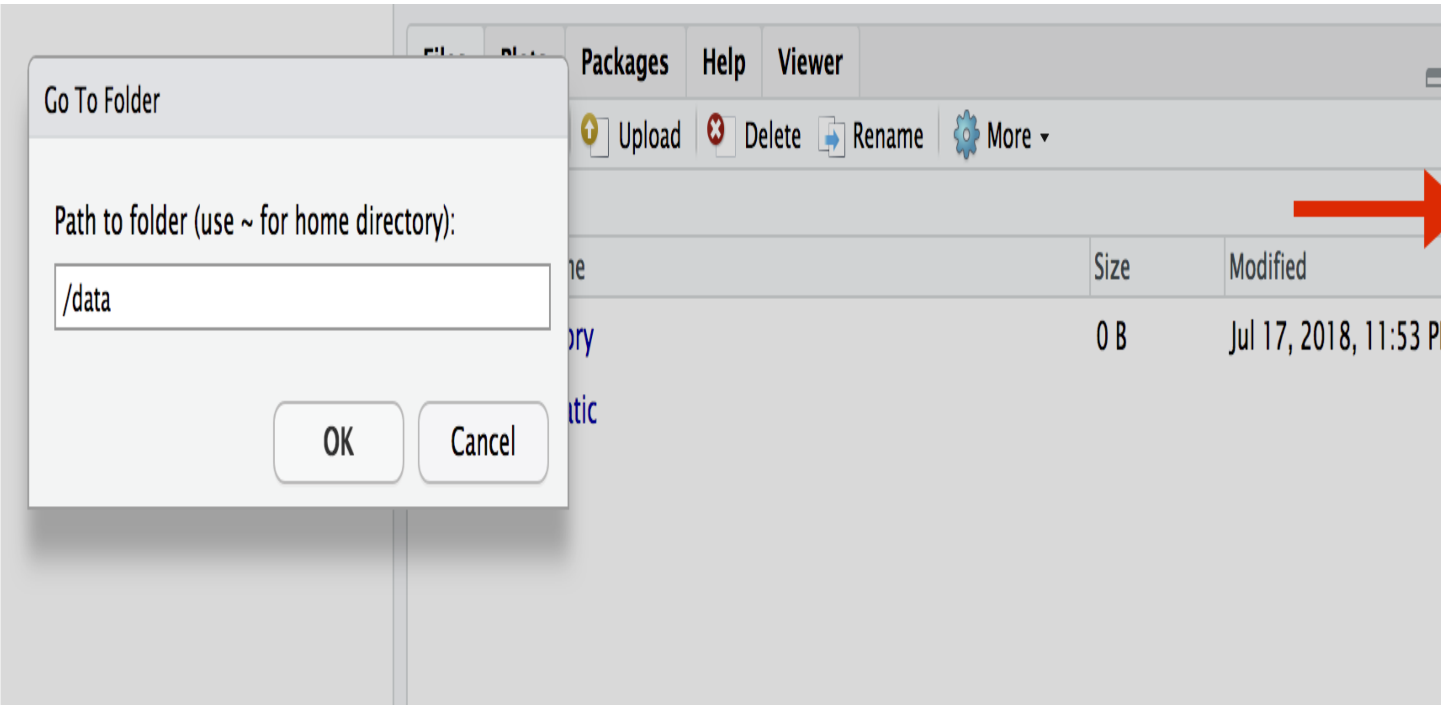
□ Stay signed in

Enter `rstudio` as username and password. Finally Rstudio shows up and you can run your R command from here



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
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.

> |
```

In order to see the directory that is currently mounted on inside the container, click . . . in the right hand down window of Rstudio which open up a dialog box. Now enter */data* which is the location of mounted directory inside the container

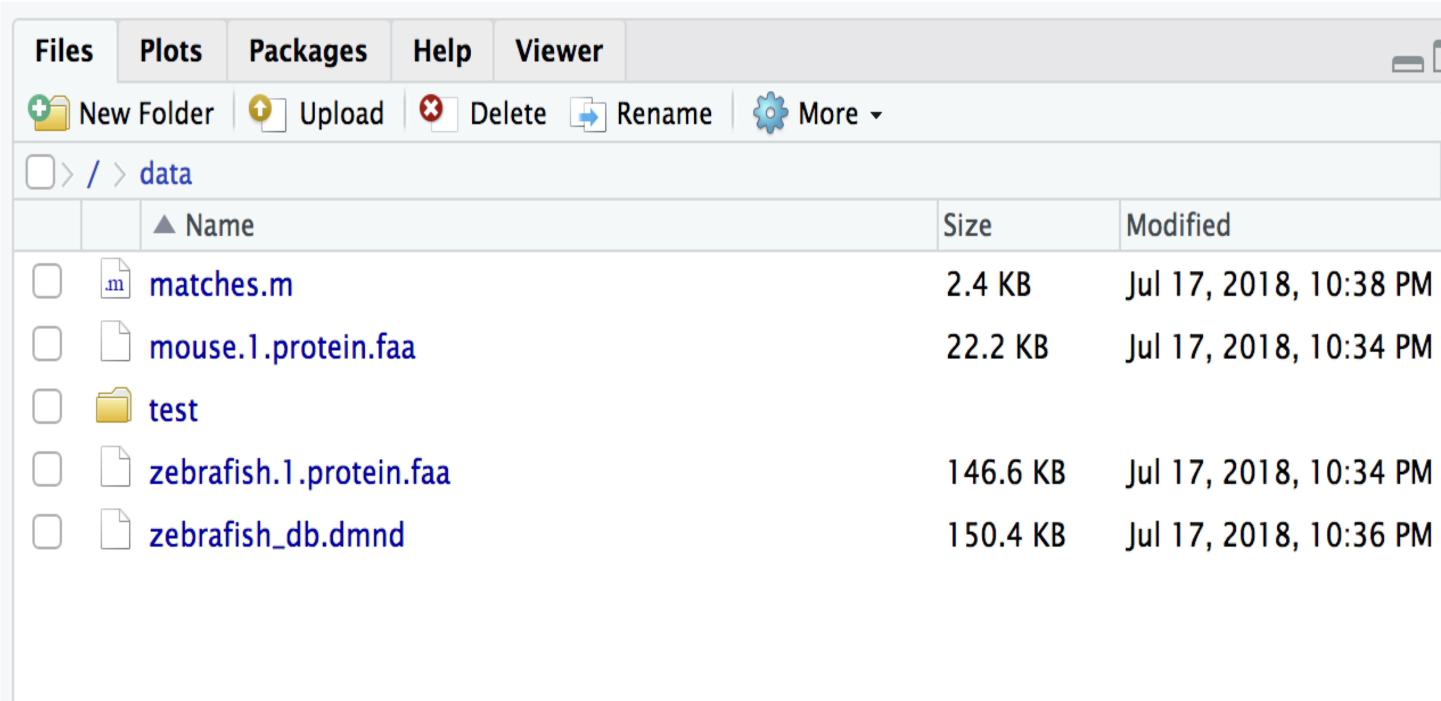


Go To Folder

Path to folder (use ~ for home directory):

/data

Size	Modified
0 B	Jul 17, 2018, 11:53 P

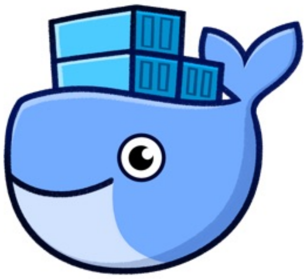


/ > data

Name	Size	Modified
matches.m	2.4 KB	Jul 17, 2018, 10:38 PM
mouse.1.protein.faa	22.2 KB	Jul 17, 2018, 10:34 PM
test		
zebrafish.1.protein.faa	146.6 KB	Jul 17, 2018, 10:34 PM
zebrafish_db.dmnd	150.4 KB	Jul 17, 2018, 10:36 PM

CHAPTER 7

Additional Docker Demo's



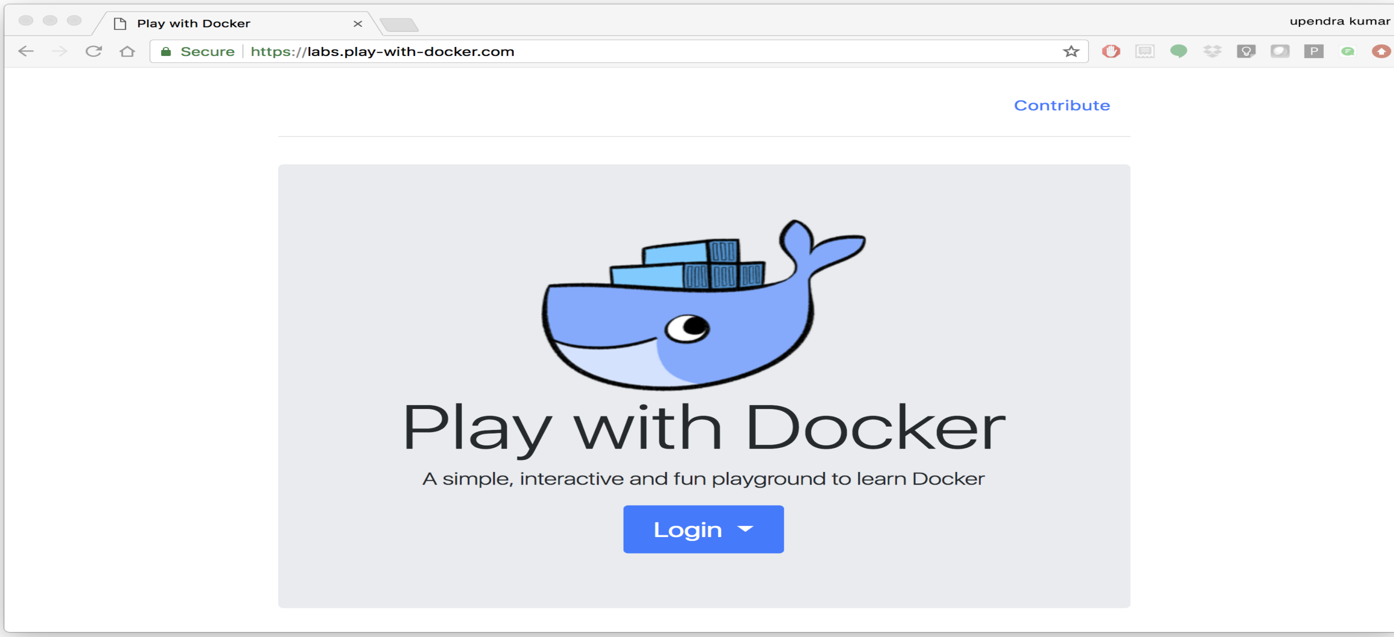
7.1 1. Play-with-docker (PWD)

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.

7.1.1 1.1 Installation

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

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


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

7.2 2. Portainer

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

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

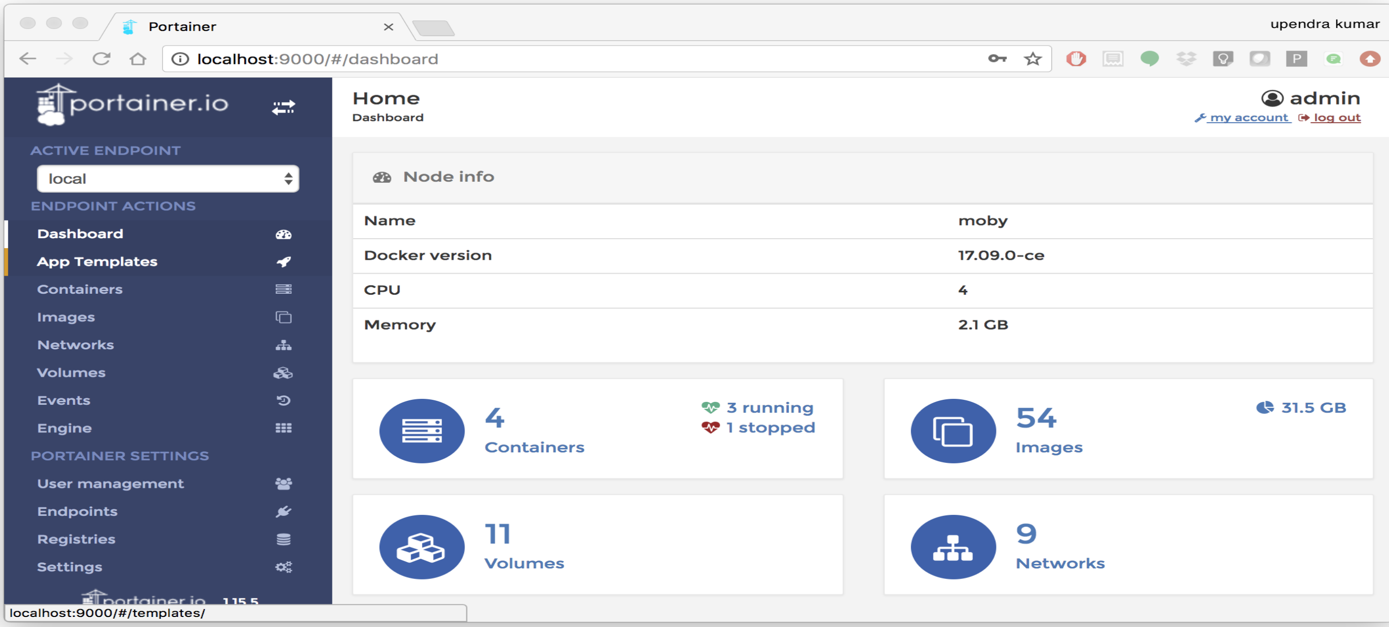
7.2.1 2.1 Installation

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

CHAPTER 8

Introduction to Singularity

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.

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 developing web applications could be helpful but is not required.

Note:

> *Important*: Docker and Singularity are friends but they have distinct differences.

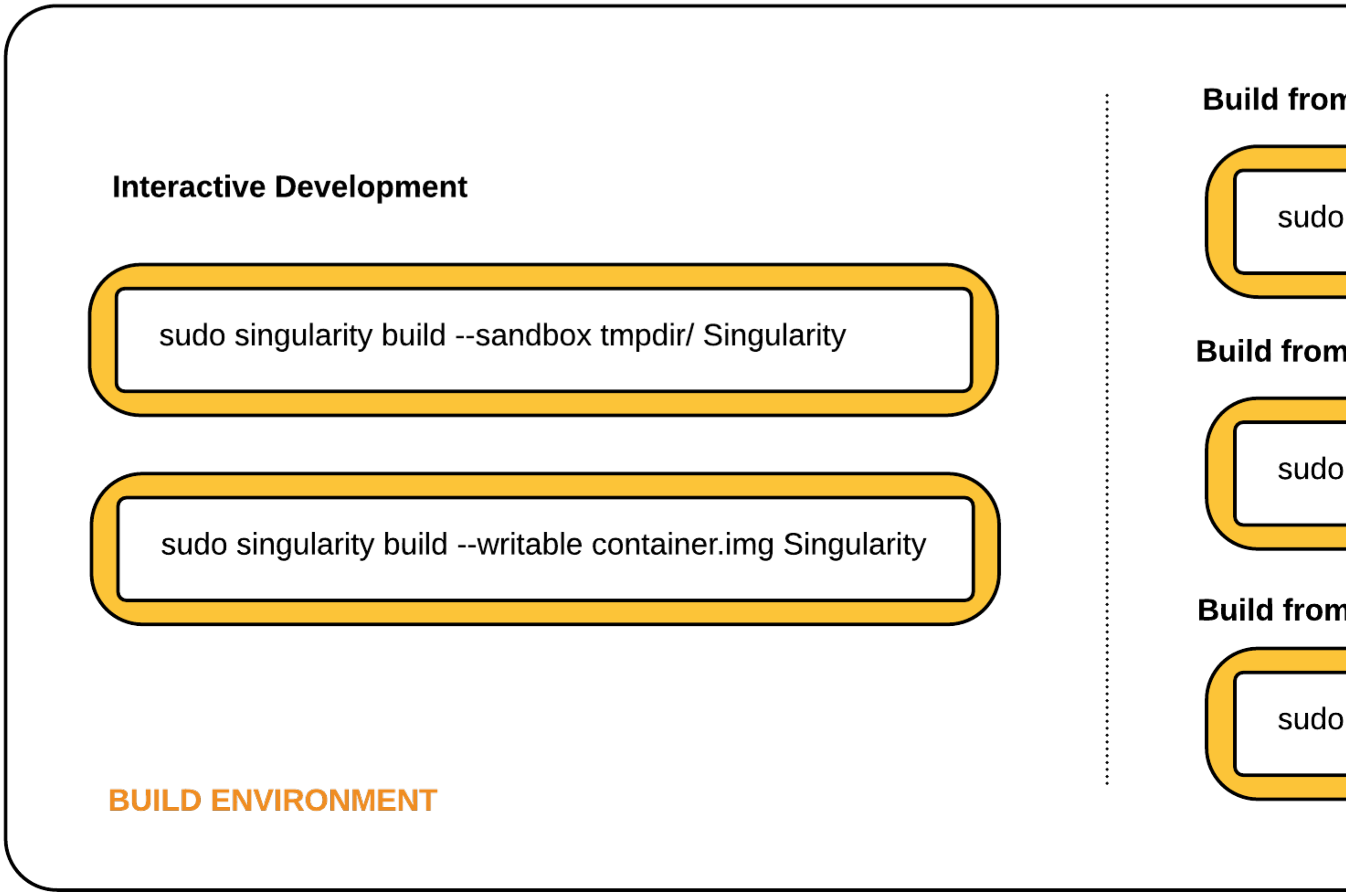
For more information see: Singularity Related Resources

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

Singularity:

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

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


Interactive Development
sudo singularity build --sandbox tmpdir/ Singularity
sudo singularity build --writable container.img Singularity
BUILD ENVIRONMENT
Build from
sudo
Build from
sudo
Build from
sudo

8.2 2. Singularity installation

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

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

8.2.1 Exercise 1 (15-20 mins)

8.2.2 2.1 Setting up your Laptop

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

- Windows
- Linux
- Mac

Note: on Mac OS X the dependency VagrantBox is required to run Singularity

8.2.3 2.2 HPC

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

```
$ module avail
```

If Singularity is installed:

```
$ module load singularity
```

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

8.2.4 2.3 XSEDE Jetstream / CyVerse Atmosphere Clouds

CyVerse staff have deployed an Ansible playbooks called *ez* installation which includes Singularity that only requires you to type a short line of code.

Start a featured instance on Atmosphere or Jetstream.

Type in the following:

```
$ ezs

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

(continues on next page)

(continued from previous page)

```
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
```

8.2.5 2.4 Check Installation

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

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

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

View the Singularity help:

```
$ singularity --help

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

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

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

CONTAINER USAGE COMMANDS:
    exec       Execute a command within container
    run        Launch a 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>
```

(continues on next page)

(continued from previous page)

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

8.3 3. Downloading Singularity containers

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

8.3.1 Exercise 2 (~10 mins)

8.3.2 3.1: Pulling a Container from Singularity Hub

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

Container Registries:

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

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

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

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

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

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

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

Singularity ubuntu_test.simg:~> cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
```

(continues on next page)

(continued from previous page)

```
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04 LTS"
NAME="Ubuntu"
VERSION="14.04, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
Singularity ubuntu_test.simg:~>
```

8.3.3 3.2: Pulling containers from Docker Hub

This example pulls a container from DockerHub

Build to your container by pulling an image from Docker:

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

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

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

```
$ singularity pull --name ubuntu_docker.simg docker://ubuntu
Importing: /home/***/.singularity/docker/
↪sha256:c71a6f8e13782fed125f2247931c3eb20cc0e6428a5d79edb546f1f1405f0e49.tar.gz
Importing: /home/***/.singularity/docker/
↪sha256:4be3072e5a37392e32f632bb234c0b461ff5675ab7e362afad6359fbd36884af.tar.gz
Importing: /home/***/.singularity/docker/
↪sha256:06c6d2f5970057aef3aef6da60f0fde280db1c077f0cd88ca33ec1a70a9c7b58.tar.gz
```

(continues on next page)

(continued from previous page)

```
Importing: /home/***/.singularity/metadata/
↪sha256:c6a9ef4b9995d615851d7786fbc2fe72f72321bee1a87d66919b881a0336525a.tar.gz
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Building Singularity image...
Singularity container built: ./ubuntu_docker.simg
Cleaning up...
Done. Container is at: ./ubuntu_docker.simg
```

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

8.3.4 3.3: Keeping track of downloaded containers

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

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

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

```
$ sudo mkdir tmp
$ sudo mkdir scratch

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

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

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

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

8.4 4. Building Singularity containers

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

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

Create Container and add content to it:

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

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

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

Create a container using a custom Singularity file:

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

In the above command:

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

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

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

Docker image path: index.docker.io/library/ubuntu:latest
Cache folder set to /root/.singularity/docker
Importing: base Singularity environment
Importing: /root/.singularity/docker/
↪sha256:1be7f2b886e89a58e59c4e685fcc5905a26ddef3201f290b96f1eff7d778e122.tar.gz
Importing: /root/.singularity/docker/
↪sha256:6fbc4a21b806838b63b774b338c6ad19d696a9e655f50b4e358cc4006c3baa79.tar.gz
Importing: /root/.singularity/docker/
↪sha256:c71a6f8e13782fed125f2247931c3eb20cc0e6428a5d79edb546f1f1405f0e49.tar.gz
Importing: /root/.singularity/docker/
↪sha256:4be3072e5a37392e32f632bb234c0b461ff5675ab7e362afad6359fbd36884af.tar.gz
Importing: /root/.singularity/docker/
↪sha256:06c6d2f5970057aef3aef6da60f0fde280db1c077f0cd88ca33ec1a70a9c7b58.tar.gz
Importing: /root/.singularity/metadata/
↪sha256:c6a9ef4b9995d615851d7786fbc2fe72f72321bee1a87d66919b881a0336525a.tar.gz
```

(continues on next page)

(continued from previous page)

```
Creating empty Singularity writable container 120MB
Creating empty 150MiB image file: ubuntu.simg
Formatting image with ext3 file system
Image is done: ubuntu.simg
Building Singularity image...
Singularity container built: ubuntu.simg
Cleaning up...

$ singularity shell ubuntu.simg

Singularity: Invoking an interactive shell within container...

Singularity ubuntu.simg:~> apt-get update

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

$ sudo singularity shell ubuntu.simg

Singularity: Invoking an interactive shell within container...

Singularity ubuntu.simg:~> apt-get update

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

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

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

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

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

8.4.1 4.1: Docker2Singularity & Singularity2Docker

One of the other features of Singularity is the ability to convert Docker containers to Singularity Containers, and Singularity containers to Docker containers

8.4.2 4.2: Exercise 3 (30 minutes): Create the Singularity file

Recipes can use any number of container registries for bootstrapping a container.

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

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

- The Header

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

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

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

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

Using a *localimage* to build:

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

Using CentOS-like container:

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

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

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

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

- %help - create text for a help menu associated with your container
- %setup - executed on the host system outside of the container, after the base OS has been installed.
- %files - copy files from your host system into the container
- %labels - store metadata in the container
- %environment - loads environment variables at the time the container is run (not built)
- %post - set environment variables during the build
- %runscript - executes a script when the container runs
- %test - runs a test on the build of the container
- Apps

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

```
Bootstrap: docker
From: ubuntu

% environment

%labels

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

%apprun foo
    exec echo "RUNNING FOO"

%applabels foo
    BESTAPP=FOO
    export BESTAPP

%appinstall foo
    touch foo.exec

%appenv foo
    SOFTWARE=foo
    export SOFTWARE

%apphelp foo
    This is the help for foo.

%appfiles foo
    avocados.txt


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

%apphelp bar
    This is the help for bar.
```

(continues on next page)

(continued from previous page)

```
%applabels bar
    BESTAPP=BAR
    export BESTAPP

%appinstall bar
    touch bar.exec

%appenv bar
    SOFTWARE=bar
    export SOFTWARE
```

- Setting up Singularity file system

%help section can be as verbose as you want

```
Bootstrap: docker
From: ubuntu

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

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

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

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

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

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

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

```
BootStrap: docker
From: ubuntu:16.04

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

    # create bind points for additional storage
    mkdir /scratch

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

%runscript
    fortune | cowsay | lolcat

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

Build the container:

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

Run the container:

```
singularity run cowsay.simg
```

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

8.5 5. Running Singularity containers

Commands:

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

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

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

inspect - inspects the container.

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

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

8.5.1 5.1 Using the *exec* command

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

8.5.2 5.2 Using the *shell* command

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

8.5.3 5.3 Using the *run* command

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

8.5.4 5.4 Using the *inspect* command

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

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

$ singularity inspect vsoch-hello-world-master-latest.simg
{
    "org.label-schema.usage.singularity.deffile.bootstrap": "docker",
    "MAINTAINER": "vanessasaur",
    "org.label-schema.usage.singularity.deffile": "Singularity",
```

(continues on next page)

(continued from previous page)

```
    "org.label-schema.schema-version": "1.0",
    "WHATAMI": "dinosaur",
    "org.label-schema.usage.singularity.deffile.from": "ubuntu:14.04",
    "org.label-schema.build-date": "2017-10-15T12:52:56+00:00",
    "org.label-schema.usage.singularity.version": "2.4-feature-squashbuild-secbuild.
↪g780c84d",
    "org.label-schema.build-size": "333MB"
}
```

8.5.5 5.5 Using the *–sandbox* and *–writable* commands

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

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

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

You can convert these images to writable versions using the *–writable* and *–sandbox* commands.

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

```
$ singularity build --sandbox lolcow/ shub://GodloveD/lolcow
WARNING: Building sandbox as non-root may result in wrong file permissions
Cache folder set to /home/.../.singularity/shub
Progress |===================================| 100.0%
Building from local image: /home/.../.singularity/shub/GodloveD-lolcow-master-latest.
↪simg
WARNING: Building container as an unprivileged user. If you run this container as root
WARNING: it may be missing some functionality.
Singularity container built: lolcow/
Cleaning up...
@vm142-73:~$ cd lolcow/
@vm142-73:~/lolcow$ ls
bin  boot  dev  environment  etc  home  lib  lib64  media  mnt  opt  proc  run  sbin ␣
↪singularity  srv  sys  tmp  usr  var
```

8.5.6 5.6 Test

Singularity can test the build of your container.

You can bypass the test by using *–no-test*.

8.5.7 5.7 Bind Paths

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

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

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

8.5.8 5.8 Overlay

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

First, download a container.

Next, create a new image in the ext3 format.

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

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

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

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

8.6 6. Singularity-Hub

You can host containers on Singularity’s own container registry Singularity-Hub

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

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

CHAPTER 9

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.

9.1 1. Using HPC Environments

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

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

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

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

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

9.1.1 How do HPC systems fit into the development workflow?

A few things to consider when using HPC systems:

1. Using ‘sudo’ is not allowed on HPC systems, and building a Singularity container from scratch requires sudo. That means you have to build your containers on a different development system. You can pull a docker image on HPC systems.
2. If you need to edit text files, command line text editors don’t support using a mouse, so working efficiently has a learning curve. There are text editors that support editing files over SSH. This lets you use a local text editor and just save the changes to the HPC system.
3. Singularity has changed image formats. Depending on the version of Singularity running on the HPC system, new squashFS or .simg formats may not work. The images take a lot less space
4. You can’t run Docker containers - security stuff!

9.1.2 Tutorial #1

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

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

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum wget
# best to build up container using kickstart mentality.
# ie, to add more packages to image,
# re-run bootstrap command again.
# bootstrap on existing image will build on top of it, not overwriting it/restarting␣
↪from scratch
# singularity .def file is like kickstart file
# unix commands can be run, but if there is any error, the bootstrap process ends
%setup
 # commands to be executed on host outside container during bootstrap
%post
  # commands to be executed inside container during bootstrap
  # add python and install some packages
  yum -y install vim wget python epel-release
  yum -y install python-pip
   # install tensorflow
  pip install --upgrade pip
  pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/
↪tensorflow-0.9.0-cp27-none-linux_x86_64.whl
  pip install --upgrade numpy scipy astropy
```

(continues on next page)

(continued from previous page)

```
   # create bind points for storage.
   mkdir /extra
   mkdir /xdisk
   exit 0
%runscript
 # commands to be executed when the container runs
 echo "Arguments received: $*"
 exec /usr/bin/python "$@"
%test
 # commands to be executed within container at close of bootstrap process
```

Create container

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

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

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

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

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

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

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

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

```
qsub astropy.pbs
```

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

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

9.2 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 which are available on Ocelote) with some non-overlapping feature sets. If the host and container are running different MPI implementations, or even different versions of the same implementation, tragedy may ensue.

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

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

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

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

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


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


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

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

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

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

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

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

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

9.3 3. Singularity and GPU Computing

GPU support in Singularity is fantastic

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

9.3.1 Tutorial #2

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

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

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

9.3.2 Tutorial #3

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

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

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

CHAPTER 10

Booting an Atmosphere computer instance for your use!

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

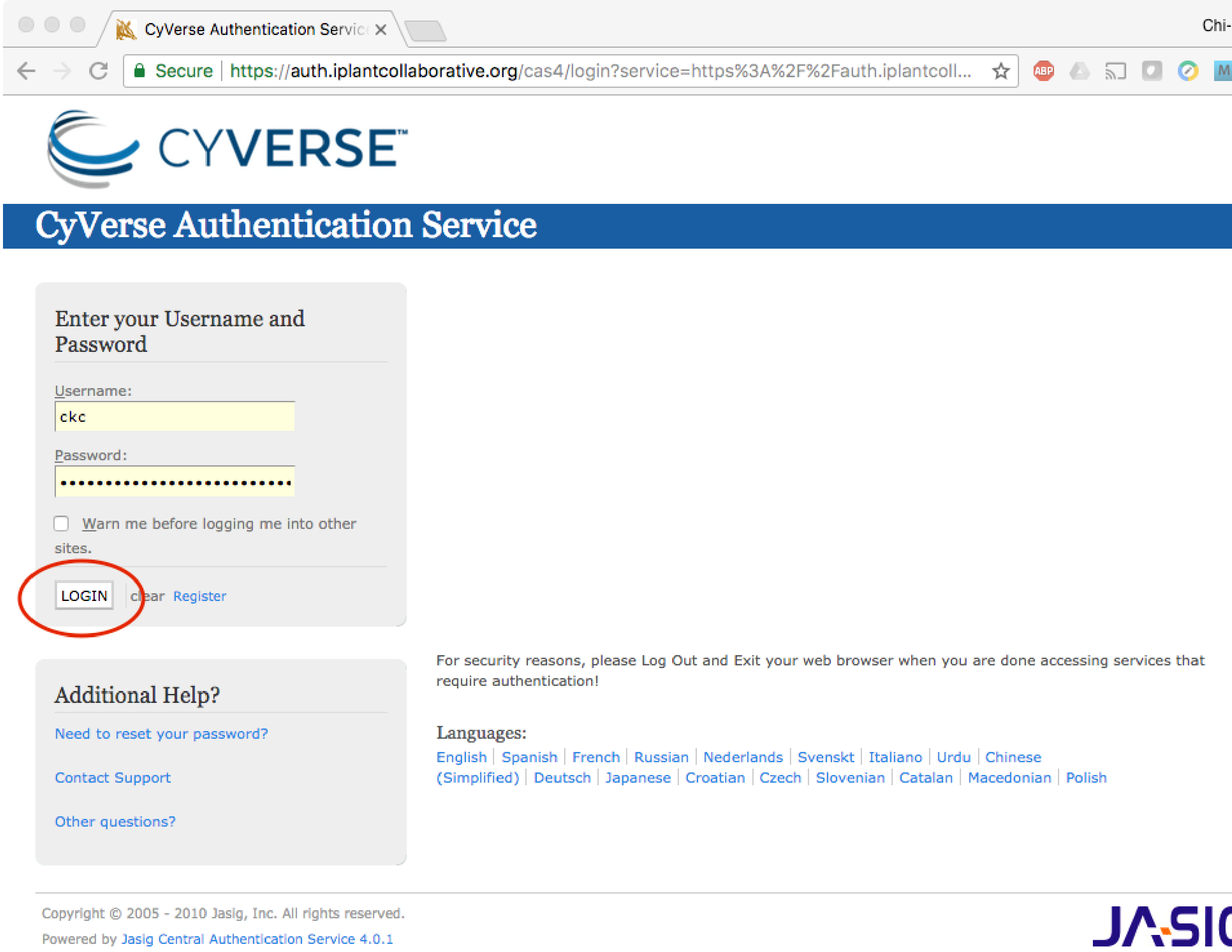
Below, we've provided screenshots of the whole process. You can click on them to zoom in. The important areas to fill in are highlighted.

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

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

1. Fill in the username and password and click "LOGIN"

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



CyVerse Authentication Servic
Chi-
Secure
https://auth.iplantcollaborative.org/cas4/login?service=https%3A%2F%2Fauth.iplantcoll...
CYVERSE™

CyVerse Authentication Service

Enter your Username and Password

Username: ckc

Password: ••••••••••••••••••••••••••••

□ Warn me before logging me into other sites.

Additional Help?

Need to reset your password?

Contact Support

Other questions?

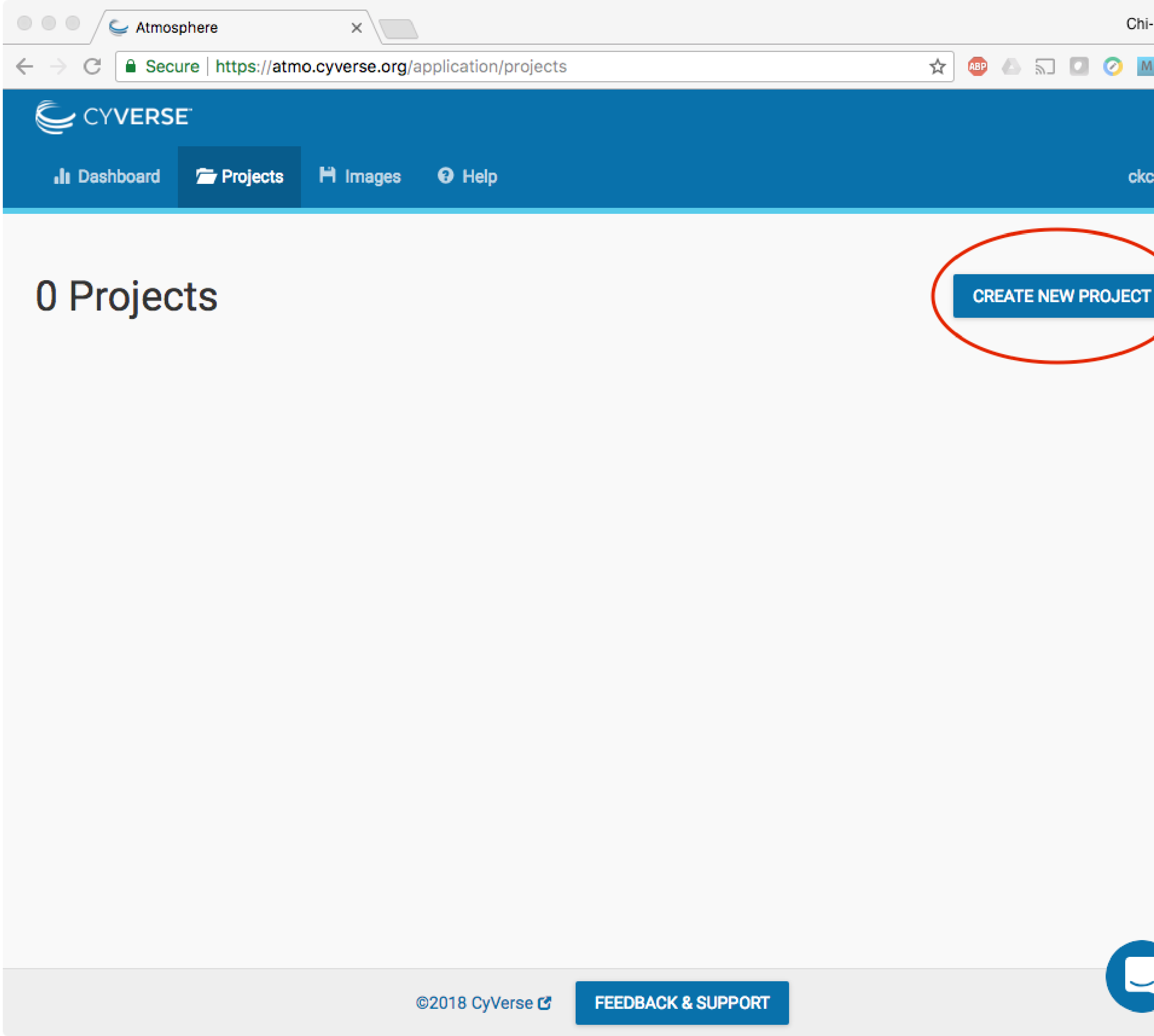
For security reasons, please Log Out and Exit your web browser when you are done accessing services that require authentication!

Languages:
English | Spanish | French | Russian | Nederlands | Svenskt | Italiano | Urdu | Chinese (Simplified) | Deutsch | Japanese | Croatian | Czech | Slovenian | Catalan | Macedonian | Polish

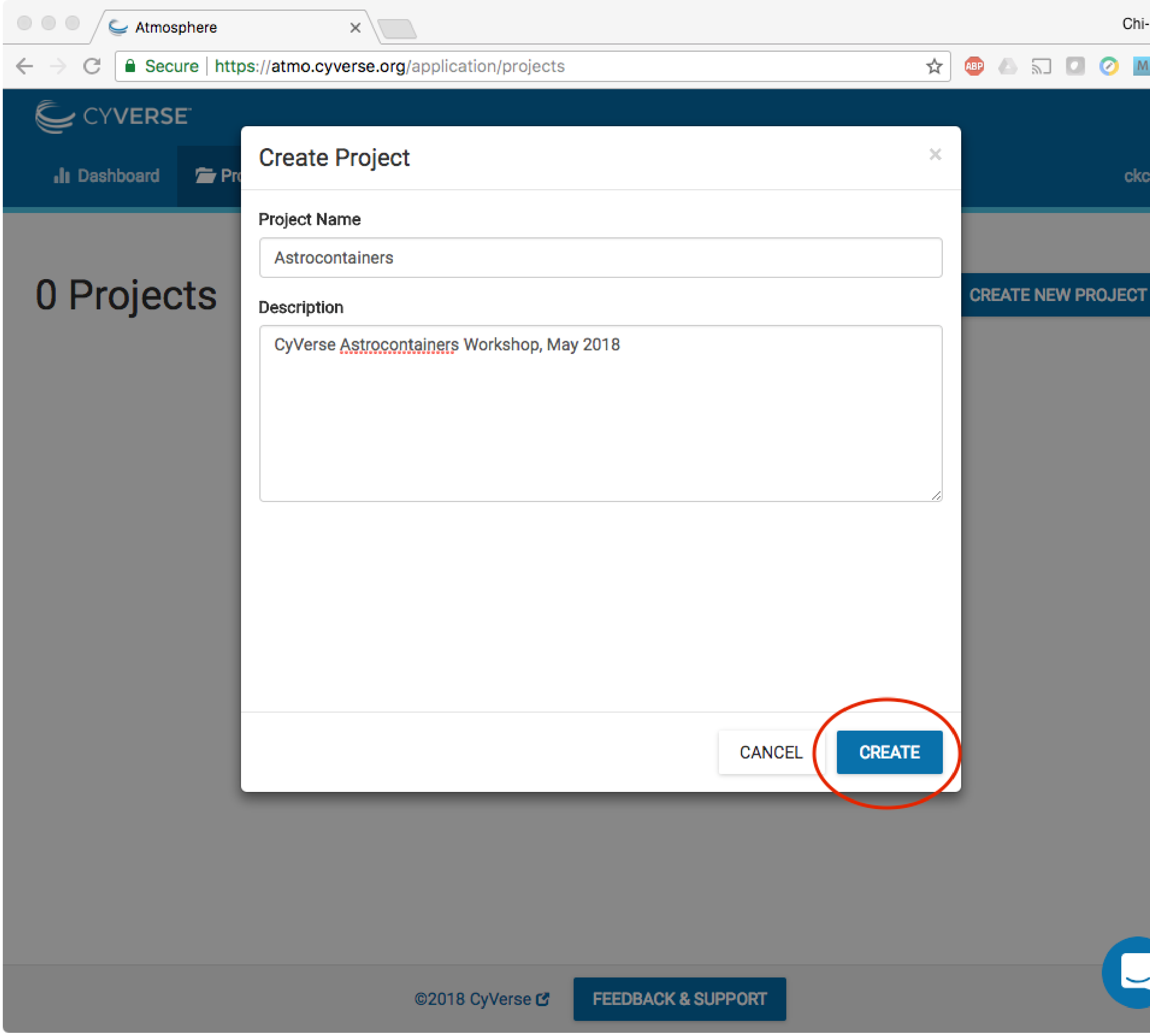
Copyright © 2005 - 2010 Jasig, Inc. All rights reserved.
Powered by Jasig Central Authentication Service 4.0.1

2. Select the "Projects" tab and then click the "CREATE NEW PROJECT" button

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



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



Create Project

Project Name

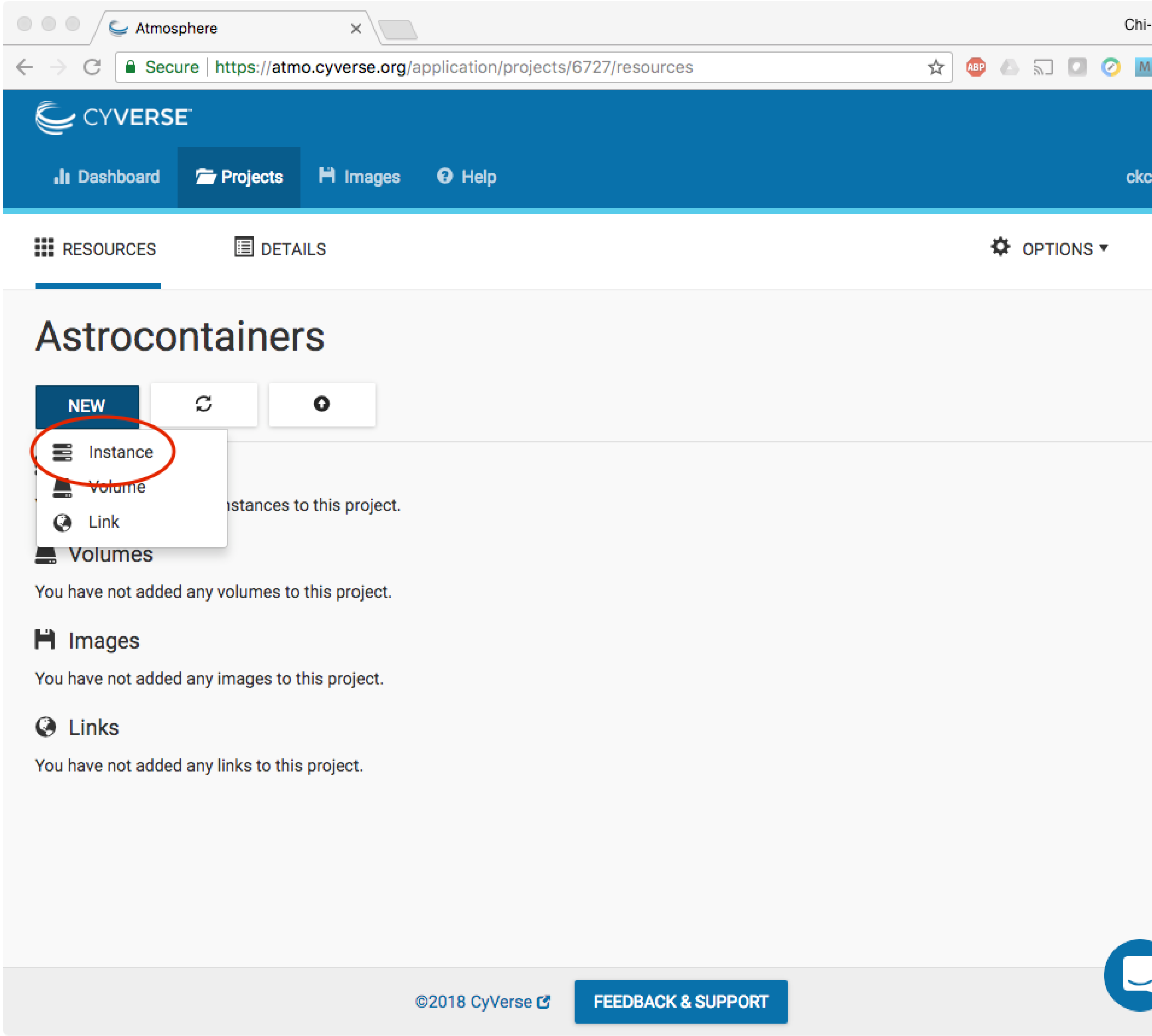
Astrocontainers

Description

CyVerse Astrocontainers Workshop, May 2018

3. Select the newly created project

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



Astrocontainers

…nstances to this project.

Volumes

You have not added any volumes to this project.

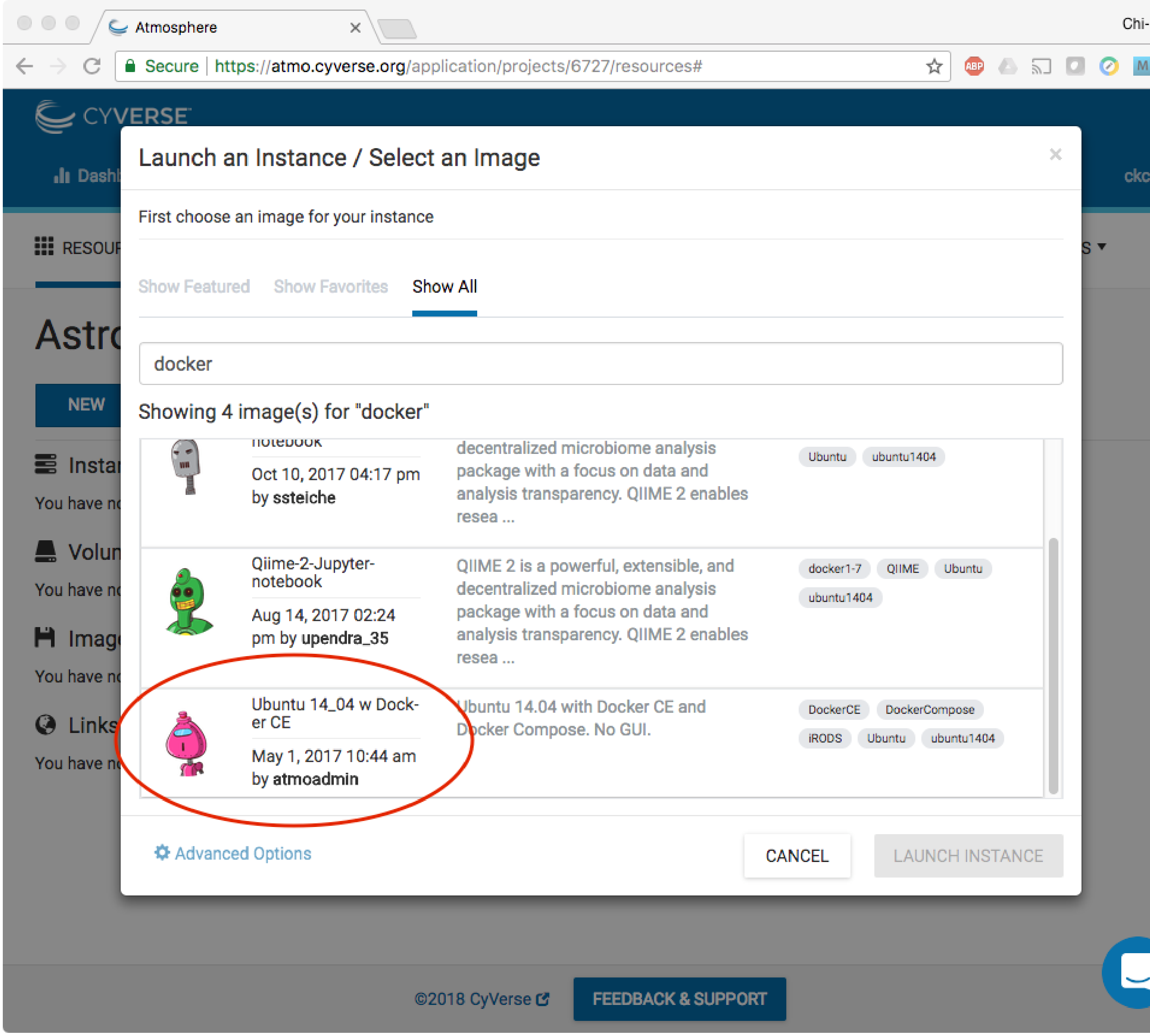
Images

You have not added any images to this project.

Links

You have not added any links to this project.

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



Launch an Instance / Select an Image

First choose an image for your instance

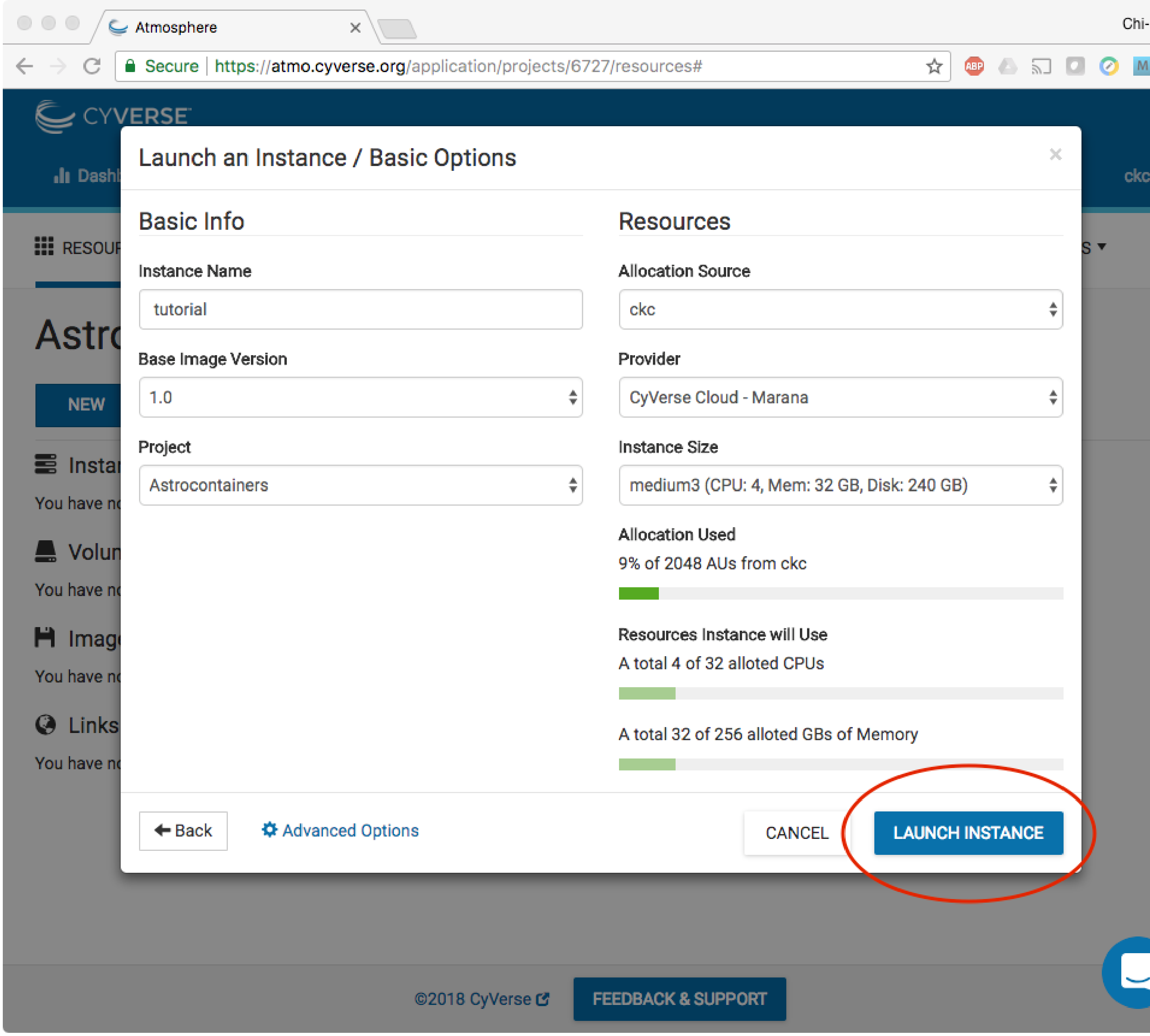
Show Featured | Show Favorites | Show All

docker

Showing 4 image(s) for "docker"

	notebook Oct 10, 2017 04:17 pm by ssteiche	decentralized microbiome analysis package with a focus on data and analysis transparency. QIIME 2 enables resea ...	Ubuntu ubuntu1404
	Qiime-2-Jupyter-notebook Aug 14, 2017 02:24 pm by upendra_35	QIIME 2 is a powerful, extensible, and decentralized microbiome analysis package with a focus on data and analysis transparency. QIIME 2 enables resea ...	docker1-7 QIIME Ubuntu ubuntu1404
	Ubuntu 14_04 w Docker CE May 1, 2017 10:44 am by atmoadmin	Ubuntu 14.04 with Docker CE and Docker Compose. No GUI.	DockerCE DockerCompose iRODS Ubuntu ubuntu1404

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



Launch an Instance / Basic Options

Basic Info

Instance Name: tutorial

Base Image Version: 1.0

Project: Astrocontainers

Resources

Allocation Source: ckc

Provider: CyVerse Cloud - Marana

Instance Size: medium3 (CPU: 4, Mem: 32 GB, Disk: 240 GB)

Allocation Used

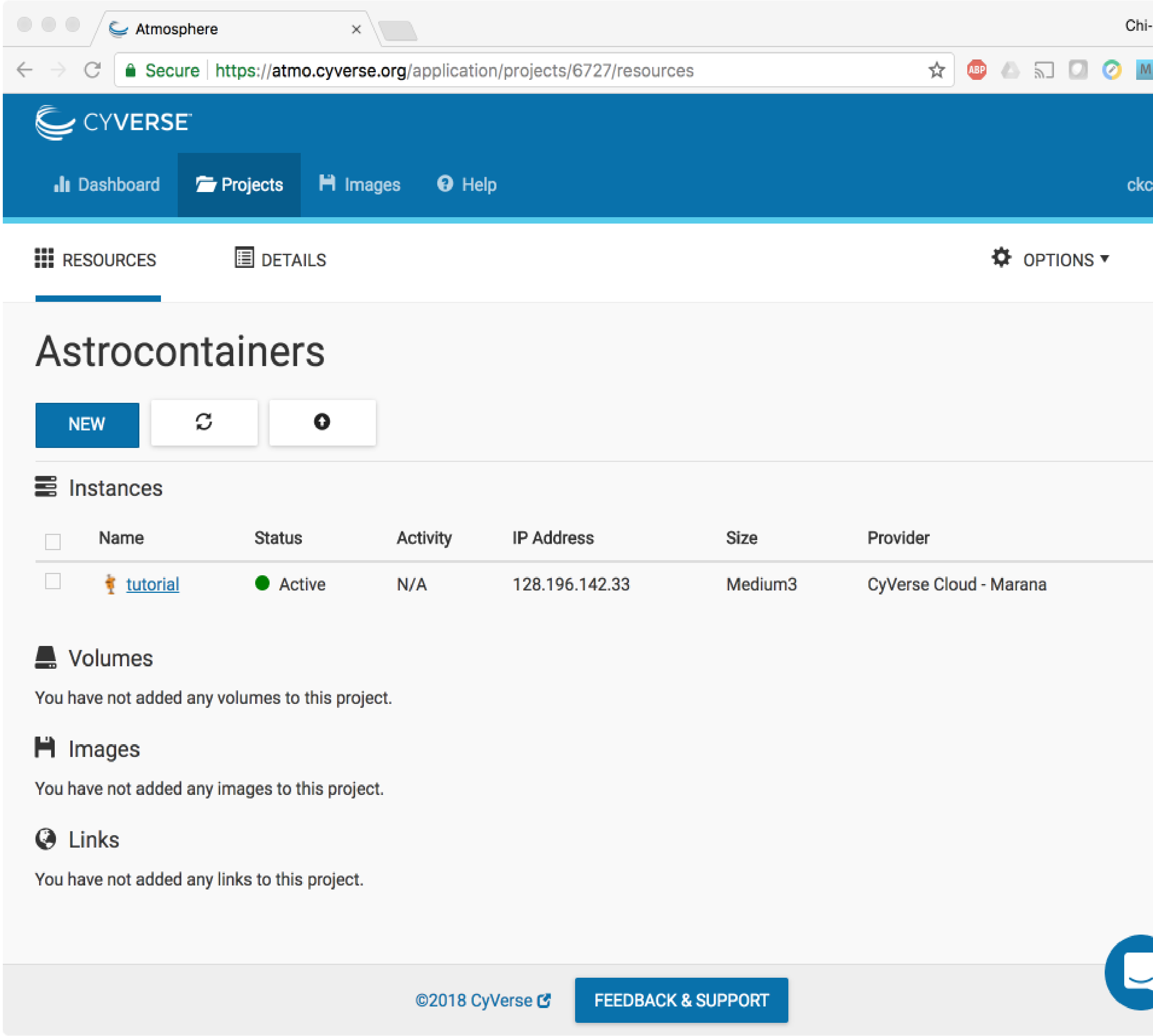
9% of 2048 AUs from ckc

Resources Instance will Use

A total 4 of 32 alloted CPUs

A total 32 of 256 alloted GBs of Memory

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



Astrocontainers

Instances

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

Volumes

You have not added any volumes to this project.

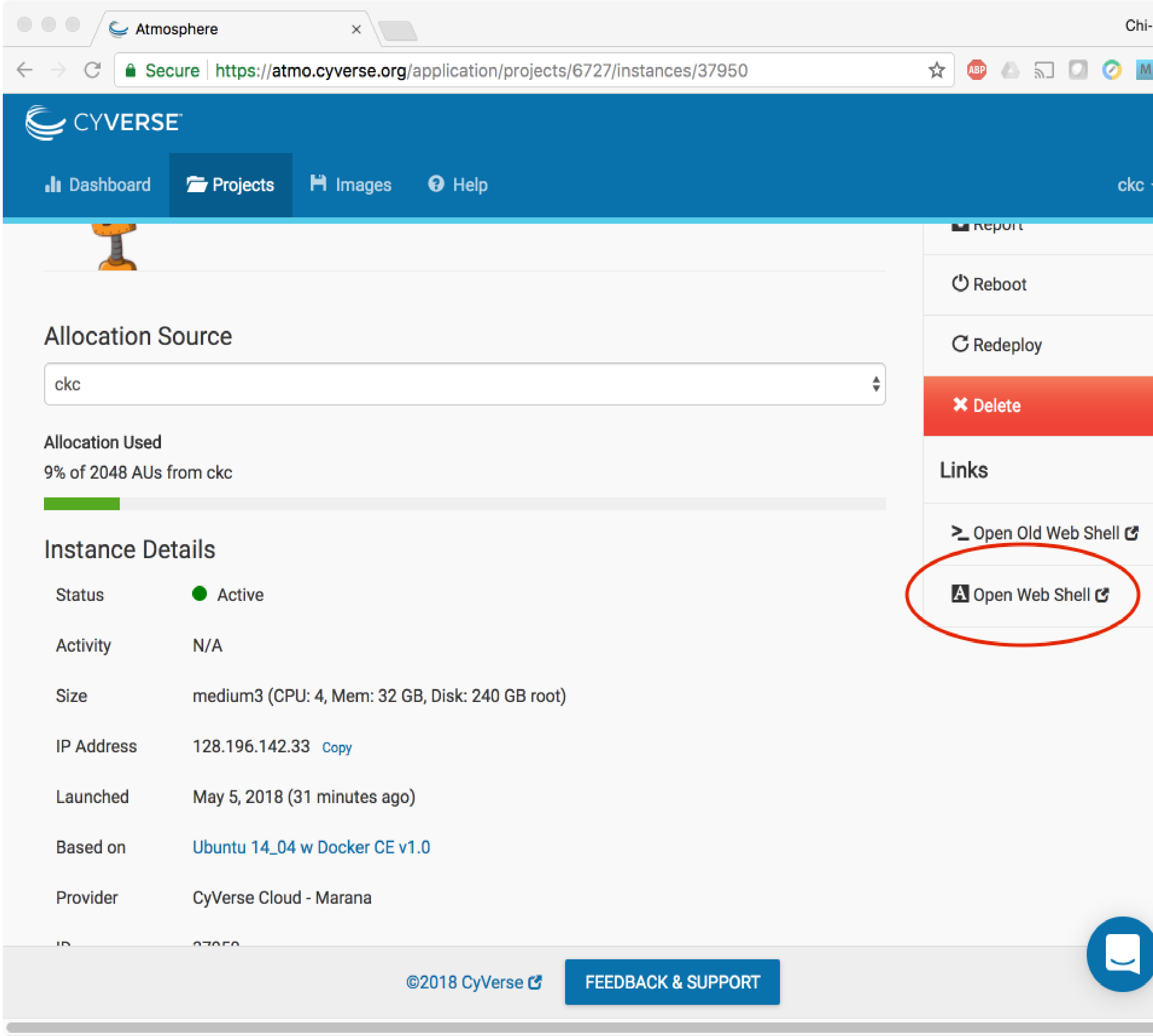
Images

You have not added any images to this project.

Links

You have not added any links to this project.

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



Allocation Source

ckc

Allocation Used

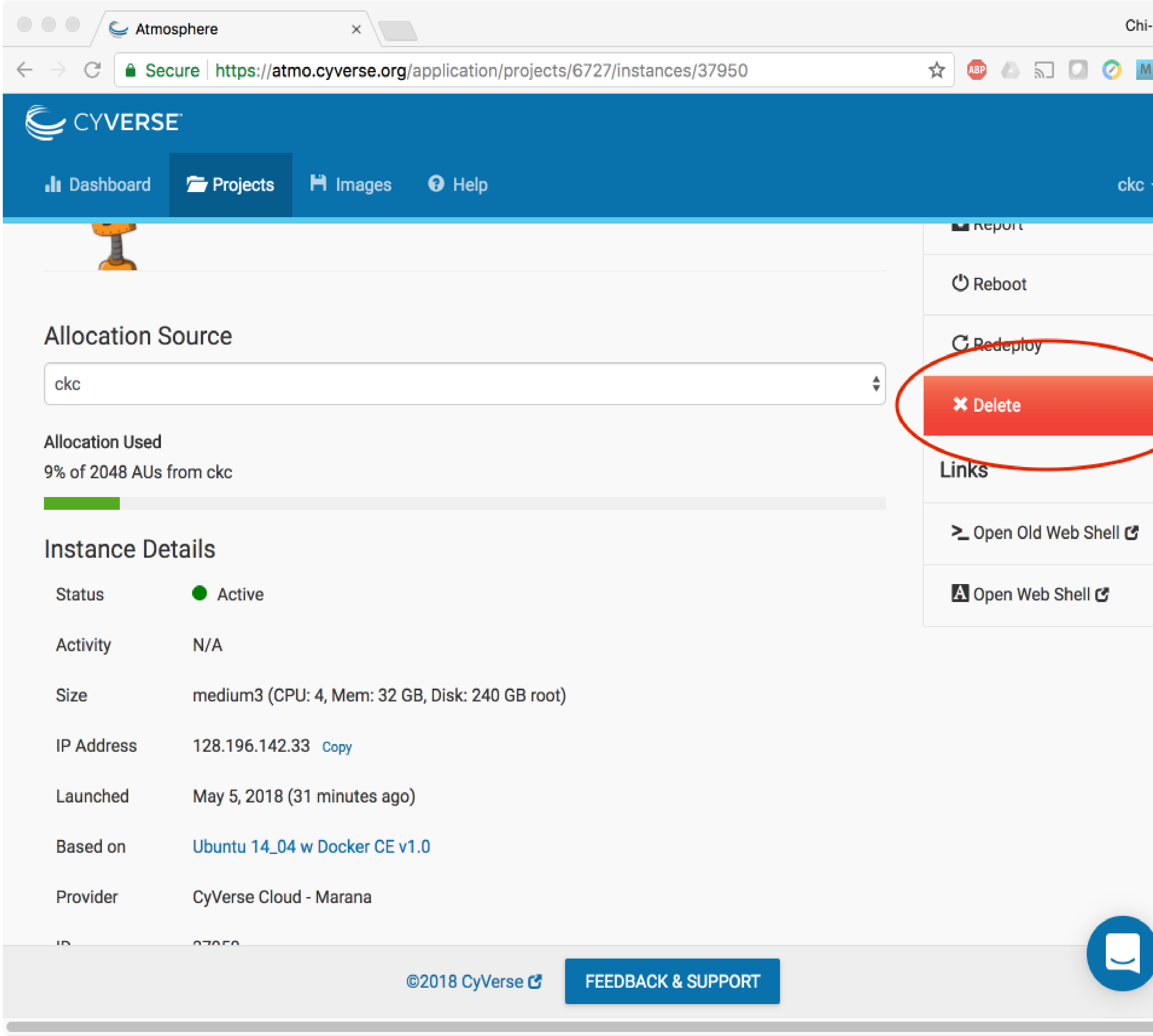
9% of 2048 AUs from ckc

Instance Details

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

4. **Deleting your instance**

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



Allocation Source

ckc

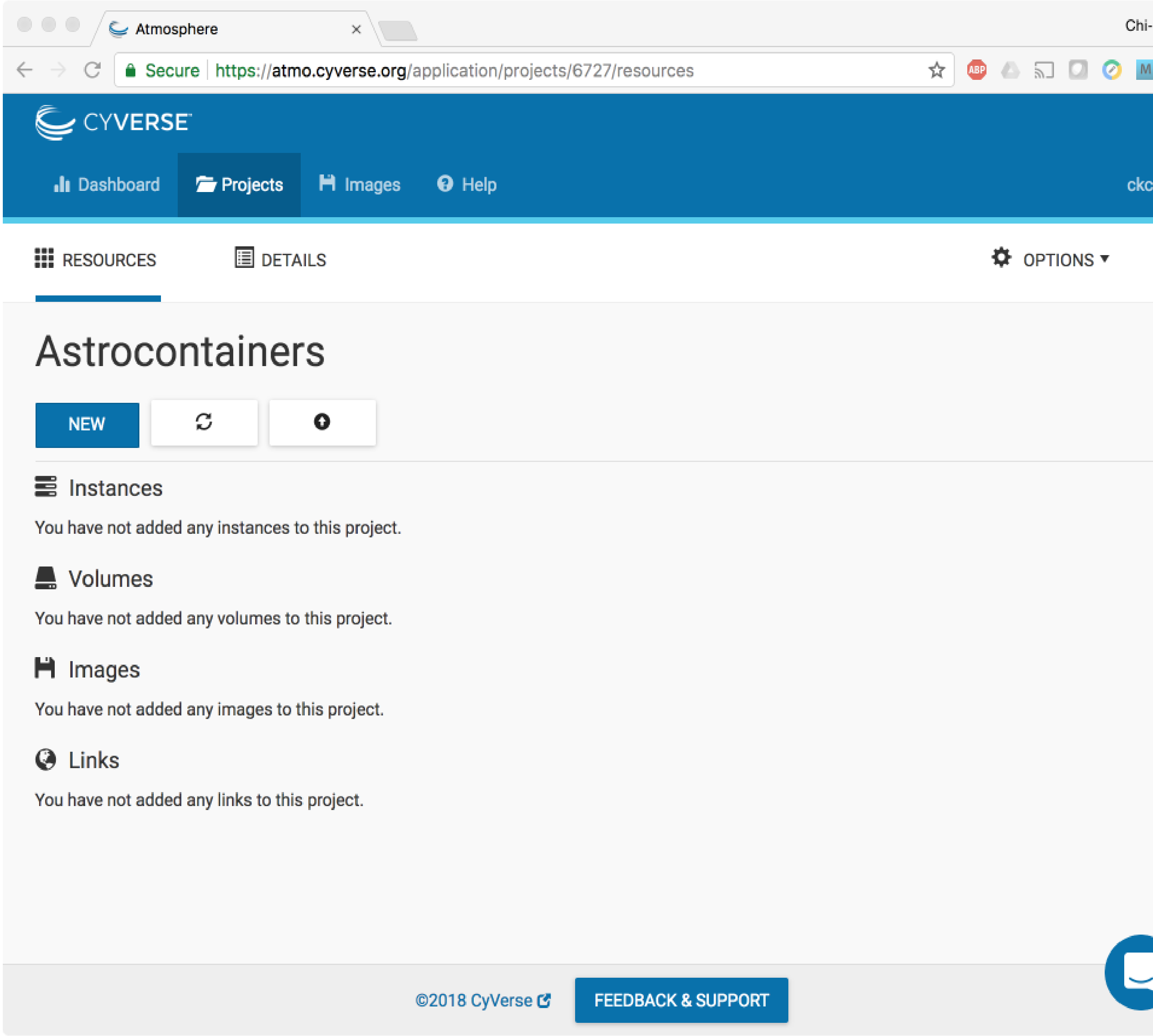
Allocation Used

9% of 2048 AUs from ckc

Instance Details

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

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



Astrocontainers

Instances

You have not added any instances to this project.

Volumes

You have not added any volumes to this project.

Images

You have not added any images to this project.

Links

You have not added any links to this project.

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 11

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 12

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*

12.1 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 13

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