
CyVerse Documentation

Release 0.3.0

CyVerse

Sep 18, 2020

Getting Started

1	Learning objectives	3
2	FAIR Data principles	5
3	Code of Conduct	7
4	Pre-Workshop Setup	9
5	Location	11
6	Agenda	13
7	About CyVerse	17
8	Introduction to Reproducible Science	19
9	Basics of Linux	21
10	Training session in Docker	23
11	Training session in Singularity	25
12	Breakout sessions	27
13	Finding the perfect container	29
14	Introduction to Docker	37
15	Advanced Docker	51
16	Introduction to Singularity	73
17	Advanced Singularity	83
18	Setting up Singularity file system	87
19	Singularity and High Performance Computing	89
20	BioContainers	95

21 Containerized Workflows	107
22 SETUP	109
23 Why Snakemake	111
24 Other Workflow Managers	113
25 Docker for Data Science	115
26 Booting a CyVerse Atmosphere instance	121
27 Tool integration in the Discovery Environment (DE)	129
28 Deploying apps in CyVerse Discovery Environment	135
29 Deploying interactive apps in CyVerse Discovery Environment	145
30 Docker related resources	155
31 Singularity related resources	157
32 Other resources	159
33 For instructors!	161
34 Problems? Bugs? Questions?	165



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

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

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



CHAPTER 1

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.

FAIR Data principles

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

Container Camp Goes Green

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

Who should attend?

Faculty, researchers, postdocs, and graduate students who use and analyze data of all types (genomics, astronomy, image data, Big Data, etc.).

Workshop level

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

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

Need help?

Couldn't find what you were looking for?

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



 [Learning Center Home](#)

CHAPTER 3

Code of Conduct

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

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

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

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

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

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

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

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

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

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



 [Learning Center Home](#)

CHAPTER 4

Pre-Workshop Setup

Please complete the minimum Setup Instructions to prepare for Container Camp at CyVerse, The University of Arizona, which will run from March 10-13, 2020.

Prerequisite	Notes	Additional notes
Wi-Fi-enabled laptop	You should be able to use any laptop (Windows/MacOS/Linux.). We strongly recommend Firefox or Chrome browser. It is recommended that you have administrative/install permissions on your laptop.	<ul style="list-style-type: none">• Download FireFox• Download Chrome
CyVerse Account	Please ensure that you have a CyVerse account and have verified your account by completing the verification steps in the email you got when you registered.	Register for your cyverse account at https://user.cyverse.org/ .
Github Account	Please ensure that you have a Github account if you don't have one already	Register for your Github account at https://github.com/ .
Dockerhub Account	Please ensure that you have a Dockerhub account if you don't have one already	Register for your Dockerhub account at https://hub.docker.com/ .
Text Editor	Please ensure that you have a Text editor of your choice. Any decent text editor would be sufficient and recommended ones include Atom, Sublime, & VSCode	Download Sublime at https://www.sublimetext.com/ . Download Atom at https://atom.io/ . Download VS-Code at https://code.visualstudio.com/
Slack for networking	We will be using Slack extensively for communication and networking purposes	Register for Slack at https://slack.com/ .

Optional Downloads

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

Tool	Notes	Link
SSH Clients (Windows)	PuTTY allows SSH connection to a remote machine, and is designed for Windows users who do not have a Mac/Linux terminal. MobaXterm is a single Windows application that provides a ton of functions for programmers, webmasters, IT administrators, and anybody is looking to manage system remotely	<ul style="list-style-type: none">• Download PuTTY• Download mobaXterm• Update Windows 10 & install Windows Subsystem for Linux v2 (WSL2)
Cyberduck	Cyberduck is a third-party tool for uploading/downloading data to CyVerse Data Store. Currently, this tool is available for Windows/MacOS only. You will need to download Cyberduck and the connection profile. We will go through configuration and installation at the workshop.	<ul style="list-style-type: none">• Download Cyberduck• Download CyVerse Cyberduck connection profile
iRODS iCommands	iCommands are command-line software to connect to the CyVerse Data Store.	Download and installation instructions available at CyVerse Learning Center



Location

CyVerse Container Camp will be held in Room A116 of the [Roy P. Drachman Hall](#), located at 1295 N Martin Ave, Tucson, AZ 85719

SERVICES

Drachman is adjacent the Banner University Medical Center. In the event of a medical emergency, attendees may be transported to Banner, or to the nearest [urgent care](#) facility.

PARKING

Nearest public parking is the Highland Garage, about 3 blocks west of Drachman (\$1/hr with \$8/day max) or you can take the Purple or Green CatTran shuttle to the northern terminus (the AHSL Library stop).

Drachman is an approximate 7 minute walk from The Aloft Hotel.

UArizona Campus Map: <https://map.arizona.edu/>

CatTran Route Map: <https://parking.arizona.edu/cattran/cat-tran-routes/>

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-





CHAPTER 6

Agenda

Below are the schedule and classroom materials for Container Camp at The University of Arizona, which will run from March 10th to 13th, 2020.

Day	Time	Topic/Activity	Objectives
03/10/20 (Tuesday)	11:00-12:00	Laptop check and pre-installation checklist	Final check to make sure you're ready
	11:30-12:00	Instructor briefing	
	12:00-12:15	Welcome & Logistics (Tyson Swetnam)	Cover Expectations for CC
	12:15-1:00	General overview of container technology landscape (Nirav Merchant)	
	1:00-2:10	What is a container? (Tyson Swetnam)	Basics & why you might use a container image for research
	2:10-2:30	Break	time to talk and network
	2:30-3:00	Searching Image Registries (Tyson Swetnam)	Finding the right image, downloading (pulling)
	3:00-5:00	Running a container (Tyson Swetnam)	Start a container, add a volume, opening ports, monitor, clean up
	5:00-5:30	Debriefing with instructors	

Day	Time	Topic/Activity	Notes/Links
03/11/20 (Wednesday)	8:00-8:30	Instructor Briefing	
	8:30-8:45	Review Day 1	time for questions, comments, suggestions
	8:45-9:45	(Mats Rynge)	Containers used at scale
	9:45-10:10	Make your own container: (Tyson Swetnam)	Customizing base images, setting up Docker-Compose
	10:10-10:30	Break	time to talk and network
	10:30-11:15	Continuous Integration with GitHub (TBA)	Building your images with CI/CD for automation and scaling
	11:15-12:00	Bringing your container image to CyVerse (Amanda Cooksey)	Tool integration in the DE (interactive, executable, & OpenScienceGrid)
	12:00-1:00	Lunch Break (on your own)	
	1:00-5:00	Breakout sessions	Breakout sessions
	5:00-5:30	Debriefing with instructors	

Day	Time	Topic/Activity	Notes/Links
03/12/20 (Thursday)	8:30-8:45	Review Day 2	time for questions, comments, suggestions
	8:45-10:10	Introduction to Singularity (Tyson Swetnam)	Using Docker on HPC
	10:10-10:30	Break	time to talk and network
	10:30-12:00	Singularity and High Performance Computing (John Fonner)	Singularity for MPI and GPU workloads
	12:00-1:00	Lunch Break (on your own)	
	1:00-1:15	Project pitches (2 min) and BYOD/BYOA	
	1:15-3:10	Project Time	Bring Your Own Data (BYOD) & Bring your Own Analyses (BYOA)
	3:10-3:30	Break	time to talk and network
	3:30-5:0	Project Time	BYOD & BYOA
	5:00-5:30	Debriefing with instructors	

Day	Time	Topic/Activity	Notes/Links
03/13/20 (Friday)	8:30-8:45	Review Day 3	time for questions, comments, suggestions
	8:45-9:45	Finalize Projects	BYOD & BYOA
	9:45-10:10	Project Presentations	
	10:10-10:30	Break	time to talk and network
	10:30-11:30	Presentations	
	11:30-12:00	Course Evaluations	
	12:00	Dismissal	
	12:00-1:00	Instructor Post Mortem	



About CyVerse

CyVerse Vision: Transforming science through data-driven discovery.

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

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

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

Funding and Citations

CyVerse is funded entirely by the National Science Foundation under Award Numbers DBI-0735191, DBI-1265383 and DBI-1743442.

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



Introduction to Reproducible Science

by Jason Williams

The so-called reproducibility crisis (see , ,) is something you have probably heard about (and maybe one of the reasons you have come to Container Camp). Headlines in the media (such as) definitely give pause to researchers and ordinary citizens who hope that the science used to recommend a course of medical treatment or design self-driving cars is sound.

Before we go further, it's actually important to ask what is reproducibility?

Question

How do you define reproducible science?

Answer

In , Hans Plesser gives the following useful definitions:

- **Repeatability** (Same team, same experimental setup): The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation.
- **Replicability** (Different team, same experimental setup): The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.
- **Reproducibility** (Different team, different experimental setup): The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

The paper goes on to further simplify:

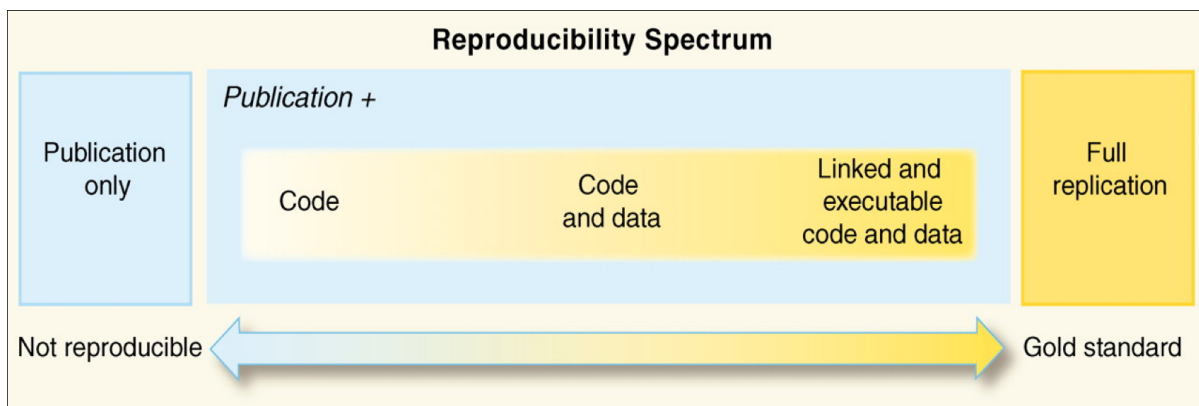
- **Methods reproducibility:** provide sufficient detail about procedures and data so that the same procedures could be exactly repeated.
 - **Results reproducibility:** obtain the same results from an independent study with procedures as closely matched to the original study as possible.
 - **Inferential reproducibility:** draw the same conclusions from either an independent replication of a study or a reanalysis of the original study.
-

Discussion Question

How do these definitions apply to your research/teaching?

Work with your fellow learners to develop a shortlist of ways reproducibility relates to your work. Try to identify challenges and even successes you'd like to share.

Often, when we say “reproducibility” we mean all or at least several of the concepts the proceeding discussion encompasses. Really, reproducibility can be thought of as set values such as some laboratories express in a code of conduct, (see for example or). Reproducibility comes from our obligations and desires to work ethically, honestly, and with confidence that the data and knowledge we produce is done has integrity. Reproducibility is also a “spectrum of practices”, not a single step. (See figure below from).



Assuming you have taken in the potentially anxiety inducing information above, the most important thing to know is that there is a lot of help to make reproducibility a foundation of all of your research.

Fix or improve this documentation:

- On Github: [|Github Repo Link|](#)
 - Send feedback: Tutorials@CyVerse.org
-

Basics of Linux

Modern web, cloud, high performance computing, and most data science applications are run on operating systems (OS) other than Microsoft Windows. To do data intensive science, you need a familiarity with [Linux](#). We've scheduled several sections during Container Camp for working on Linux Systems using [CyVerse' Atmosphere Cloud](#), which runs Linux OS virtual machines.

The good news comes in two parts. First, whether you know it or not, you **probably** already use Linux or a platform based on Linux, on a daily basis. *Do you have an Android or iOS phone?* If you own a [Mac OS X](#) device, you already enjoy many of the benefits of a Linux-like OS, including access to a terminal. Second, the Linux experience has generally been described as [satisfying](#), and many users report moving on from Windows OS to Linux comes [without regret](#).

Over [87%](#) of the personal computer market still relies on the popular Microsoft OS. However, the landscape changes completely for mobile apps (99% Linux or Linux-like [Android, iOS], <0.1% Windows), web (66% Linux, 32% Windows), and cloud or HPC (100% Linux). Microsoft is acutely aware of this disparity, and is actively working to integrate Linux into their OS, including their [acquisition of GitHub](#) (and [how it has changed](#)), and the release of Windows Subsystem for Linux (WSL) 2.

9.1 Common Linux Operating Systems

The most common operating systems you'll see used for data science are:

- [Alpine](#) - small and lightweight, useful in container applications
- [CentOS](#) - stable, reliable, most commonly used on web and cloud servers
- [Debian](#) - lightweight, utilitarian, stable
- [Ubuntu](#) - utilitarian, user friendly, most popular distribution, based on Debian

Enterprise Distributions:

- [Red Hat](#) - based on open source software, you pay for customer support

9.2 *Installing Linux*

9.2.1 *Desktop-based Distributions*

- [Ubuntu](#)
- [Debian](#)
- [Mint](#) - “modern, elegant and comfortable operating system which is both powerful and easy to use.”
- [OpenSUSE](#) - “The makers’ choice for sysadmins, developers and desktop users.”

9.2.2 *Windows Subsystem for Linux*

The so-called “WSL” is a complete linux subsystem that runs under Windows 10. Microsoft recently announced [WSL 2.0](#).

9.2.3 *Windows Linux Dual boot*

Not ready to take the Linux plunge yet? Why not set up a Windows-Linux dual boot?

- [Ubuntu](#)
- [Mint](#)

9.2.4 *Package Managers*

Linux uses [package management](#) services to install programs. If you’re a R user, this should seem familiar. Packages can be installed on the command line, or in graphic UI.

9.3 *Self Paced*

[Best Linux Distributions for Beginners](#)

[Beginners Guide to Linux](#)

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org

Training session in Docker

In these sessions we will cover various aspects of Docker containers for data science applications. Starting with the basics of pulling images from Docker Registries, running Docker containers locally and on cloud, and managing your data in a container using volumes. Topics include Docker installation, pulling and running pre-built Docker containers, and deploying browser-based applications (like Jupyter and RStudio) with Docker.

- [Docker Introduction](#)

In the advanced session, you will modify an existing container by installing your own science libraries or packages. Topics include pulling Docker containers from public and private registries, automated Docker image building from GitHub repositories, managing data in Docker containers, Docker Compose for building multiple Docker containers, and improving your data science workflows using Docker containers.

- [Advanced Docker](#)

Training session in Singularity

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

- [Singularity Introduction](#)

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

- [Advanced Singularity](#)

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

Breakout sessions

1. **Data Science IDEs** Complexity: beginner - Lead: Tyson Swetnam, CyVerse

[Docker for Data Science breakout session content](#)

In this breakout session, you'll learn the basics about deploying popular IDE (RStudio and Jupyter) and ML containers from the NVIDIA GPU Cloud. We will discuss the complexities of working with these different container types, with hands on examples running CyVerse and HPC with Singularity.

2. **Biocontainers** Complexity: moderate - Lead: Amanda Cooksey, CyVerse Scientific Analyst

[Biocontainer breakout session content](#)

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

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

[Containerized workflows breakout session content](#)

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





Finding the perfect container

Chances are a Docker *image* already exists for the application you use in your research. Rather than starting from scratch and creating your own *image*, you need to know where to look for existing images.

Important: But wait, what are the differences in a *container* and an *image*? An important distinction must be made with regard to *base images* and *child images*, *official images* and *user images*

container - Running instance of an *image* — the *container* runs the actual processes. A container includes an application and all of its dependencies. It shares its kernel with other containers, and runs as an isolated process in the space on the host OS.

layer - an intermediate image, the result of a single set of build commands. A Docker image is built from layers.

image - The file system and configuration of an application which is used to create the container.

tag - identifies exact version of the image. If a tag is not given, by default the `:latest` tag will be used.

base image - have no parent image, usually images with an OS like ubuntu, alpine or debian.

child image - build on base images, added layers with additional functionality.

official image - 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 Docker Hub the `python`, `node`, `alpine`, and `nginx` images are official (base) images. To find out more about them, check out the [Official Docker Images Documentation](#).

publisher image: - Certified images that also include support and guarantee compatibility with Docker Enterprise.

user image - 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.

Dockerfile - is a text file that contains a list of commands that the Docker daemon calls while creating an image. The Dockerfile contains all the information that Docker needs to know to run the app — a base Docker image to run from, location of your project code, any dependencies it has, and what commands to run at start-up. It is a simple way to automate the image creation process. The best part is that the commands you write in a Dockerfile are almost

identical to their equivalent Linux commands. This means you don't really have to learn new syntax to create your own Dockerfiles.

13.1 Docker Registries

Docker uses the concept of “*Registries*”

Question

What *EXACTLY* is a **Registry**?

Answer

a storage and distribution system for named Docker images

Organized by owners into “repositories” with compiled “*images*” that users can download and run

Things you can do with Docker registries:

- Search for public images
- Pull images
- Share private images
- Push images
- You must have an account on a registry to create repositories and images.
- You can create many repositories.
- You can create many tagged images in a repository
- You can even set up your own private registry using a *Docker Trusted Registry*

13.2 Search image registries

Warning: Only use images from trusted sources or images for which you can see the `Dockerfile`. Any image from an untrusted source could contain something other than what's indicated. If you can see the Dockerfile you can see exactly what is in the image.

The Docker command line interface uses the [Docker Hub](#) public registry by default.

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

- [Docker Hub](#)
- [Docker Trusted Registry](#)
- [Amazon Elastic Container Registry](#)
- [Google Container Registry](#)
- [Azure Container Registry](#)

- NVIDIA GPU Cloud
- Private Docker Registry - not official Docker
- Gitlab Container Registry
- Quay
- TreeScale
- Canister
- BioContainers Registry

13.2.1

Docker Hub is a service provided by Docker for finding and sharing container images with your team. Docker Hub is the most well-known and popular image registry for Docker containers.

Important: **Registry** a storage and distribution system for named Docker images

Repository collection of “images” with individual “tags”.

Teams & Organizations: Manages access to private repositories.

Builds: Automatically build container images from GitHub or Bitbucket on the Docker Hub.

Webhooks: Trigger actions after a successful push to a repository to integrate Docker Hub with other services.



13.2.2

BioContainers is a community-driven project that provides the infrastructure and basic guidelines to create, manage and distribute bioinformatics containers with **special focus in proteomics, genomics, transcriptomics and metabolomics**. BioContainers is based on the popular frameworks of Docker.

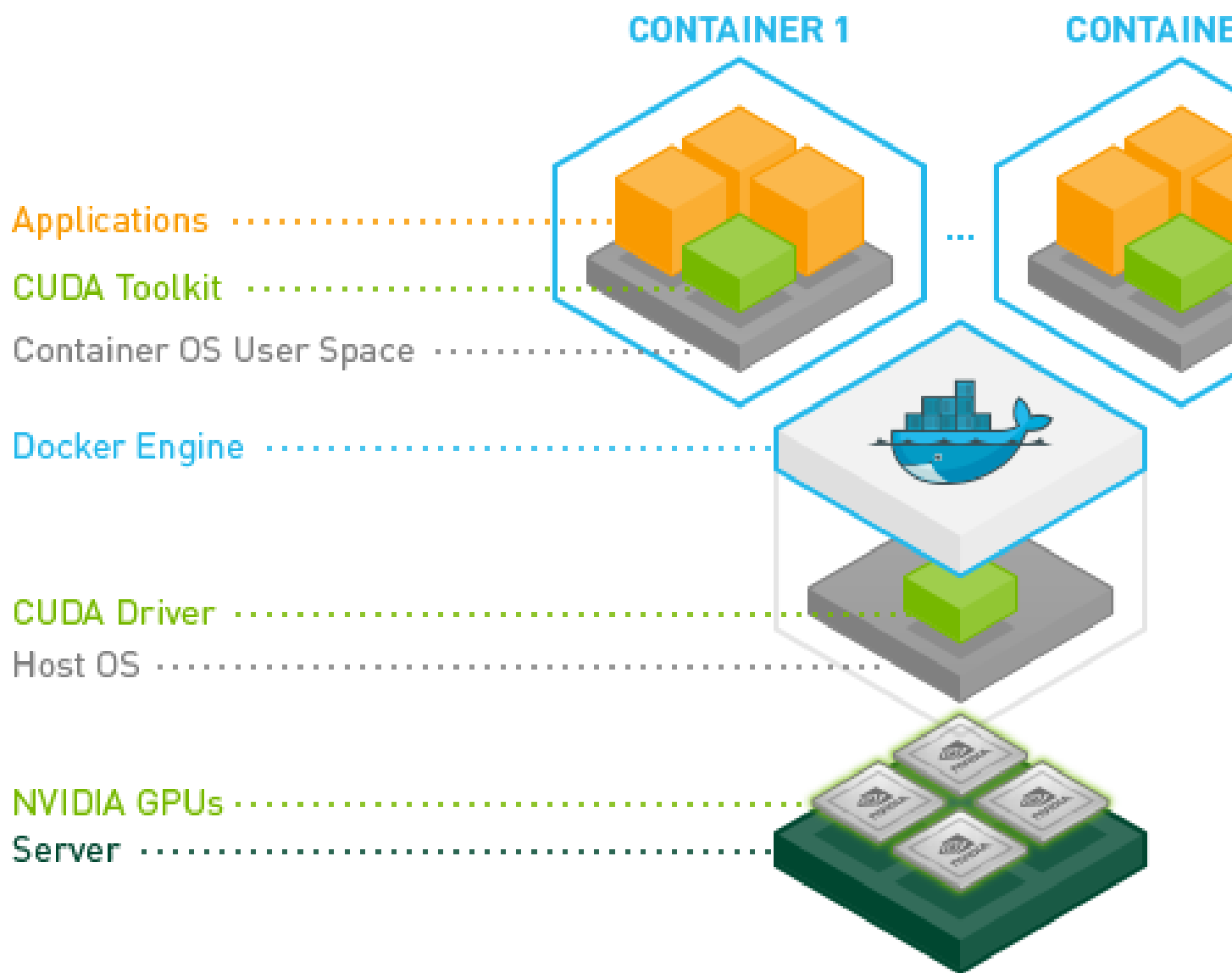
Although anyone can create a BioContainer, the majority of BioContainers are created by the Bioconda project. Every Bioconda package has a corresponding BioContainer available at Quay.io.

13.2.3

Quay is another general image registry. It works the same way as Docker Hub. However, Quay is home to all BioContainers made by the Bioconda project. Now we will find a BioContainer image at Quay, pull that image and run it on cloud virtual machine.

13.2.4 NVIDIA GPU Cloud

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



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

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

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

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

NVIDIA GPU Cloud hosts three [registry spaces](#)

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

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

Programs which leverage 3D applications include [VirtualGL](#), [TurboVNC](#), & [TigerVNC](#).

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

13.2.5 Pull an image from a registry

To run your container you will need a computer with Docker installed. We will use an Atmosphere cloud instance today but this can be done on any computer.

Open an Atmosphere instance

1. Go to [Atmosphere](#) and log in with your CyVerse credentials.
2. Click on 'projects' tab at the top of the page.
3. You should have a project called 'Conatainer Camp 2020'; click on that tile.
4. You should already have a running instance called **Ubuntu 18_04 GUI XFCE Base**. To confirm this look for a green dot and the word 'Active' under 'status'.

The screenshot shows the CyVerse Atmosphere web interface. At the top is a navigation bar with 'Dashboard', 'Projects', 'Images', and 'Help'. The 'Projects' tab is selected. Below the navigation bar, there are tabs for 'RESOURCES' and 'DETAILS', with 'DETAILS' being active. The main content area shows the project 'CC2020' with buttons for 'NEW', a refresh icon, and an info icon. Below this is a section titled 'Instances' with a table listing the instances. The table has columns for Name, Status, Activity, IP Address, Size, and Provider. One instance is listed: 'Ubuntu 18_04 GUI XFCE Base' with a green status dot, 'Active' status, 'N/A' activity, IP address '128.196.142.89', size 'Tiny1', and provider 'CyVerse Cloud - Marana'.

Name	Status	Activity	IP Address	Size	Provider
Ubuntu 18_04 GUI XFCE Base	Active	N/A	128.196.142.89	Tiny1	CyVerse Cloud - Marana

5. Copy the IP address for your instance
6. Open a terminal on your computer
7. Connect to your Atmosphere instance via ssh **using the IP address you copied**

```
$ ssh 128.196.142.89
```

8. You will be asked if you are sure you want to continue—say **yes**.

```
amcooksey@ubuntuai:~$ ssh 128.196.142.89
The authenticity of host '128.196.142.89 (128.196.142.89)' can't be established.
ECDSA key fingerprint is SHA256:8U/jm8DsF1feHFauqfKLRLQiRHJLZ+UwPyH+MbnxDLI.
Are you sure you want to continue connecting (yes/no)? yes
```

9. If you see something like this (below) then you have successfully logged into your Atmosphere instance.

[illegible]

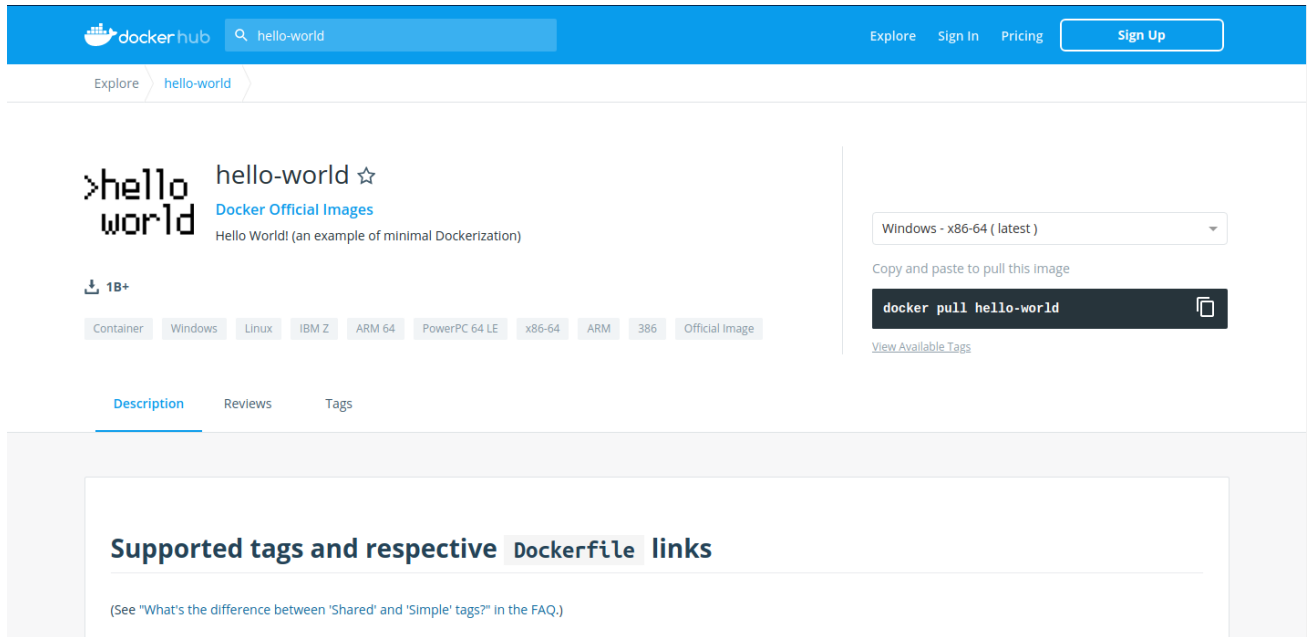
Install Docker

Installing Docker on your computer takes a little time but it is reasonably straight forward and it is a one-time setup. . Docker installation is much easier on an Atmosphere instance with the 'ezd' command.

\$ezd

Use 'docker pull' to get the image

Go to and search for 'hello-world' in the search bar at the top of the page.



The screenshot shows the Docker Hub interface for the 'hello-world' image. At the top, there's a search bar with 'hello-world' entered. Below the search bar, the 'hello-world' image is displayed with a star icon and the text 'Docker Official Images'. The description reads 'Hello World! (an example of minimal Dockerization)'. There are 18+ downloads indicated. A list of supported architectures is shown: Container, Windows, Linux, IBM Z, ARM 64, PowerPC 64 LE, x86-64, ARM, 386, and Official image. A dropdown menu is set to 'Windows - x86-64 (latest)'. A button 'docker pull hello-world' is present. Below the image, there is a section titled 'Supported tags and respective Dockerfile links' with a link to the FAQ.

Click on the ‘tag’ tab to see all the available ‘hello-world’ images.

Click the ‘copy’ icon at the right to copy the docker pull command that we will need on the command line.

Now you will need to pull the image from the registry onto your computer. Use the ‘docker pull’ command you copied from the registry above.

Note: If you are working on a system for which you don’t have root permissions you will need to use ‘sudo’ and provide your password. Like this:

```
$ sudo docker pull hello-world:latest
```

Now list the files in your current working directory

```
$ ls -l
```

Where is the image you just pulled? Docker saves container images to the Docker directory (where Docker is installed). You won’t ever see them in your working directory.

Use ‘docker images’ to see all the images on your computer:

```
$ sudo docker images
```

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org

Introduction to Docker



14.1 Prerequisites

There are no specific skills needed for this tutorial beyond a basic comfort with the command line and using a text editor.

- Install Docker on your laptop:
 - [Mac](#)
 - [Windows](#)
 - [Ubuntu](#)
- Install Docker on a featured Atmosphere image:

```
$ ezd
```

14.2 1.0 Docker Run

As we covered in the [previous section](#), containers can be found in “registries” (such as the Docker Hub). You can also build your own container, but we’ll cover that tomorrow (See [Advanced Section](#)).

When you’re looking for the right container, you can search for images within a given registry directly from the command line using `docker search` (after you’ve logged into that registry).

```

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

```

Note: Depending on how and where you've installed Docker, you may see a permission denied error after running the `$ docker run hello-world` command. If you're on Linux, you may need to prefix your Docker commands with `sudo`. Alternatively to run docker command without `sudo`, you need to add your user name (who

has root privileges) to the docker “group”.

Create the docker group:

```
$ sudo groupadd docker
```

Add your user to the docker group:

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

Log out or close terminal and log back in and your group membership will be initiated

The single most common command that you’ll use with Docker is `docker run` ([help manual](#)).

`docker run` starts a container and executes the default entrypoint, or any other command line statement that follows `run`.

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

Note: To find out more about a Docker images, run `docker inspect hello-world`.

In the demo above, you could have used the `docker pull` command to download the `hello-world` image first.

When you executed the command `docker run alpine`, Docker looked for the image, did not find it, and then ran a `docker pull` behind the scenes to download the `alpine` image with the `:latest` tag.

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

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

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

Images need to have an `ENTRYPOINT` set in their Dockerfile recipe in order for them to return a result when they are run. The `hello-world` image echos out the statement that it is present when it executes.

You can change the entrypoint of a container by making a statement after the `repository/container_name:tag`:

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

In this case, the Docker client dutifully ran the `echo` command in our `alpine` container and then exited. 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!

Now it’s time to see the `docker ps` command which shows you all containers that are currently running.

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
↳ STATUS	PORTS	NAMES	

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

```
$ docker ps --all
```

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

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

Try another command, this time to access the container as a shell:

```
$ docker run alpine sh
```

Wait, nothing happened! Is that a bug? Well, no.

The container will exit after running any scripted commands such as `sh`, unless they are run in an “interactive” terminal (TTY) - so for this example to not exit, you need to add the `-i` for interactive and `-t` for TTY. You can run them both in a single flag as `-it`, which is the more common way of adding the flag:

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

The prompt should change to something more like `/ # ``` -- You are now running a shell inside the container. Try out a few commands like ```ls -l`, `uname -a` and others.

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

```
/ # exit
```

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

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

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

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

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

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

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

```
$ docker attach 0db38ea51a48
```

14.2.1 1.1 House Keeping and Cleaning Up

Docker images are cached on your machine in the location where Docker was installed. These image files are not visible in the same directory where you might have used `docker pull <imagename>`.

Some Docker images can be large. Especially Data Science images with many libraries and packages pre-installed.

Important: Pulling many images from the Docker Registries may fill up your hard disk!

To inspect your system and disk use:

```
$ docker system info
$ docker system df
```

To find out how many images are on your machine, type:

```
$ docker images --help
```

To remove images that you no longer need, type:

```
$ docker system prune --help
```

This is where it becomes important to differentiate between *images*, *containers*, and *volumes* (which we'll get to more in a bit). You can take care of all of the dangling images and containers on your system. Note, that `prune` will not removed your cached *images*

```
$ docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

Are you sure you want to continue? [y/N]
```

If you add the `-af` flag it will remove "all" -a dangling images, empty containers, AND ALL CACHED IMAGES with "force" -f.

14.2.2 2.0 Managing Docker images

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

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              bionic             47b19964fb50       4 weeks ago       88.1MB
alpine              latest             caf27325b298       4 weeks ago       5.53MB
hello-world         latest             fce289e99eb9       2 months ago      1.84kB
.....
```

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

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

14.2.3 2.1 Pulling and Running a JupyterLab or RStudio-Server

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

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

```
$ docker search jupyter
NAME                                DESCRIPTION
jupyter/datascience-notebook      Jupyter Notebook Data Science Stack from
jupyter/all-spark-notebook         Jupyter Notebook Python, Scala, R, Spark,
jupyterhub/jupyterhub             JupyterHub: multi-user Jupyter notebook
jupyter/scipy-notebook            Jupyter Notebook Scientific Python Stack
jupyter/tensorflow-notebook       Jupyter Notebook Scientific Python Stack w/
jupyter/pyspark-notebook          Jupyter Notebook Python, Spark, Mesos Stack
jupyter/minimal-notebook          Minimal Jupyter Notebook Stack from https://
jupyter/base-notebook             Small base image for Jupyter Notebook
jupyterhub/singleuser             single-user docker images for use with
jupyter/r-notebook                Jupyter Notebook R Stack from https://
jupyter/nbviewer                  Jupyter Notebook Viewer
mikebirdgeneau/jupyterlab         Jupyterlab based on python / alpine linux
```

(continues on next page)

(continued from previous page)

jupyter/demo	(DEPRECATED) Demo of the IPython/Jupyter	
↪ Not... 14		
eboraas/jupyter	Jupyter Notebook (aka IPython Notebook)	
↪ with... 12	[OK]	
jupyterhub/k8s-hub		
↪ 11		
nbgallery/jupyter-alpine	Alpine Jupyter server with nbgallery	
↪ integra... 9		
jupyter/repo2docker	Turn git repositories into Jupyter enabled	
↪ D... 7		
jupyterhub/configurable-http-proxy	node-http-proxy + REST API	
↪ 5	[OK]	
...		

Search for images on Docker Hub which contain the string ‘rstudio’

```
$ docker search rstudio
```

NAME	STARS	OFFICIAL	DESCRIPTION	
rocker/rstudio			RStudio Server image	
↪ 289			[OK]	
opencpu/rstudio			OpenCPU stable release with rstudio-server	
↪ (... 29			[OK]	
rocker/rstudio-stable			Build RStudio based on a debian:stable	
↪ (debi... 16			[OK]	
dceoy/rstudio-server			RStudio Server	
↪ 8			[OK]	
rocker/rstudio-daily				
↪ 6			[OK]	
rstudio/r-base			Docker Images for R	
↪ 6				
rstudio/r-session-complete			Images for sessions and jobs in RStudio	
↪ Serv... 4				
rstudio/rstudio-server-pro			Default Docker image for RStudio Server Pro	
↪ 1				
aghorbani/rstudio-h2o			An easy way to start rstudio and H2O to run	
↪ ... 1			[OK]	
centerx/rstudio-pro			NA	
↪ 1			[OK]	
mobilizingcs/rstudio			RStudio container with mz packages pre-	
↪ insta... 1			[OK]	
calpolydatascience/rstudio-notebook			RStudio notebook	
↪ 1			[OK]	
...				

2.2 Interactive Containers

Let’s go ahead and run some basic Integrated Development Environment images from “trusted” organizations on the Docker Hub registry.

When we want to run a container that runs on the open internet, we need to add a **TCP or UDP port number** from which we can access the application in a browser using the machine’s IP (Internet Protocol) address or DNS (Domain Name Service) location.

Here are some examples to run basic RStudio and Jupyter Lab:

```
$docker run --rm -p 8787:8787 -e PASSWORD=cc2020 rocker/rstudio
```

```
$docker run --rm -p 8888:888 jupyter/base-notebook
```

Note: We've added the `--rm` flag, which means the container will automatically removed from the cache when the container is exited.

When you start an IDE in a terminal, the terminal connection must stay active to keep the container alive.

If we want to keep our window in the foreground we can use the `-d` - the *detached* flag will run the container as a background process, rather than in the foreground. When you run a container with this flag, it will start, run, telling you the container ID:

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

Unable to find image 'jupyter/base-notebook:latest' locally
latest: Pulling from jupyter/base-notebook
5c939e3a4d10: Pull complete
c63719cdbe7a: Pull complete
19a861ea6baf: Pull complete
651c9d2d6c4f: Pull complete
21b673dc817c: Pull complete
1594017be8ef: Pull complete
b392f2c5ed42: Pull complete
8e4f6538155b: Pull complete
7952536f4b86: Pull complete
61032726be98: Pull complete
3fc223ec0a58: Pull complete
23a29aed8d6e: Pull complete
25ed667252a0: Pull complete
434b2237517c: Pull complete
d33fb9062f74: Pull complete
fdc8c4d68c3d: Pull complete
Digest: sha256:3b8ec8c8e8be8023f3eeb293bbcb1d80a71d2323ae40680d698e2620e14fdcbc
Status: Downloaded newer image for jupyter/base-notebook:latest
561016e4e69e22cf2f3b5ff8cbaa229779c2bdf3bdece89b66957f3f3bc5b734
$
```

Note, that your terminal is still active and you can use it to launch more containers. To view the running container, use the `docker ps` command

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
↪ STATUS          PORTS              NAMES
561016e4e69e       jupyter/base-notebook "tini -g -- start-no..." About a minute
↪ ago             Up About a minute  8888/tcp, 0.0.0.0:8888->8888/tcp  affectionate_banzai
```

What if we want a Docker container to *always (re)start*, even after we reboot our machine?

```
$ docker run --restart always
```

14.3 3. Managing Data in Docker

It is possible to store data within the writable layer of a container, but there are some limitations:

- The data doesn't persist when that container is no longer running, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.
- Its better to put your data into the container **AFTER** it is build - this keeps the container size smaller and easier to move across networks.

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

- **volumes**
- **bind mounts**
- **tmpfs volumes**

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

14.3.1 3.1 Volumes

volumes

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

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

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

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

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

- In the case of named volumes, the first field is the name of the volume, and is unique on a given host machine.
- The second field is the path where the file or directory are mounted in the container.

- The third field is optional, and is a comma-separated list of options, such as `ro`.

```
-v /home/username/your_data_folder:/data
```

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

```
$docker run --rm -v $(pwd):/work -p 8787:8787 -e PASSWORD=cc2020 rocker/rstudio
```

In the Jupyter Lab example, we use the `-e` environmental flag to re-direct the URL of the container at the localhost

```
$docker run --rm -v $(pwd):/work -p 8888:8888 -e REDIRECT_URL=http://localhost:8888_
↪ jupyter/base-notebook
```

Once you're in the container, you will see that the `/work` directory is mounted in the working directory.

Any data that you add to that folder outside the container will appear **INSIDE** the container. And any work you do inside the container saved in that folder will be saved **OUTSIDE** the container as well.

14.4 Docker Commands

Command	Usage
<code>docker pull</code>	Download an image from Docker Hub
<code>docker run</code>	<i>Usage:</i> <code>docker run -it user/image:tag</code> starts a container with an entrypoint
<code>docker build</code>	<i>Usage:</i> <code>docker build -t user/image:tag .</code> Builds a docker image from a Dockerfile in current working directory. <code>-t</code> for tagname
<code>docker images</code>	List all images on the local machine
<code>docker tag</code>	Add a new tag to an image
<code>docker login</code>	Authenticate to the Docker Hub requires username and password
<code>docker push</code>	<i>Usage:</i> <code>docker push user/image:tag</code> Upload an image to Docker Hub
<code>docker inspect</code>	<i>Usage:</i> <code>docker inspect containerID</code> Provide detailed information on constructs controlled by Docker
<code>docker ps -a</code>	List all containers on your system
<code>docker rm</code>	<i>Usage:</i> <code>docker rm -f <container></code> Deletes a container <code>-f</code> remove running container
<code>docker rmi</code>	Deletes an <i>image</i>
<code>docker stop</code>	<i>Usage:</i> <code>docker stop <container></code> Stop a running container
<code>docker system</code>	<i>Usage:</i> <code>docker system prune</code> Remove old images and cached layers <i>Usage:</i> <code>docker system df</code> View system details (cache size)

14.5 Getting more help with Docker

- The command line tools are very well documented:

```
$ docker --help
# shows all docker options and summaries
```

```
$ docker COMMAND --help
# shows options and summaries for a particular command
```

- Learn more about docker

14.6 4. Extra Demos

14.6.1 4.1 Portainer

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

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

4.1.1 Installation

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

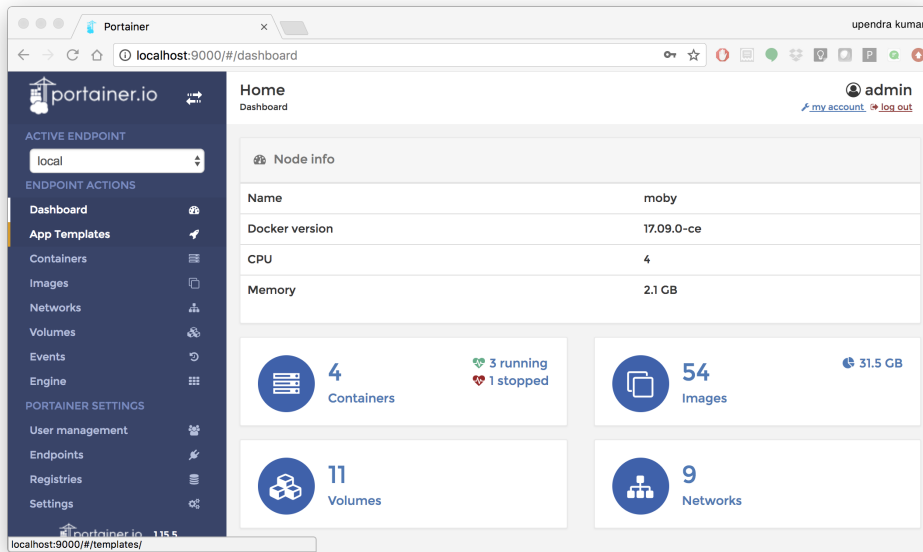
```
# on CyVerse Atmosphere:
$ ezd -p

$ 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.



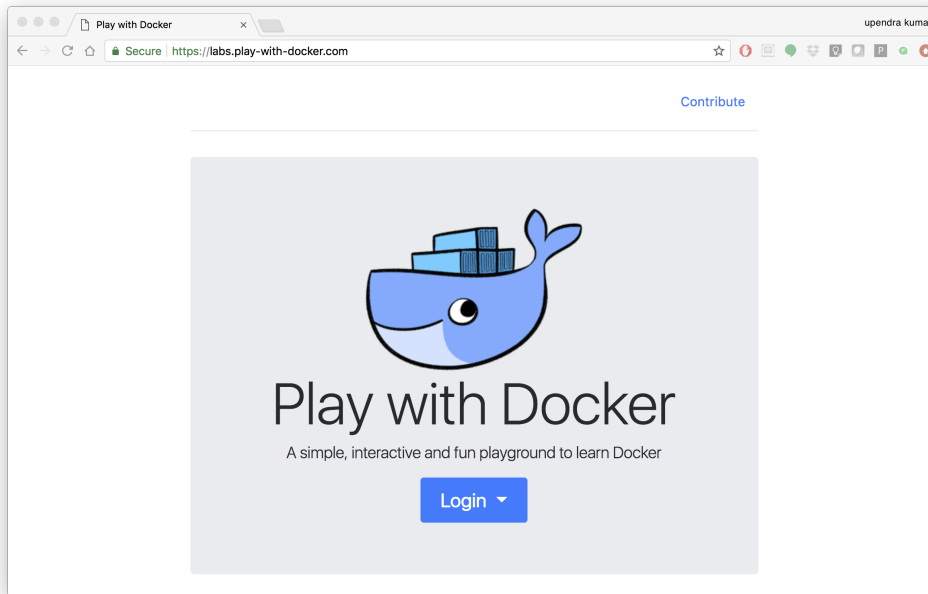
14.6.2 4.2 Play-with-docker (PWD)

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

4.2.1 Installation

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

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



CHAPTER 15

Advanced Docker

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

- Push a Docker image to the Docker Hub Registry
- Modifying a Dockerfile and creating a new container
- Establish a Docker Hub autobuild on GitHub with CI/CD

15.1 1.0 The Dockerfile

Note: This is one of the official Docker images provided by the [Jupyter Project](#) for you to build your own data science notebooks on:

Create a file called Dockerfile, and add content to it as described below, e.g.

```
$ nano Dockerfile
```

Important: Dockerfile needs to be capitalized.

Contents of our Dockerfile:

```
# base image
FROM jupyter/scipy-notebook:latest

# reset user to root for installing additional packages
USER root

# Install a few dependencies for iCommands, text editing, and monitoring instances
RUN apt-get update && apt-get install -y \
```

(continues on next page)

(continued from previous page)

```

apt-transport-https \
gcc \
gnupg \
htop \
less \
libfuse2 \
libpq-dev \
libssl1.0 \
lsb \
nano \
nodejs \
python-requests \
software-properties-common \
vim

# Install iCommands
RUN wget https://files.renci.org/pub/irods/releases/4.1.12/ubuntu14/irods-icommands-4.
↪1.12-ubuntu14-x86_64.deb && \
dpkg -i irods-icommands-4.1.12-ubuntu14-x86_64.deb && \
rm irods-icommands-4.1.12-ubuntu14-x86_64.deb

# reset container user to jovyan
USER jovyan

# set the work directory
WORKDIR /home/jovyan

# copy configuration json and entry file into the container
COPY jupyter_notebook_config.json /opt/conda/etc/jupyter/jupyter_notebook_config.json
COPY entry.sh /bin

# expose the public port we want to run on
EXPOSE 8888

# directory will be populated by iCommands when entry.sh is run
RUN mkdir -p /home/jovyan/.irods

ENTRYPOINT ["bash", "/bin/entry.sh"]

```

Note: We use a code line escape character `\` to allow single line scripts to be written on multiple lines in the Dockerfile.

We also use the double characters `&&` which essentially mean “if true, then do this” while executing the code. The `&&` can come at the beginning of a line or the end when used with `\`

Now let’s talk about what each of those lines in the Dockerfile mean.

1. We’ll start by specifying our base image, using the FROM statement

```
FROM jupyter/scipy-notebook:latest
```

2. Copy existing files into the new image by using the COPY statement

```

COPY entry.sh /bin
COPY jupyter_notebook_config.json /opt/conda/etc/jupyter/jupyter_notebook_config.json

```

Before we forget, create a new file called `entry.sh` – use your preferred text editor to create the file, e.g. `nano entry.sh` and put it in the same directory as `Dockerfile`

```
#!/bin/bash

echo '{"irods_host": "data.cyverse.org", "irods_port": 1247, "irods_user_name": "
↳ $IPLANT_USER", "irods_zone_name": "iplant"}' | envsubst > $HOME/.irods/irods_
↳ environment.json

exec jupyter lab --no-browser
```

The `entry.sh` file creates an iRODS environment `.json` which has CyVerse Data Store configurations pre-written. It also tells Docker to start Jupyter Lab and to not pop open a browser tab when doing so.

We also create a `jupyter_notebook_config.json` which will help launch the notebook without a token

```
{
  "NotebookApp": {
    "allow_origin" : "*",
    "token": "",
    "password": "",
    "nbserver_extensions": {
      "jupyterlab": true
    }
  }
}
```

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

```
EXPOSE 8888
```

Note: What about `CMD`?

Notice that unlike some other `Dockerfile` this one does not end with a `CMD` command statement. This is on purpose.

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

Can you guess what will happen if we don't specify an `ENTRYPOINT` or `CMD`?

4. Setting a new entrypoint

When this container is run, it will now use a different default `ENTRYPOINT` than the original container from `jupyter/scipy-notebook:latest`

```
ENTRYPOINT ["bash", "/bin/entry.sh"]
```

This entrypoint runs the shell script `entry.sh` which we just copied into the image

A quick summary of the few basic commands we used in our `Dockerfiles`.

- **FROM** starts the `Dockerfile`. It is a requirement that the `Dockerfile` must start with the `FROM` command. Images are created in layers, which means you can use another image as the base image for your own. The `FROM` command defines your base layer. As arguments, it takes the name of the image. Optionally, you can add the Dockerhub username of the maintainer and image version, in the format `username/imagename:version`.
- **RUN** is used to build up the Image you're creating. For each `RUN` command, Docker will run the command then create a new layer of the image. This way you can roll back your image to previous states easily. The

syntax for a RUN instruction is to place the full text of the shell command after the RUN (e.g., RUN mkdir /user/local/foo). This will automatically run in a /bin/sh shell. You can define a different shell like this: RUN /bin/bash -c 'mkdir /user/local/foo'

- **COPY** copies local files into the container.
- **CMD** defines the commands that will run on the Image at start-up. Unlike a RUN, this does not create a new layer for the Image, but simply runs the command. There can only be one CMD per a Dockerfile/Image. If you need to run multiple commands, the best way to do that is to have the CMD run a script. CMD requires that you tell it where to run the command, unlike RUN. So example CMD commands would be:
- **EXPOSE** creates a hint for users of an image which ports provide services. It is included in the information which can be retrieved via `$ docker inspect <container-id>`.

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

15.2 2.0 Docker Build

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

```
DOCKERHUB_USERNAME=<YOUR_DOCKERHUB_USERNAME>
```

For example this is how I assign my dockerhub username

```
DOCKERHUB_USERNAME=tswetnam
```

Now build the image using the following command:

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

If everything went well, your image should be ready! Run docker images and see if your image `$DOCKERHUB_USERNAME/jupyterlab-scipy:cyverse` shows.

15.2.1 2.1 Test the image

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

```
$ docker run --rm -it -p 8888:8888 $DOCKERHUB_USERNAME/jupyterlab-scipy:cyverse
```

You should see something like this:

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

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

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

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

Note: If you want to learn more about Dockerfiles, check out [Best practices for writing Dockerfiles](#).

15.2.2 2.2 Tagging images

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

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

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

```
$ docker tag jupyterlab-scipy:cyverse $DOCKERHUB_USERNAME/jupyterlab-scipy:cyverse
```

15.3 3.0 Publishing your image

15.3.1 3.1 Log into the Docker Hub Registry

Note: If you don't have an account, sign up for one at [Docker Cloud](#) or [Docker Hub](#). Make note of your username – it may or may not be the same as your email, GitHub, or CyVerse username. There are several advantages to registering with registries like DockerHub which we will see later on in the session.

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

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

Login Succeeded
```

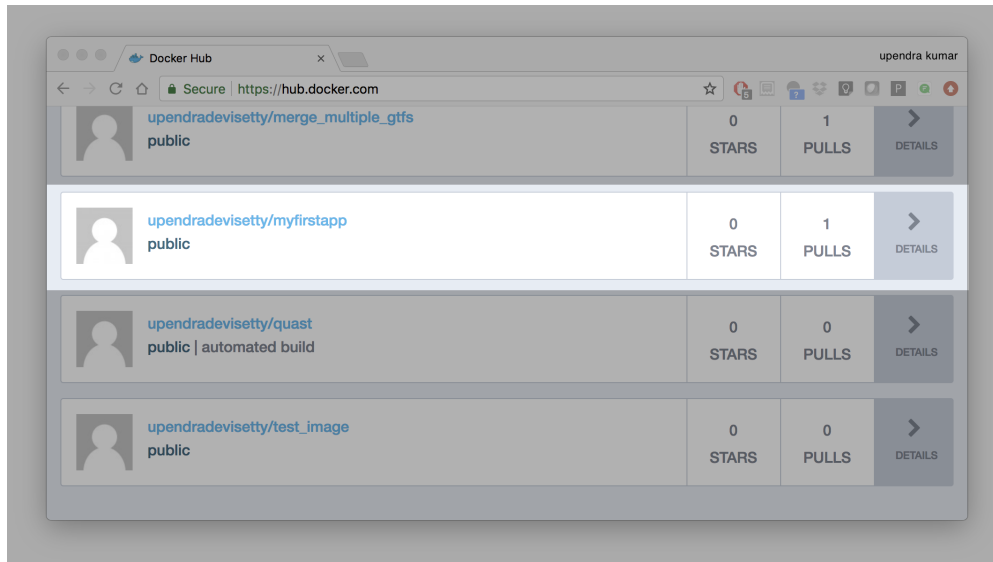
If it is your first time logging in you will be queried for your username and password.

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

If you don't have a Docker ID, head over to <https://hub.docker.com> to create one. Upload your tagged image to the Dockerhub repository

```
$ docker push $DOCKERHUB_USERNAME/jupyterlab-scipy:cyverse
```

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



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

15.3.2 3.2 Pull and run the image from the remote repository

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

```
$ docker run -p 8888:8888 --name notebooktest $DOCKERHUB_USERNAME/jupyterlab-
->scipy:cyverse
```

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

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

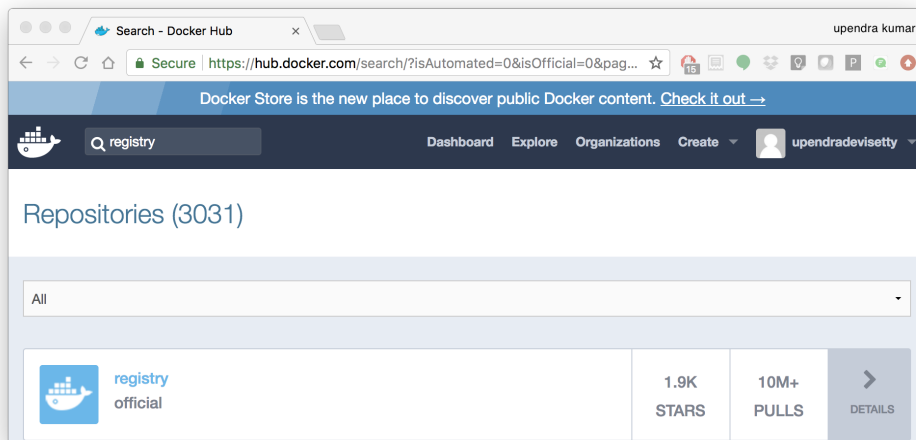
15.3.3 3.3 Private repositories

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

3.4 Pull down the Registry Image

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

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



Run a container from `registry` Dockerhub image

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

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

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

Tag the image that you want to push

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

```
$ REGISTRY=localhost:5000

$ docker tag $DOCKERHUB_USERNAME/jupyterlab-scipy:cyverse $REGISTRY/$(whoami)/
↪ mynotebook:1.0
```

Publish the image into the local registry

Finally push the image to the local registry

```
$ docker push $REGISTRY/$(whoami)/mynotebook:1.0
The push refers to a repository [localhost:5000/julianp/mynotebook]
64436820c85c: Pushed
831cff83ec9e: Pushed
c3497b2669a8: Pushed
1c5b16094682: Pushed
c52044a91867: Pushed
60ab55d3379d: Pushed
1.0: digest: sha256:5095dea8b2cf308c5866ef646a0e84d494a00ff0e9b2c8e8313a176424a230ce_
↪ size: 1572
```

Pull and run the image from the local repository

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

```
$ docker run -P --name=mynotebooklocal $REGISTRY/$(whoami)/jupyterlab-scipy:cyverse
```

15.4 4.0 Automated Docker image building from GitHub

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

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

Automated Builds have several advantages:

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

15.4.1 4.1 Prerequisites

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

Note:

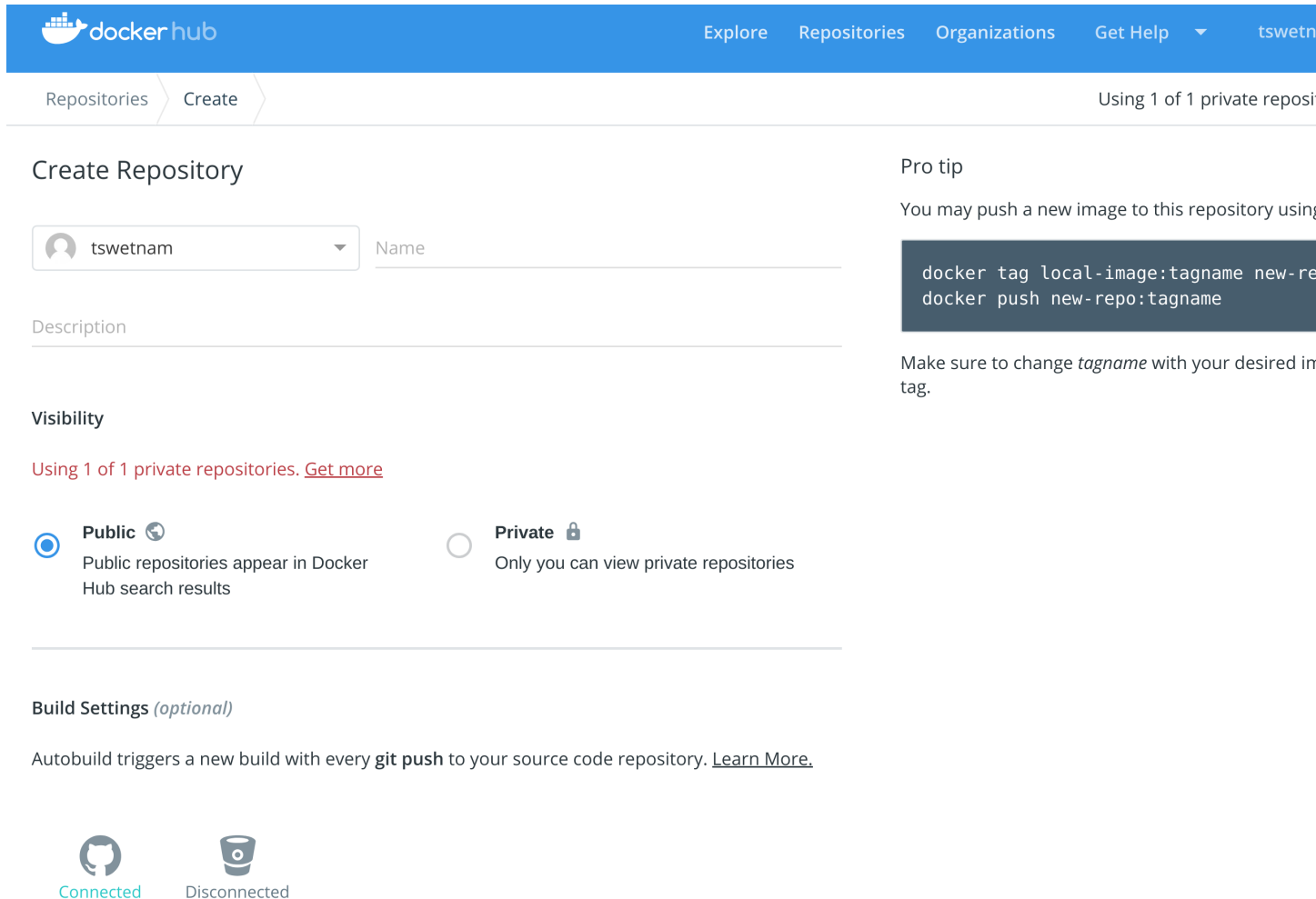
- If you have previously linked your Github or Bitbucket account, you must have chosen the Public and Private connection type. To view your current connection settings, log in to Docker Hub and choose Profile > Settings > Linked Accounts & Services.
- Building Windows containers is not supported.

15.4.2 4.2 Link your Docker Hub account to GitHub

1. Log into Docker Hub.
2. Click “Create Repository+”



3. Click the Build Settings and select GitHub.

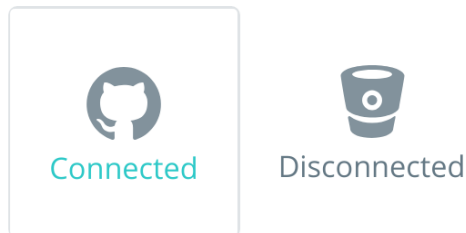


The system prompts you to choose between **Public and Private** and **Limited Access**. The **Public** and **Private** connection type is required if you want to use the Automated Builds.

4. Press `Select` under **Public and Private** connection type. If you are not logged into GitHub, the system prompts you to enter GitHub credentials before prompting you to grant access. After you grant access to your code repository, the system returns you to Docker Hub and the link is complete.

Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository. [Learn More](#)



tyson-swetnam



cc-camp

▼ Click here to customize the build settings

BUILD RULES +

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

Source Type	Source	Docker Tag	Dockerfile location	Build Caching
Branch ▼	master	latest	Dockerfile	<input checked="" type="checkbox"/>

► View example build rules

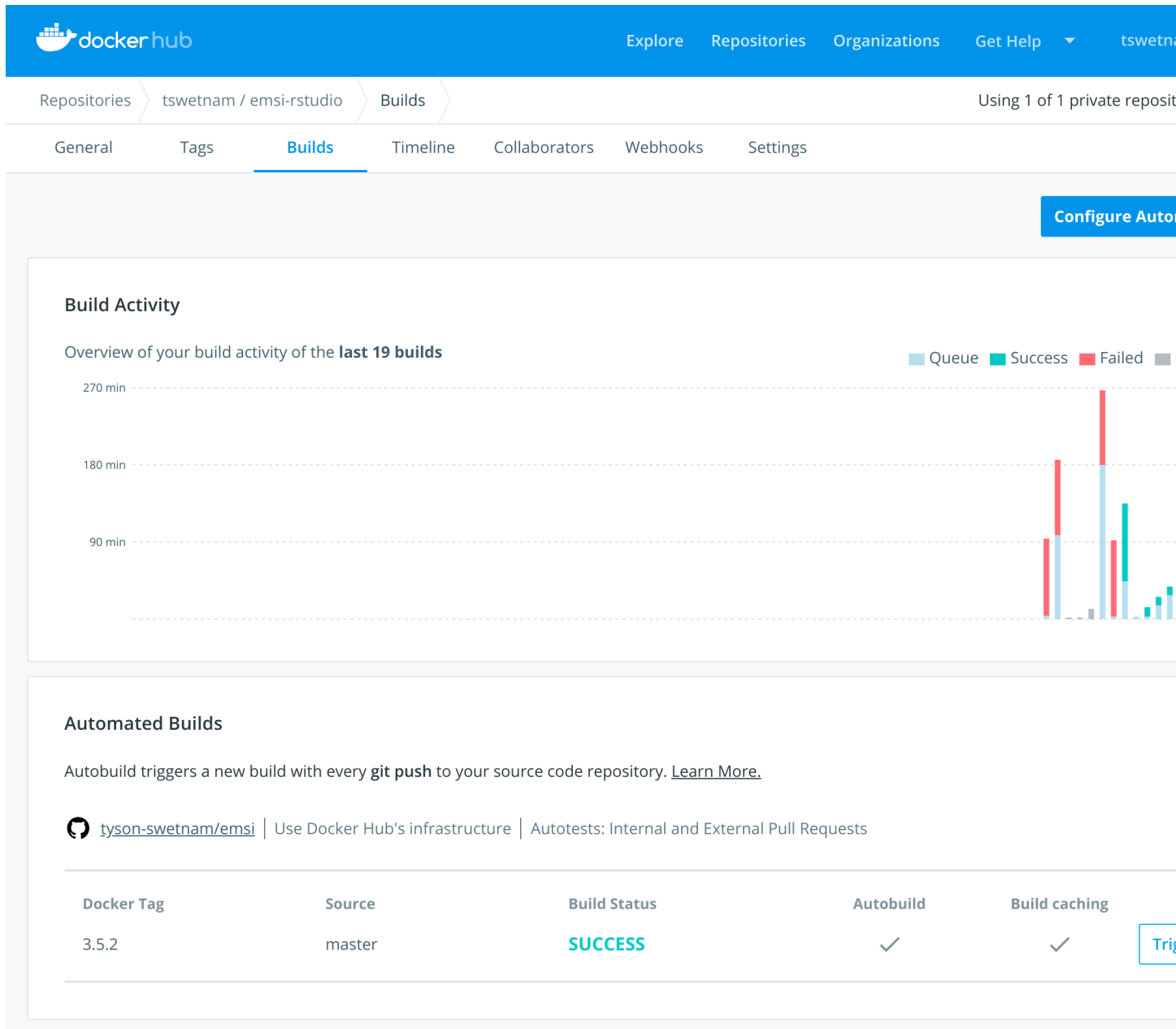
Cancel

Create

Create & B

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:



15.4.3 4.3 Automated Container Builds

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

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

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

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

(continues on next page)

(continued from previous page)

```
$ git status
On branch master

Initial commit

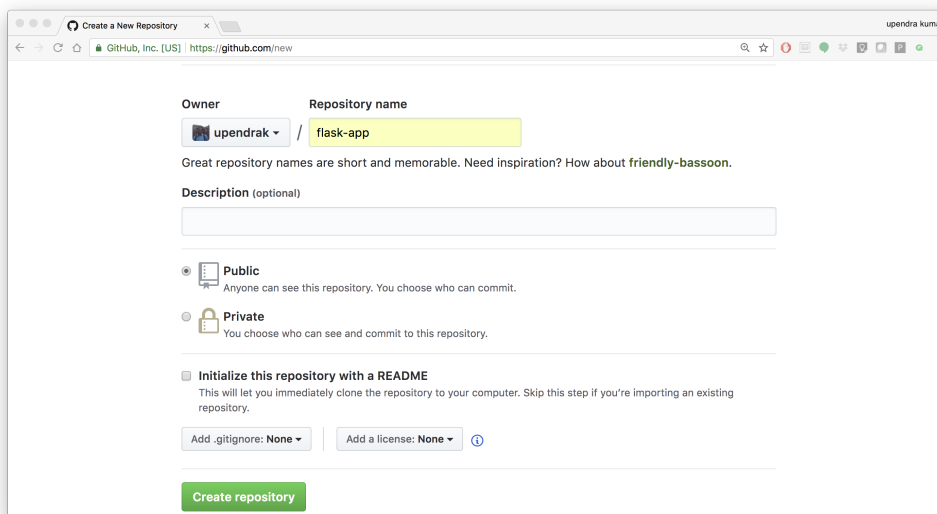
Untracked files:
(use "git add <file>..." to include in what will be committed)

    Dockerfile
    model.py

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

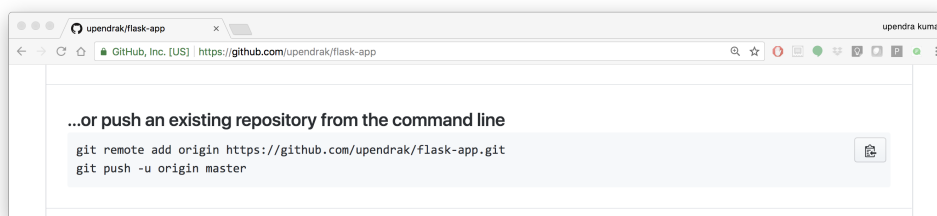
$ git add * && git commit -m "Add files and folders"
[master (root-commit) a4f732a] Add files and folders
 2 files changed, 10 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 model.py
```

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



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

3. Push the repository to github




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

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

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

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



[Explore](#)
[Repositories](#)
[Organizations](#)
[Get Help](#)
tswetna

[Repositories](#)
[tswetnam / emsi-rstudio](#)
[Builds](#)
[Edit](#)
Using 1 of 1 private repositories

[General](#)
[Tags](#)
[Builds](#)
[Timeline](#)
[Collaborators](#)
[Webhooks](#)
[Settings](#)

Build configurations

SOURCE REPOSITORY



tyson-swetnam

×

emsi

NOTE: Changing source repository may affect existing build rules.

BUILD LOCATION

Build on Docker Hub's infrastructure

AUTOTEST

☐ Off
 ☐ Internal Pull Requests
 ☒ Internal and External Pull Requests

REPOSITORY LINKS

☐ Off
 ☒ Enable for Base Image ⓘ

BUILD RULES +

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

Source Type	Source	Docker Tag	Dockerfile location	Build Context ⓘ	Autobuild	Build Caching
Branch	master	3.5.2	Dockerfile	/docker/rstud	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

▶ View example build rules

BUILD ENVIRONMENT VARIABLES +

Delete

Cancel

Save

Save and

Build triggers

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

Trigger name

+

Name

Trigger Url

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

5. Customize the automated build by pressing the [Click here to customize behavior link](#).

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

Customize Autobuild Tags

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

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

Create

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

- Enter the `master` (default) for the name of the branch.
- Leave the Dockerfile location as is.
- Recall the file is in the root of your code repository.
- Specify `1.0` for the Tag Name.

6. Click `Create`.

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

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

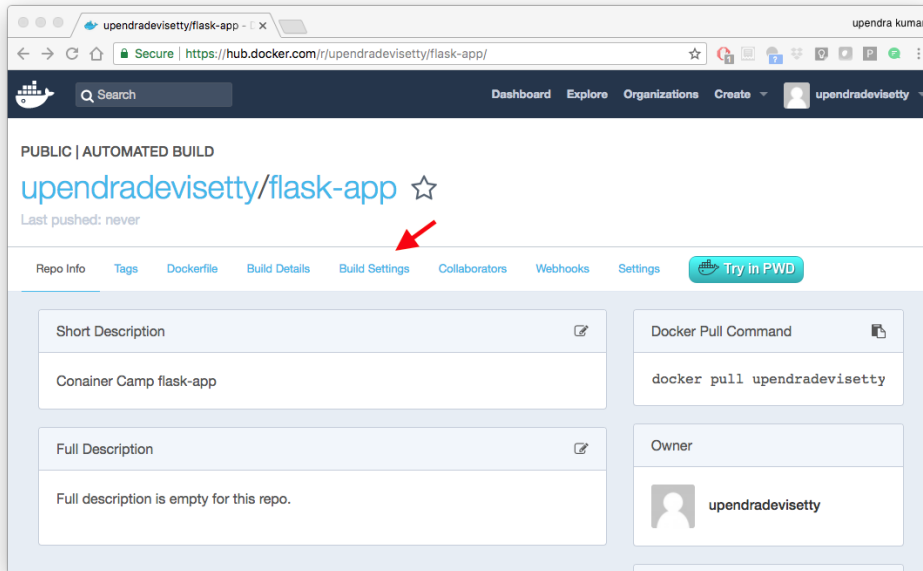
Warning: You can only trigger one build at a time and no more than one every five minutes. If you already have a build pending, or if you recently submitted a build request, Docker ignores new requests.

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

7. Manually Trigger a Build

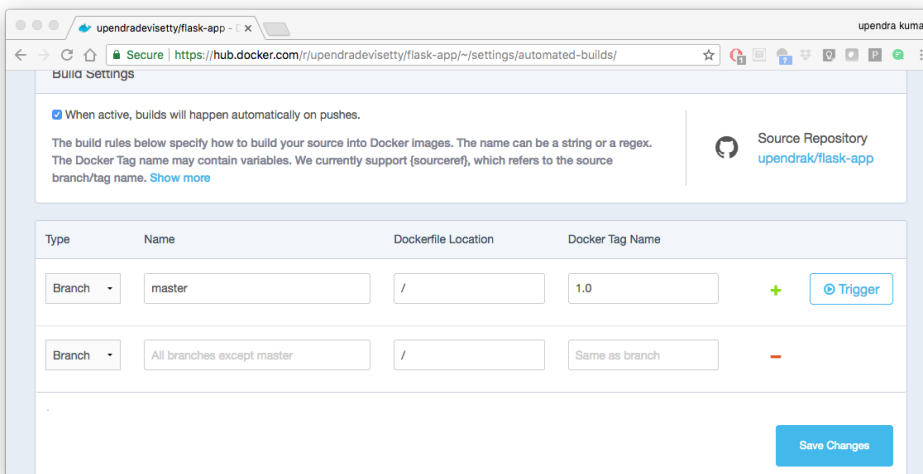
Before you trigger an automated build by pushing to your GitHub mynotebook repo, you'll trigger a manual build. Triggering a manual build ensures everything is working correctly.

From your automated build page choose Build Settings



Press Trigger button and finally click Save Changes.

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



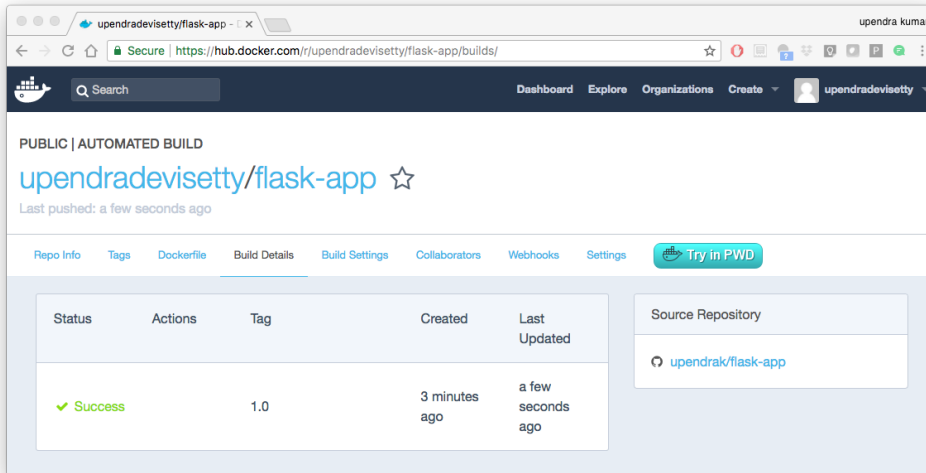
8. Review the build results

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

Navigate to the Build Details page.

Wait until your image build is done.

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



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

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

15.5 5.0 Volumes Continued

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

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

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

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

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

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

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

Let's create a volume

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

List volumes:

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

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

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

Remove a volume

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

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

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

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

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

You should see something like:

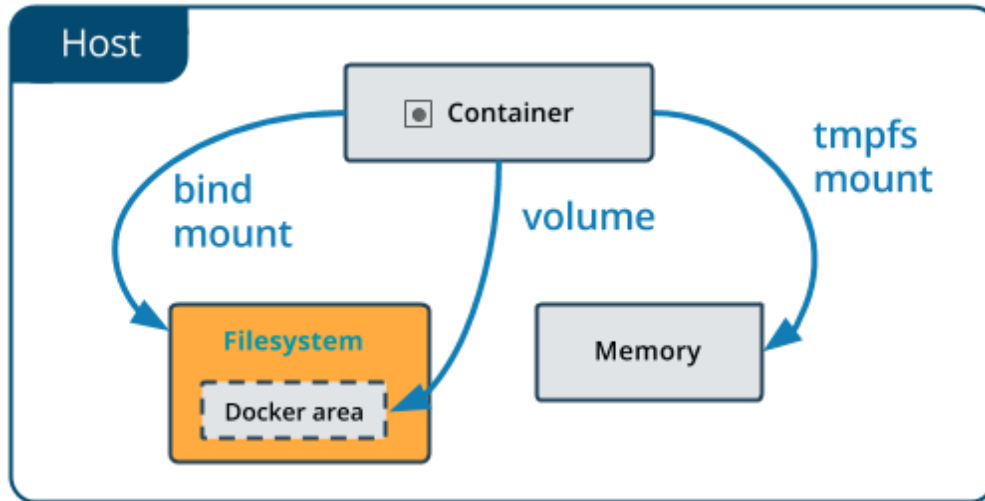
After running either of these examples, run the following commands to clean up the container and volume.

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

15.5.1 5.3 Bind mounts

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

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



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

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

Start a container with a bind mount

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

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

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

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

Check that the output looks right.

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

Use a read-only bind mount

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

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

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

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

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

15.6 6.0 Docker Compose for multi-container apps

Docker Compose is a tool for defining and running multi-container Docker applications. It requires you to have a `docker-compose.yml` file.

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

For Linux users

```
sudo curl -L https://github.com/docker/compose/releases/download/1.25.4/docker-
↪ compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

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

Main advantages of Docker compose include:

- Your applications can be defined in a YAML file where all the same options required in `docker run` are now defined (reproducibility).
- It allows you to manage your application(s) as a single entity rather than dealing with starting individual containers (simplicity).

Let's now create a Docker Compose `.yml` that calls Jupyter Lab SciPy

1. Copy or create the `jupyter_compose` directory

```
$ mkdir jupyter_compose && cd jupyter_compose
```

We will also create `data/` and `notebooks/` folders to stage our future data and notebook work

```
$ mkdir jupyter_compose/data
$ mkdir jupyter_compose/notebooks
```

2. Copy or create a `entry.sh` and a `jupyter_notebook_config.json` in the `jupyter_compose/` directory

`entry.sh` creates an iRODS environment JSON with the user's name and CyVerse (iPlant) zone.

```
#!/bin/bash

echo '{"irods_host": "data.cyverse.org", "irods_port": 1247, "irods_user_name": "
↪$IPLANT_USER", "irods_zone_name": "iplant"}' | envsubst > $HOME/.irods/irods_
↪environment.json

exec jupyter lab --no-browser
```

jupyter_notebook_config.json starts the notebook without requiring you to add the token:

```
{
  "NotebookApp": {
    "allow_origin" : "*",
    "token": "",
    "password": "",
    "nbserver_extensions": {
      "jupyterlab": true
    }
  }
}
```

3. create your docker-compose.yml in the same directory jupyter_compose/
4. Edit the contents of your docker-compose.yml

```
version: "3"
services:
  scipy-notebook:
    build: .
    image: jupyter/scipy-notebook
    volumes:
      - "./notebooks:/notebooks"
      - "./data:/data"
      - "${LOCAL_WORKING_DIR}:/home/jovyan/work"
    ports:
      - "8888:8888"
    container_name: jupyter_scipy
    command: "entry.sh"
    restart: always
```

4. Create a Dockerfile (use the same Jupyter SciPy Notebook as in Advanced Section 1.0)
5. Build the container with docker-compose instead of docker build

Note: Handling containers with Docker Compose is fairly simple

```
docker-compose up
```

mounts the directory and starts the container

```
docker-compose down
```

destroys the container

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

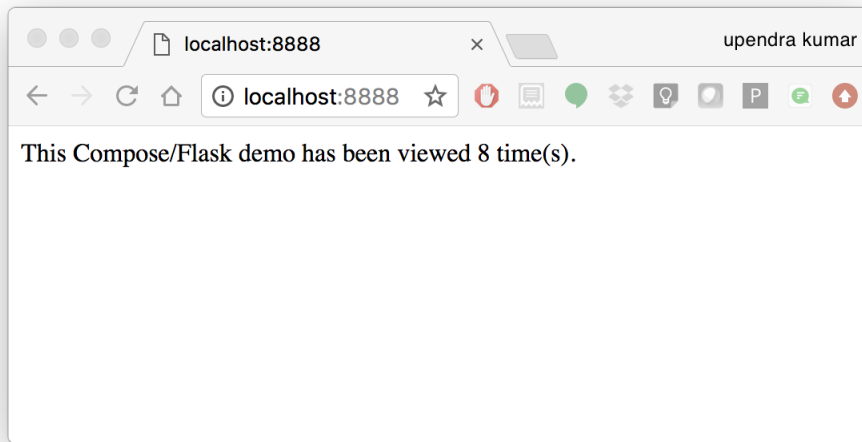
- The web service builds from the Dockerfile in the current directory.

- Forwards the container's exposed port to port 8888 on the host.
- Mounts the project directory on the host to /notebooks inside the container (allowing you to modify code without having to rebuild the image).
- `restart: always` means that it will restart whenever it fails.

5. Run the container

```
$ docker-compose up -d
```

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



Introduction to Singularity



16.1 1. Prerequisites

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

Note:

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

Key Differences:

Docker:

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

Singularity:

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

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

16.2 2. Singularity Installation

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

Singularity is more likely to be used on a remote system that you don't have control of (e.g. HPC).

16.2.1 2.1 Install Singularity on Laptop

To Install Singularity on your laptop or desktop PC follow the instructions from Singularity: <https://www.sylabs.io/guides/3.5/user-guide/installation.html#installation>

16.2.2 2.2 HPC

Load the Singularity module on a HPC

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

Most HPC systems are running Environment Modules with the simple command *module*.

You can check to see what is available:

```
$ module avail singularity
```

If Singularity is installed, load a specific version:

```
$ module load singularity/3/3.5
```

16.2.3 2.3 Atmosphere Cloud

CyVerse staff have deployed an Ansible playbook called *ez* for software installation which includes [Singularity](#). This command only requires you to type a short line of code to install an entire software stack with all of its dependencies.

Start any *Featured* instance on *Atmosphere* `<../cyverse/boot.html>_`.

Type in the following in a web shell or *ssh* terminal.

```
$ ezs -r 3.5.1
DEBUG: set version to 3.5.1

* Updating ez singularity and installing singularity (this may take a few minutes, ☕
→coffee break!)
Cloning into '/opt/cyverse-ez-singularity'...
```

(continues on next page)

(continued from previous page)

```

remote: Enumerating objects: 24, done.
remote: Total 24 (delta 0), reused 0 (delta 0), pack-reused 24
Unpacking objects: 100% (24/24), done.
* ez singularity or singularity itself may not have updated successfully, but you can
→probably try executing it

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

```

16.2.4 2.4 Check Installation

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

```

$ singularity pull shub://vsoch/hello-world
INFO:   Downloading shub image
      59.75 MiB / 59.75 MiB
→[=====]
→100.00% 49.24 MiB/s 1s
  tswetnam@tysons-box:~$ singularity run hello-world_latest.sif
  RaawwWWWWRRRR!! Avocado!

```

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

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

```

$ singularity
Usage:
  singularity [global options...] <command>

Available Commands:
  build      Build a Singularity image
  cache      Manage the local cache
  capability  Manage Linux capabilities for users and groups
  config     Manage various singularity configuration (root user only)
  delete     Deletes requested image from the library
  exec       Run a command within a container
  inspect    Show metadata for an image
  instance   Manage containers running as services
  key        Manage OpenPGP keys
  oci        Manage OCI containers
  plugin     Manage Singularity plugins
  pull       Pull an image from a URI
  push       Upload image to the provided URI
  remote     Manage singularity remote endpoints
  run        Run the user-defined default command within a container
  run-help   Show the user-defined help for an image
  search     Search a Container Library for images
  shell      Run a shell within a container
  sif        siftool is a program for Singularity Image Format (SIF) file
→manipulation
  sign       Attach a cryptographic signature to an image
  test       Run the user-defined tests within a container
  verify     Verify cryptographic signatures attached to an image

```

(continues on next page)

(continued from previous page)

```
version      Show the version for Singularity

Run 'singularity --help' for more detailed usage information.
```

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

```
$ singularity help pull
Pull an image from a URI

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

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

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

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

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

  oras: Pull a SIF image from a supporting OCI registry
        oras://registry/namespace/image:tag

  http, https: Pull an image using the http(s?) protocol
                https://library.sylabs.io/v1/imagefile/library/default/alpine:latest

Options:
  --arch string      architecture to pull from library (default "amd64")
  --dir string       download images to the specific directory
  --disable-cache    dont use cached images/blobs and dont create them
  --docker-login     login to a Docker Repository interactively
  -F, --force        overwrite an image file if it exists
  -h, --help         help for pull
  --library string   download images from the provided library
                    (default "https://library.sylabs.io")
  --no-cleanup       do NOT clean up bundle after failed build, can be
                    helpul for debugging
  --nohttps          do NOT use HTTPS with the docker:// transport
                    (useful for local docker registries without a
                    certificate)

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

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

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

(continues on next page)

(continued from previous page)

```
From supporting OCI registry (e.g. Azure Container Registry)
$ singularity pull image.sif oras://<username>.azurecr.io/namespace/image:tag
```

For additional [help](#) or support, please visit <https://www.sylabs.io/docs/>

16.3 3. Downloading pre-built images

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

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

Container Registries:

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

16.3.1 3.1 Pulling an image from Singularity Hub

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

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

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

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

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

16.3.2 3.2 Pulling an image from Docker Hub

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

```
$ singularity pull docker://ubuntu:20.04
INFO:      Converting OCI blobs to SIF format
INFO:      Starting build...
Getting image source signatures
Copying blob 8f6b7df711c8 done
Copying blob 0703c52b8763 done
Copying blob 07304348ce1b done
Copying blob 4795dceb8869 done
Copying config 05ac933964 done
Writing manifest to image destination
Storing signatures
2020/03/09 16:14:12 info unpack layer:┐
↪sha256:8f6b7df711c8a4733138390ff2aba1bfeb755bf4736c39c6e4858076c40fb5eb
2020/03/09 16:14:13 info unpack layer:┐
↪sha256:0703c52b8763604318dcbb1730c82ad276a487335ecabde2f43f69a6222e8090
2020/03/09 16:14:13 info unpack layer:┐
↪sha256:07304348ce1b6d24f136a3c4ebaa800297b804937a6942ce9e9fe0dac0b0ca74
2020/03/09 16:14:13 info unpack layer:┐
↪sha256:4795dceb8869bdfa64f3742e1df492e6f31baf9cfc36f1a042a8f981607e99a2
INFO:      Creating SIF file...
INFO:      Build complete: ubuntu_20.04.sif
```

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

16.3.3 3.3 Pulling an image from Sylabs cloud library

Let's use an easy example of `alpine.sif` image from the [container library](#)

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

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

16.4 4 Interact with images

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

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

```
$ singularity pull library://tyson-swetnam/default/cowsay
INFO:      Downloading library image
 67.00 MiB / 67.00 MiB
↪[=====
↪100.00% 5.45 MiB/s 12s
WARNING: unable to verify container: cowsay_latest.sif
WARNING: Skipping container verification

tswetnam@tysons-box:~$ singularity run cowsay_latest.sif

/ Expect a letter from a friend who will \
\ ask a favor of you.                    /
-----
      \      ^__^
       (oo)\_______
          (__)\\       )\/\
              ||----w |
              ||     ||
```

16.4.1 4.1 Shell

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

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

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

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

```
$ Singularity cowsay_latest.sif:~> whoami
tswetnam
```

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

16.4.2 4.2 Executing commands

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

```
$ singularity exec cowsay_latest.sif cowsay container camp rocks

< container camp rocks >
-----
      \      ^__^
       (oo)\_______
          (__)\\       )\/\
```

(continues on next page)

(continued from previous page)

```
  ||----w |
  ||      ||
```

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

16.4.3 4.3 Running a container

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

```
singularity run lolcow_latest.sif

/  You will remember, Watson, how the  \
| dreadful business of the Abernetty  |
| family was first brought to my notice |
| by the depth which the parsley had sunk |
| into the butter upon a hot day.      |
|                                     |
\  -- Sherlock Holmes                  /

-----
      ^__^
      (oo)\_______
      (__)\       )\/\
           ||----w |
           ||     ||
```

Exercise - 1

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

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

Here are the brief steps:

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

16.4.4 4.3 Running a container on HPC

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

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

```
$ module avail singularity
----- /cm/shared/uamodulefiles -----
↪-----
singularity/2/2.6.1 singularity/3/3.2 singularity/3/3.2.1 singularity/3/3.4.2 ↪
↪singularity/3/3.5.3
```

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

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

Advanced Singularity

lsingularityl

17.1 5.0 Building your own Containers from scratch

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

17.2 5.1 Keep 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.sif docker://ubuntu
```

17.2.1 5.2 Building Singularity containers

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

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

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

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

```
$ sudo singularity build --sandbox ubuntu-latest/ docker://ubuntu
$ singularity shell --writable ubuntu-latest/
Singularity ubuntu-latest.sif:~> apt-get update
```

Does it work?

```
$ sudo singularity shell ubuntu-latest.sif
Singularity: Invoking an interactive shell within container...
Singularity ubuntu-latest.sif:~> apt-get update
```

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

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

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

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

17.3 5.2.1: Exercise (~30 minutes): Create a Singularity file

SyLabs User-Guide

A Singularity file can be hosted on Github and will be auto-detected by [Singularity-Hub](#) when you set up your container Collection.

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

- **Header**

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

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

From: selects the tag name.

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

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

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

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

Using a *localimage* to build:

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

Using CentOS-like container:

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

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

The container registries which Singularity uses are listed in the [Introduction Section 3.1](#).

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

The additional sections of a Singularity file include:

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

Setting up Singularity file system

- **Help**

%help section can be as verbose as you want

```
Bootstrap: docker
From: ubuntu

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

- **Setup**

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

- **Files**

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

- **Post**

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

- **Environment**

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

- **Runscript**

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

18.1 Example Singularity file

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

```
BootStrap: docker
From: ubuntu:18.04

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

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

%runscript
    fortune | cowsay | lolcat

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

Build the container:

```
singularity build cowsay.sif Singularity
```

Run the container:

```
singularity run cowsay.sif
```

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

18.2 Cryptographic Security

[Documentation](#)

Singularity and High Performance Computing

High Performance Computing resources fill an important role in research computing and can support container execution through runtimes such as Singularity or, hopefully soon, rootless Docker, among other options.

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

Importantly, most jobs run on an HPC cluster are neither **interactive**, nor **realtime**. When you submit a job to the scheduler, you must tell it what resources you need (e.g. how many nodes, how much RAM, what type of nodes, and for how long) in addition to what you want to run. Then the scheduler finally has resources matching your requirements, it runs the job for you. If your request is very large, or very long, you may never make it out of the queue.

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

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

The container will immediately start.

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

```
#!/bin/bash
#
#SBATCH -J myjob                # Job name
#SBATCH -o output.%j           # Name of stdout output file (%j expands to
↪ jobId)
#SBATCH -p development         # Queue name
#SBATCH -N 1                   # Total number of nodes requested (68 cores/
↪ node)
#SBATCH -n 17                  # Total number of mpi tasks requested
#SBATCH -t 02:00:00            # Run time (hh:mm:ss) - 4 hours

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

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

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

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

19.1 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.

These constraints make HPC systems perfectly suitable for execution environments, but currently a limiting choice for a development environment. We usually recommend your local laptop or a VM as a development environment where you can iterate on your code rapidly and test container building and execution.

19.2 Singularity and MPI

Singularity supports MPI fairly well. Since (by default) the network is the same inside and outside the container, the communication between containers usually just works. The more complicated bit is making sure that the container has the right set of MPI libraries. MPI is an open specification, but there are several implementations (OpenMPI, MVAPICH2, and Intel MPI to name three) with some non-overlapping feature sets. If the host and container are running different MPI implementations, or even different versions of the same implementation, hilarity may ensue.

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

Note: Many HPC Systems, like Stampede2 at TACC and Ocelote at UAHP, have high-speed, low-latency networks that have special drivers. Infiniband, Ares, and OmniPath are three different specs for these types of networks. When running MPI jobs, if the container doesn’t have the right libraries, it won’t be able to use those special interconnects to communicate between nodes.

19.3 Base Docker images

Depending on the system you will use, you may have to build your own MPI enabled Singularity images (to get the versions to match).

When running at TACC, there is a set of curated Docker images for use in the FROM line of your own containers. You can see a list of available images at <https://hub.docker.com/u/tacc>

Specifically, you can use the `tacc/tacc-ubuntu18-mvapich2.3-psm2` image to satisfy the MPI architecture and version requirements for running on Stampede2.

Because you may have to build your own MPI enabled Singularity images (to get the versions to match), here is a 3.1 compatible example of what it may look like:

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

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

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

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

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

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

19.4 Example Containerized MPI App

In your Docker development environment, make a new directory in which to build up a new image and download (or copy and paste) two files in that directory:

https://raw.githubusercontent.com/TACC/containers_at_tacc/master/docs/scripts/Dockerfile.mpi

https://raw.githubusercontent.com/TACC/containers_at_tacc/master/docs/scripts/pi-mpi.py

Take a look at both files. `pi-mpi.py` is a simple MPI Python script that approximates pi (very inefficiently) by random sampling. `Dockerfile.mpi` is an updated Dockerfile that uses the TACC base image to satisfy all the MPI requirements on Stampede2.

Next, try building the new container.

```
$ docker build -t USERNAME/pi-estimator:0.1-mpi -f Dockerfile.mpi .
```

Don't forget to change USERNAME to your DockerHub username.

Once you have successfully built an image, push it up to DockerHub with the `docker push` command so that we can pull it back down on Stampede2.

19.5 Running an MPI Container on Stampede2

To test, we can grab an interactive session that has two nodes. That way we can see if we can make the two nodes work together. On TACC systems, the "idev" command will start an interactive session on a compute node:

```
$ idev -m 60 -p normal -N 2 -n 128
```

Once you have nodes at your disposal and a container on DockerHub, invoking it would look something like:

```
module load tacc-singularity
cd $WORK
singularity pull docker://USERNAME/pi-estimator:0.1-mpi
time singularity exec pi-estimator_0.1-mpi.sif pi-mpi.py 10000000
time ibrun singularity exec pi-estimator_0.1-mpi.sif pi-mpi.py 10000000
```

Note: TACC uses a command called `ibrun` on all of its systems that configures MPI to use the high-speed, low-latency network. If you are familiar with MPI, this is the functional equivalent to `mpirun`

The first `singularity exec pi-estimator_0.1-mpi.sif pi-mpi.py 10000000` command will use 1 CPU core to sample ten million times. The second command, using `ibrun` will run 128 processes that sample ten million times each and pass their results back to the “rank 0” MPI process to merge the results.

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

As an aside, for a single node you can also use the **container MPI** to run in parallel (but usually you don’t want this).

When you are done with your interactive session, don’t forget to `exit` to end the session and go back to the login node.

19.6 Singularity and GPU Computing

GPU support in Singularity is very good.

Since Singularity supported docker containers, it has been fairly simple to utilize GPUs for machine learning code like TensorFlow. We will not do this as a hands-on exercise, but in general the procedure is as follows.

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

# Pull your image
singularity pull docker://nvidia/caffe:latest

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

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

```
# this is missing the --nv flag and will not work
singularity exec caffe-latest.sif caffe device_query -gpu 0
```

The main requirement for GPU containers to work is that the version of the host drivers matches the major version of the library inside the container. So, for example, if CUDA 10 is on the host, the container needs to use CUDA 10 internally.

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

```
# Change to your $WORK directory
cd $WORK
#Get the software
git clone https://github.com/tensorflow/models.git ~/models
# Pull the image
singularity pull docker://tensorflow/tensorflow:latest-gpu
```

(continues on next page)

(continued from previous page)

```
# Run the code
singularity exec --nv tensorflow-latest-gpu.sif python $HOME/models/tutorials/image/
↪mnist/convolutional.py
```

The University of Arizona HPS [Singularity](#) examples.



 [Learning Center Home](#)

BioContainers

BioContainers is a community-driven project that provides the infrastructure and basic guidelines to create, manage and distribute bioinformatics containers with **special focus in proteomics, genomics, transcriptomics and metabolomics**. BioContainers is based on the popular frameworks of Docker.



BIOCONTAINERS

BioContainers Goals:

- Provide a base specification and images to easily build and deploy new bioinformatics/proteomics software including the source and examples.
- Provide a series of containers ready to be used by the bioinformatics community (<https://github.com/BioContainers/containers>).
- Define a set of guidelines and specifications to build a standardized container that can be used in combination with other containers and bioinformatics tools.
- Define a complete infrastructure to develop, deploy and test new bioinformatics containers using continuous integration suites such as Travis Continuous Integration (<https://travis-ci.org/>), Shippable (<https://app.shippable.com/>) or manually built solutions.
- Provide support and help to the bioinformatics community to deploy new containers for researchers that do not have bioinformatics support.
- Provide guidelines and help on how to create reproducible pipelines by defining, reusing and reporting specific container versions which will consistently produce the exact same result and always be available in the history of the container.
- Coordinate and integrate developers and bioinformaticians to produce best practice of documentation and software development.

20.1 Introduction to Bioconda



Bioconda is a channel for the conda package manager specializing in bioinformatics software. It consists of:

- A repository of recipes hosted on [GitHub](#)
- A build system that turns these recipes into conda packages
- A repository of > 6000 bioinformatics packages ready to use with a simple conda install command
- **Each package added to Bioconda also has a corresponding Docker BioContainer automatically created and uploaded to Quay.io**
- Over 600 contributors that add, modify, update and maintain the recipes

Note: Recipe vs package A **recipe** is a directory containing a small set of files that defines name, version, dependencies, and URL for source code. A recipe typically contains a meta.yaml file that defines these settings and a build.sh script that builds the software. A recipe is converted into a package by running “conda-build” on the recipe. A **package** is a bgzipped tar file (.tar.bz2) that contains the built software. Packages are uploaded to anaconda.org so that users can install them with “conda install” command.

You can [contribute](#) to the Bioconda project by building your own packages. Each package will also be made available as a BioContainer at [Quay.io](#).

20.2 Glossary

- **Image:** self-contained, read-only ‘snapshot’ of your applications and packages, with all their dependencies
- **Container:** a running instance of your image
- **Image registry:** a storage and content delivery system, holding named images, available in different tagged versions
- **Docker:** a program that runs and handles life-cycle of containers and images
- **CyVerse tool:** Software program that is integrated into the back end of the DE for use in DE apps
- **CyVerse app:** graphic interface of a tool made available for use in the DE

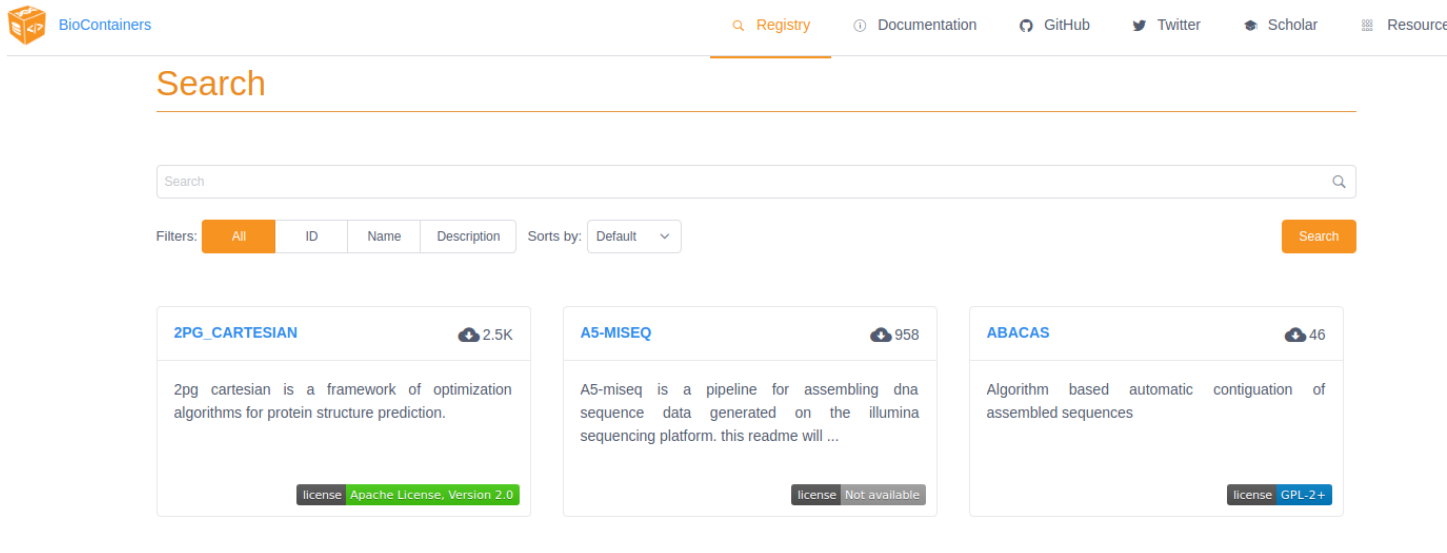
20.3 Where to Get a BioContainer

Images are made publicly available through image registries. There are several different image registries that provide access to BioContainers. The three major registries are detailed here.

20.3.1 The BioContainers Registry

[BioContainers Registry](#) UI provides the interface to search, tag, and document BioContainers across all the registries. Which means that if a BioContainer exists you can find it here.

To find the tool you want to use just search for it by name in the search box at the top of the registry page. The BioContainers registry returns partial matches and matches to the tool description. So, if you want to find all the tools relevant to Nanopore analysis you can search for ‘nanopore’.



2PG_CARTESIAN 2.5K

2pg cartesian is a framework of optimization algorithms for protein structure prediction.

license Apache License, Version 2.0

A5-MISEQ 958

A5-miseq is a pipeline for assembling dna sequence data generated on the illumina sequencing platform. this readme will ...

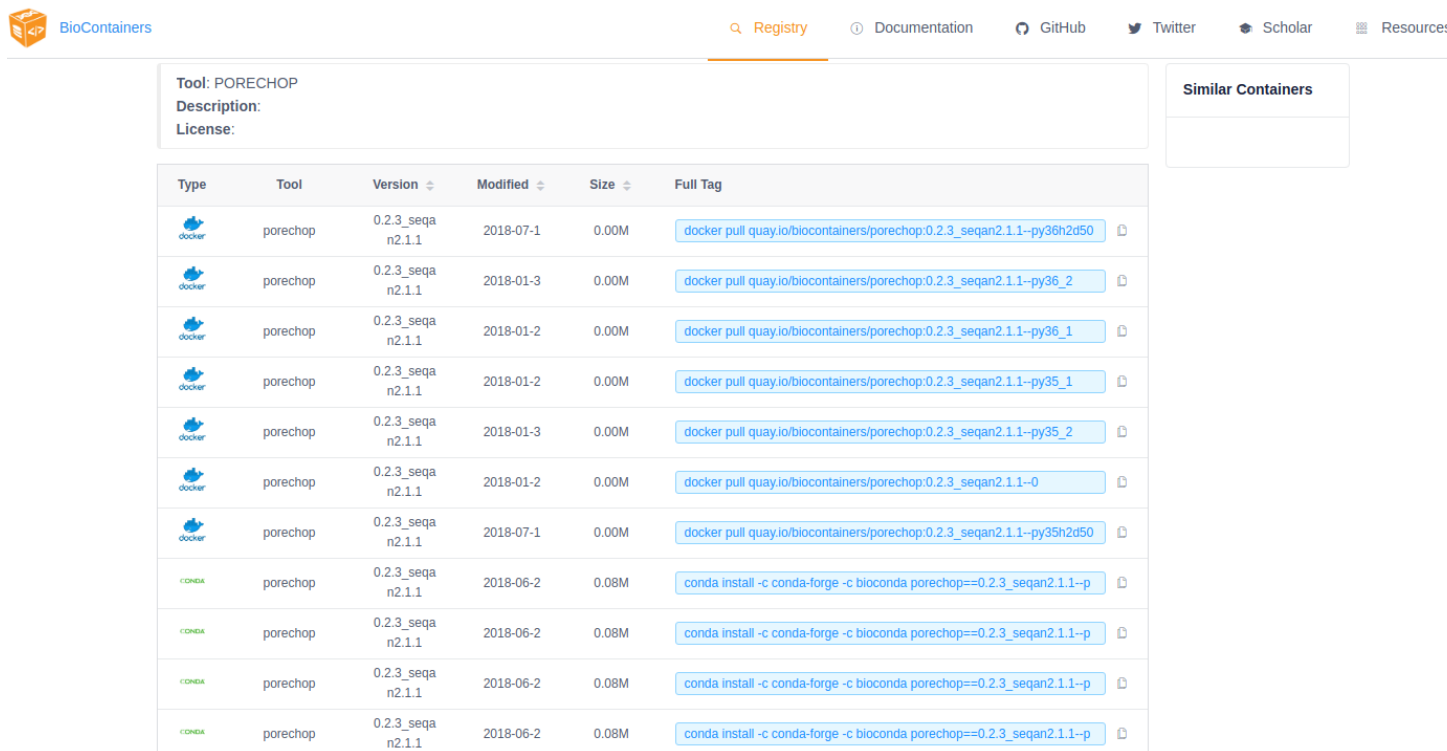
license Not available

ABACAS 46

Algorithm based automatic contiguation of assembled sequences

license GPL-2+

If the tool you are looking for is already available as a BioContainer click on that tile in the search results. This will display all the available BioContainers and Conda packages for this tool (ie. different versions of the tool). Choose the version of the tool you want to use (when in doubt, choose the most recent version). Select the icon at the right to copy the ‘docker pull’ command for that version.














Tool: PORECHOP

Description:

License:

Similar Containers

Type	Tool	Version	Modified	Size	Full Tag
	porechop	0.2.3_seqa_n2.1.1	2018-07-1	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-py36h2d50
	porechop	0.2.3_seqa_n2.1.1	2018-01-3	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-py36_2
	porechop	0.2.3_seqa_n2.1.1	2018-01-2	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-py36_1
	porechop	0.2.3_seqa_n2.1.1	2018-01-2	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-py35_1
	porechop	0.2.3_seqa_n2.1.1	2018-01-3	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-py35_2
	porechop	0.2.3_seqa_n2.1.1	2018-01-2	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-0
	porechop	0.2.3_seqa_n2.1.1	2018-07-1	0.00M	docker pull quay.io/biocontainers/porechop:0.2.3_seqa_n2.1.1-py35h2d50
	porechop	0.2.3_seqa_n2.1.1	2018-06-2	0.08M	conda install -c conda-forge -c bioconda porechop==0.2.3_seqa_n2.1.1-p
	porechop	0.2.3_seqa_n2.1.1	2018-06-2	0.08M	conda install -c conda-forge -c bioconda porechop==0.2.3_seqa_n2.1.1-p
	porechop	0.2.3_seqa_n2.1.1	2018-06-2	0.08M	conda install -c conda-forge -c bioconda porechop==0.2.3_seqa_n2.1.1-p
	porechop	0.2.3_seqa_n2.1.1	2018-06-2	0.08M	conda install -c conda-forge -c bioconda porechop==0.2.3_seqa_n2.1.1-p

Note: You want the docker images, not the Conda packages. Conda packages are not containers.

Note: If your tool is not already available as a BioContainer (ie. your search returned nothing) proceed to the [How to Request a BioContainer](#) or [How to Build a BioContainer](#) section below.

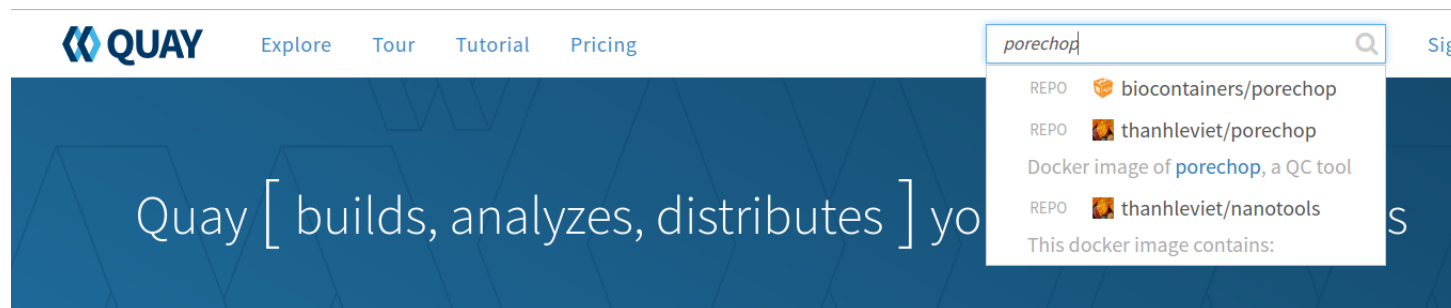
20.3.2 Quay

Quay is another image registry. Unlike the BioContainers Registry, Quay.io is not specific to BioContainers. Anyone (including you) can create an account at Quay.io and host your own images but **an account is not necessary to use BioContainers (or other publicly available images)**.

Although anyone can create a BioContainer, the majority of BioContainers are created by the Bioconda project. Every Bioconda package has a corresponding BioContainer available at Quay.io. From the Quay.io page search for the tool you want by name.

Note: The Quay.io search will only find those tools with an exact match of the name (unlike the BioContainers Registry).

Important: Other users may also have images available that contain your tool. Be sure to choose the image that is part of the ‘biocontainers’ organization. BioContainers is a trusted source and you know what you’re getting.



Note: If your search yields no results then double-check by searching the BioContainers Registry (just to be sure). If your tool isn’t available as a BioContainer then proceed to the [How to request a BioContainer](#) or [How to build a BioContainer](#) section below.

From the repo page, choose the ‘tags’ tab on the left side of the screen and you will get a list of the available images. Unlike the BioContainers Registry, Quay.io will not display conda packages in the list. Again, when in doubt choose the most recent version available for your tool. Click on the ‘fetch tag’ icon to the right of your chosen version. Then select ‘Docker pull (by tag)’ from the drop-down and copy the ‘docker pull’ command.

The screenshot shows the Quay.io web interface. At the top, there's a navigation bar with 'QUAY' logo and links for 'Explore', 'Tour', 'Tutorial', and 'Pricing'. A search bar is on the right. Below the navigation bar, the page title is 'biocontainers / porechop'. The main content area is titled 'Repository Tags'. It features a table with columns: TAG, LAST MODIFIED, SECURITY SCAN, SIZE, EXPIRES, and MANIFEST. The table lists several tags, all marked as 'Unsupported' for security scans. A 'Fetch Tag' button is visible in the top right of the table. A modal window is open, titled 'Fetch Tag: 0.2.3_seqan2.1.1--py36h2d50403_3'. It has an 'Image Format' dropdown menu. A secondary dropdown menu is open, showing options: 'Docker Pull (by tag)', 'Docker Pull (by digest)', 'Squashed Docker Image', and 'rkt Fetch'.

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
<input type="checkbox"/> 0.2.3_seqan2.1.1--py36h2d50403_3	7 months ago	Unsupported	50.1 MB	Never	SHA256 65f1cbe96399
<input type="checkbox"/> 0.2.3_seqan2.1.1--py35h2d50403_3	7 months ago	Unsupported	49.3 MB	Never	SHA256 8ea41e99346a
<input type="checkbox"/> 0.2.3_seqan2.1.1--py35_2	a year ago	Unsupported	47.8 MB	Never	SHA256 5c0403963e69
<input type="checkbox"/> 0.2.3_seqan2.1.1--py36_2	a year ago	Unsupported	48.4 MB	Never	SHA256 3fb07c7b2451
<input type="checkbox"/> 0.2.3_seqan2.1.1--py36_1	a year ago	Unsupported	48.4 MB	Never	SHA256 1e3f4794cc59
<input type="checkbox"/> 0.2.3_seqan2.1.1--py35_1	a year ago	Unsupported	47.8 MB	Never	SHA256 22504bb6deae
<input type="checkbox"/> 0.2.3_seqan2.1.1--0	a year ago	Unsupported	48.4 MB	Never	SHA256 4bf6abaebe85

20.3.3 DockerHub

DockerHub is the most well-known and popular image registry for Docker containers. Like Quay.io, you can create an account at DockerHub and host your own images but **an account is not necessary to use BioContainers (or other publicly available images)**.

There are fewer BioContainers images available at DockerHub than the other two registries. You can see them all by searching for 'biocontainers' in the search bar of the DockerHub page.

The screenshot shows the Docker Hub interface with a search for 'biocontainer'. The top navigation bar includes 'docker hub', a search bar with 'biocontainer', and links for 'Explore', 'Repositories', 'Organizations', 'Get Help', and a user profile 'amcooksey'. Below the navigation bar, there are tabs for 'Docker EE', 'Docker CE', 'Containers' (selected), and 'Plugins'. The main content area shows search results for 'biocontainer' with 1 - 25 of 905 results. On the left, there are filters for 'Docker Certified', 'Images' (Verified Publisher, Official Images), and 'Categories' (Analytics, Application Frameworks, Application Infrastructure, Application Services). The search results list two items: 'biocontainers/biocontainers' (10K+ Downloads, 5 Stars) and 'biocontainers/samtools' (50K+ Downloads, 5 Stars). Both items are by 'biocontainers' and updated 2 and 4 months ago respectively. The 'biocontainers/biocontainers' item is described as 'Biocontainers base Image' and the 'biocontainers/samtools' item is described as 'Tools for manipulating next-generation sequencing data'. Both items have tags for 'Container', 'Linux', and 'x86-64'.

Note: You can also search for the name of the tool you want. Be sure that you choose images that belong to the BioContainers organization. There will be many other options available on DockerHub. BioContainers is a trusted source.

The second image in this search results list is 'vcftools'. Select 'vcftools' and you will see the repo page for this tool. The 'docker pull' command can be copied from the overview page; however, there is no tag specified. To see the available versions, select the tags tab at the top of the page. You will need to supply the tag of the version you want following a colon at the end of your docker pull command to get a specific version.

```
$ docker pull biocontainers/vcftools:v0.1.14_cv2
```

While DockerHub offers fewer BioContainers than the other registries it does offer some advantages for those who want to build their own BioContainers.

- The first image in the search results for 'biocontainers' is the 'biocontainers base image'. This image can be built upon if you wish to build your own BioContainers.
- Dockerfiles are available for these containers so you can see exactly how they were built.

For more information on building your own BioContainer see [How to build a BioContainer](#) section below.

20.4 How to Request a BioContainer

If the tool you want isn't available as a BioContainer you can request that one be built for you. Users can request a container by opening an issue in the [containers repository](#)

BioContainers / containers

Watch 27 Star 295 Fork 11

Code Issues 36 Pull requests 5 ZenHub Projects 0 Insights

Want to contribute to BioContainers/containers? If you have a bug or an idea, browse the open issues before opening a new one. You can also take a look at the [Open Source Guide](#).

Filters is:open is:issue label:"Container Request" Labels 13 Milestones 1 New issue

Clear current search query, filters, and sorts

18 Open 46 Closed Author Labels Projects Milestones Assignee Sort

Container request: breseq Container Request #227 opened on May 3, 2018 by yeemey

PAUP Container Request #164 opened on Jun 28, 2017 by andzandz11

phylobayes Container Request #162 opened on Jun 28, 2017 by andzandz11

The issue should contain:

- the name of the software
- the url of the code or binary to be packaged
- information about the software
- tag the issue with the 'Container Request' label

When the container is deployed and fully functional, the issue will be closed by the developer or the contributor to BioContainers. When a container is deployed and the developer closes the issue in GitHub the user receives a notification that the container is ready. You can find your container at Quay.io and use the 'docker pull' command to run it as you would any other container.

20.5 How to Use a BioContainer

To run your BioContainer you will need a computer with Docker installed.

20.5.1 How to Install Docker

Installing Docker on your computer takes a little time but it is reasonably straight forward and it is a one-time setup. [Docker can be installed by following these directions.](#)

Docker installation is much easier on an Atmosphere instance with the 'ezd' command.

```
$ ezd
```

20.5.2 Get Data to Use with Your Container

Set up iCommands.

```
$ cd Desktop
$ iget /iplant/home/shared/iplantcollaborative/example_data/porechop/SRR6059710.fastq
```

20.5.3 Use ‘docker pull’ to Get the Image

First, you will need to pull the image from the registry onto your computer. Use the ‘docker pull’ command you copied from the registry above (*Where to Get a BioContainer*).

```
$ docker pull quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
```

Note: If you are working on a system for which you don’t have root permissions you will need to use ‘sudo’ and provide your password. Like this:

```
$ sudo docker pull quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
```

```
[amcooksey@rogue ~]$ sudo docker pull quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
[sudo] password for amcooksey:
0.2.3_seqan2.1.1--py36h2d50403_3: Pulling from biocontainers/porechop
a3ed95caeb02: Already exists
b0dc45cd432d: Already exists
9466b3513669: Already exists
ddd482ea7b54: Already exists
4d69f833b9d8: Already exists
e7c454e5167d: Already exists
e38092b005c0: Already exists
f879b42dfe2b: Already exists
9417599398f7: Pull complete
Digest: sha256:65f1cbe96399eff89df55169f25d2b52f46115f9d4080c388fdeb7b22dc76b30
Status: Downloaded newer image for quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
```

20.5.4 Use the ‘docker run’ Command to Run the Container

The easiest way to test that the container will run is to run the help command for the tool. In this case ‘-h’ is the help command.

```
sudo docker run --rm -v $(pwd):/working-dir -w /working-dir --entrypoint="porechop" \
↳ quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3 -h
```

From the result we are able to see the only required option is ‘-i INPUT’. Options in [square brackets] are not required.

Now we can run the container with our data file to see the output.

```
sudo docker run --rm -v $(pwd):/working-dir -w /working-dir --entrypoint="porechop" \
↳ quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3 -i SRR6059710.fastq \
↳ -o porechop_output.fastq
```

(continues on next page)

(continued from previous page)

We can break the command down into pieces so it is easier to read (the backslash represents where we have broken the line).

```
sudo \
docker run \
--rm \
-v $(pwd):/working-dir \
-w /working-dir \
--entrypoint="porechop" \
quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3 \
-i SRR6059710.fastq \
-o porechop_out.fastq
```

20.5.5 What it All Means

- ‘sudo’ allows you to run the container with ‘root’ permissions—only required if you don’t have root permissions on your machine
- ‘docker run’ tells docker to run the container
- ‘--rm’ removes the container (not the image) from your system when the analysis is complete
- ‘-v’ mounts a *local* directory into a directory *within the container*
- ‘-w’ specifies the working directory within the container
- ‘--entrypoint’ tells the container what to do (usually the name of the tool; the command you would use to run the tool on the command line)
- ‘quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3’ is the name of the image we pulled from Quay.io
- ‘-i’ is the argument for the input file (FASTQ) for Porechop
- ‘-o’ is the argument for the output file (trimmed FASTQ) for Porechop

Important: You must supply an entrypoint on the command line when you run a BioContainer. It is possible to build entrypoints into a container but that is not the case with BioContainers.

```
Loading reads
SRR6059710.fastq
543,374 reads loaded

Looking for known adapter sets
4,990 / 10,000 (49.9%)
```

Trimming adapters from read ends

```

      SQK-NSK007_Y_Top: AATGTACTTCGTTACGTATTGCT
      SQK-NSK007_Y_Bottom: GCAATACGTAACGAAGT
      SQK-MAP006_Y_Top_SK63: GGTTGTTTCTGTTGGTGCTGATATTGCT
      SQK-MAP006_Y_Bottom_SK64: GCAATATCAGCACCAACAGAAA
      PCR_1_start: ACTTGCCTGTCGCTCTATCTTC
      PCR_1_end: GAAGATAGAGCGACAGGCAAGT
      PCR_tail_1_start: TTAACCTTTCTGTTGGTGCTGATATTGC
      PCR_tail_1_end: GCAATATCAGCACCAACAGAAAGGTTAA
      PCR_tail_2_start: TTAACCTACTTGCCTGTCGCTCTATCTTC
      PCR_tail_2_end: GAAGATAGAGCGACAGGCAAGTAGGTTAA

```

No adapters found - output reads are unchanged from input reads

Saving trimmed reads to file

Saved result to /working-dir/porechop_out.fastq

The output from Porechop is saved into the working directory within the container. We ran the container we mounted our current *local* working directory into the working directory *within the container*. The analysis has finished, the container has been removed (remember `-rm`) and now we should find our outputs in our *local* current working directory.

List the files:

```
$ ls -l
```

```

[amcooksey@rogue racon]$ ls -l
total 11350140
-rw-r----- 1 amcooksey iplant-everyone 346188054 May  8  2018 concat_reads.fastq
-rw-r----- 1 amcooksey iplant-everyone   23424 May  8  2018 miniasm_cat_output.fas
-rw-r----- 1 amcooksey iplant-everyone   11745 May  8  2018 minimap_cat_rnd2_out.p
-rw-r--r-- 1 root      root             838803132 Feb 14 12:23 porechop_out.fastq
-rw-r--r-- 1 amcooksey iplant-everyone 9579801472 May 14  2018 SRR6059708.fastq
-rw-r--r-- 1 amcooksey iplant-everyone 857704006 May 14  2018 SRR6059710.fastq

```

You can see the ‘porechop_out.fastq’ file is in our current working directory. Notice that the this file is owned by ‘root’. This is because Docker containers always run as ‘root’.

At this point you can run your container on any system with Docker installed. To use this container on an HPC system you will need to use Singularity (rather than Docker) to run your container. For more information about running Docker containers with Singularity see the [Singularity documentation](#)

Note: Reporting a problem with a container: If you find a problem with a BioContainer an issue should be opened in the [containers repository](#), you should use the ‘broken’ tag (see tags). Developers of the project will pick-up the issue and deploy a new version of the container. A message will be delivered when the container has been fixed.

20.6 How to Build a BioContainer

For more information on building Bioconda BioContainers see the [Bioconda documentation](#)

For more information on building Docker BioContainers see [BioContainers contribution guidelines](#).

20.7 Useful Links

- [BioContainers](#)
- [Bioconda](#)
- [Bioconda GitHub](#)
- [Quay.io BioContainers organization](#)
- [BioContainers Registry](#)
- [DockerHub](#)
- [Request a BioContainer](#)
- [Singularity documentation](#)
- [BioContainers contribution guidelines](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-

Containerized Workflows

21.1 Workflow Management Using Snakemake

Snakemake

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

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

See Snakemake Slides [here](#) and [pdf](#).

CHAPTER 22

SETUP

- Right-Click the button below and login to CyVerse Discovery Environment for a quick launch of Snakemake VICE Jupyter lab app.

- To run Snakemake inside a docker container, run the following on your instance with docker installed:

```
docker run -it --entrypoint bash cyversevice/jupyterlab-snakemake
```

- Click [here](#) for a Snakemake tutorial by [NBISweden](#).
- Clone RNAseq Snakemake tutorial repository

```
git clone https://github.com/NBISweden/workshop-reproducible-research.git
cd workshop-reproducible-research/docker/
git checkout devel
ls
```

- Dry-Run RNAseq Snakefile

```
snakemake -n
```

- Run RNAseq Snakefile

```
snakemake
```

Why Snakemake

From where and how to get data for your analysis, to where and how to treat the outputs, workflow managers can help you achieve better scientific reproducibility and scalability. Once you learn to properly use Snakemake (or similar workflow management tools), keeping track of and sharing your work becomes second nature, not only saving you time whenever you need to re-run all or part of an analysis but helping you reduce errors that naturally get introduced whenever a non-automated activity is done (i.e., as part of the human condition of doing computational science and not being a bot!).

Other Workflow Managers

- CCTools offers [Makeflow](#) a workflow management system similar to Snakemake and also [WorkQueue](#) for scaling-up through Distributed Computing for customized and efficient utilization of resources. Read more [here](#).

Docker for Data Science

For domain scientists (and budding data scientists), running a container 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.

Discussion Question

Why Set Up a Data Science Software Environment in a Container?

answers

- Speed. Docker containers allow a Jupyter or RStudio session to launch in seconds to minutes
 - Configuring environments can be a pain.
 - Standardize how data scientists work, and ensure that old code doesn't stop running because of environment changes.
 - Containerization benefits both data science and IT/technical operations teams.
 - Containers solve a lot of common problems associated with doing data science work at the enterprise level.
-

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 of Docker users, there are readily available images for the common components in data science pipelines.

Here are some Docker Images that may help you quickly configure your own data science pipeline:

- [MySQL](#)
- [Postgres](#)
- [Redmine](#)

- MongoDB
- Hadoop
- Spark
- Zookeeper
- Kafka
- Cassandra
- Storm
- Flink
- R

How does all this stuff fit together??

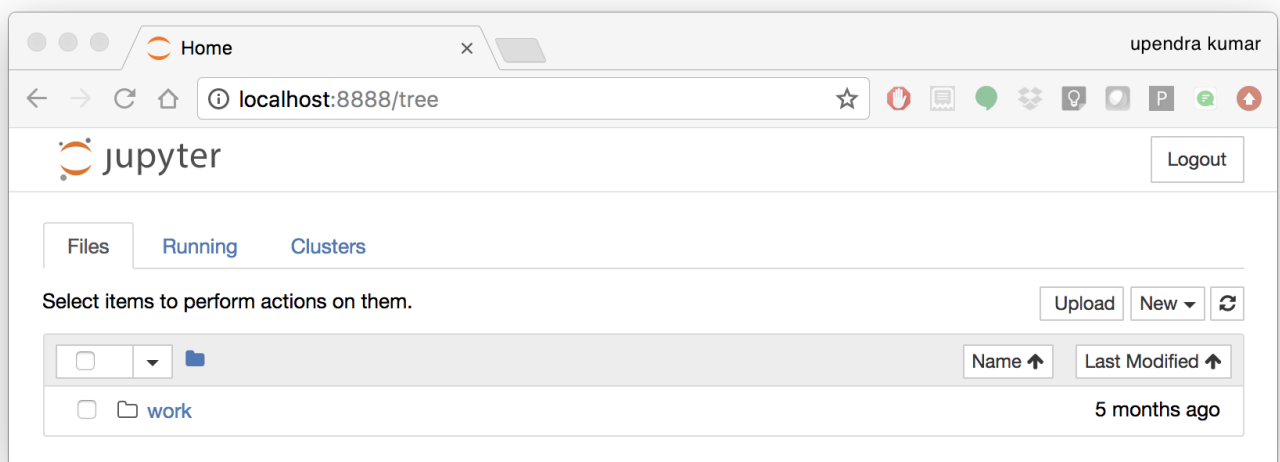
Motivation: Say you want to play around with some cool data science libraries in Python or R but what you don't want to do is spend hours on installing Python or R, working out what libraries you need, installing each and every one and then messing around with the tedium of getting things to work just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

25.1 1. Launch a Jupyter notebook container

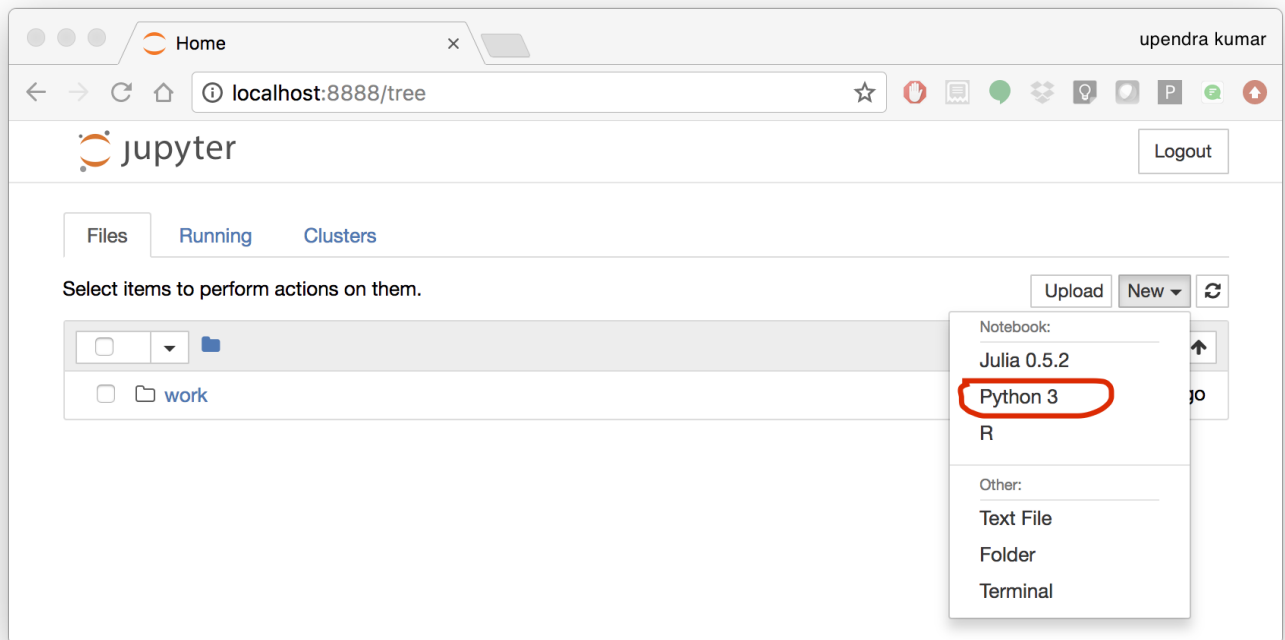
Docker allows us to run a 'ready to go' Jupyter data science stack in what's known as a container:

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

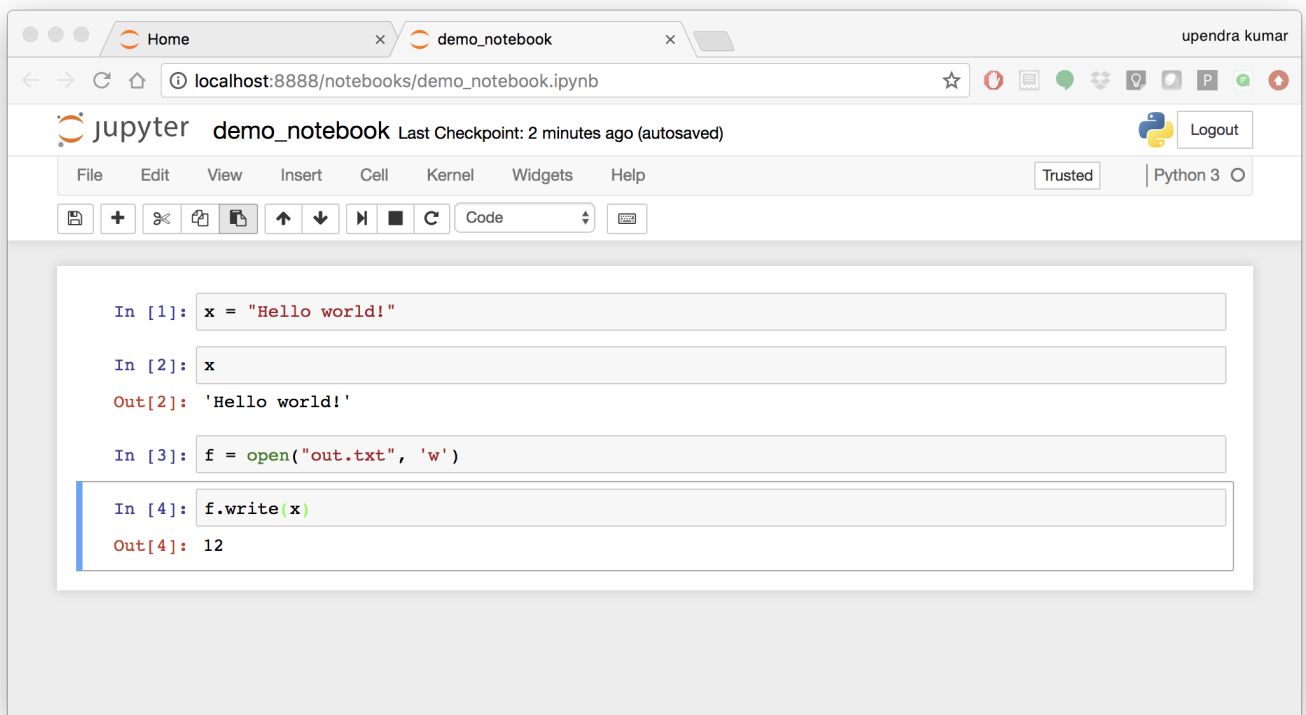
Once you've done that you should be greeted by your very own containerised Jupyter service!

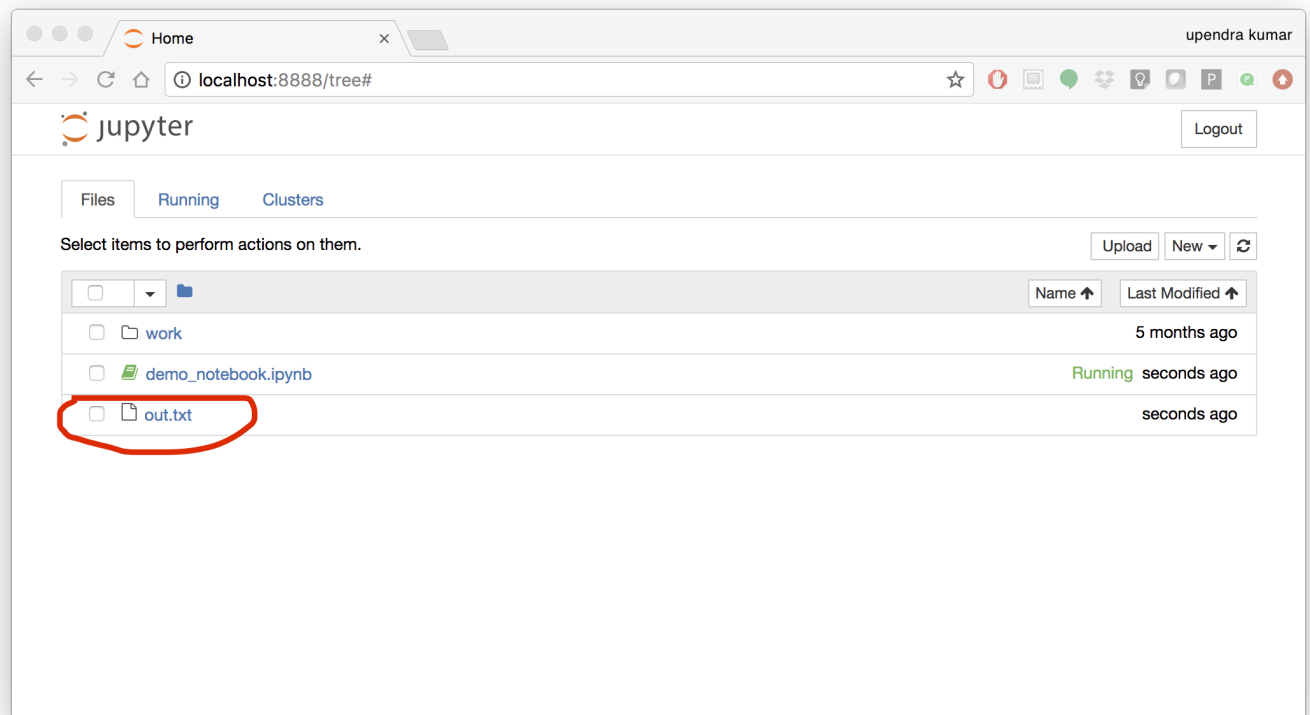


To create your first notebook, drill into the work directory and then click on the 'New' button on the right hand side and choose 'Python 3' to create a new Python 3 based Notebook.



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





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

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

you can run the command as below

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

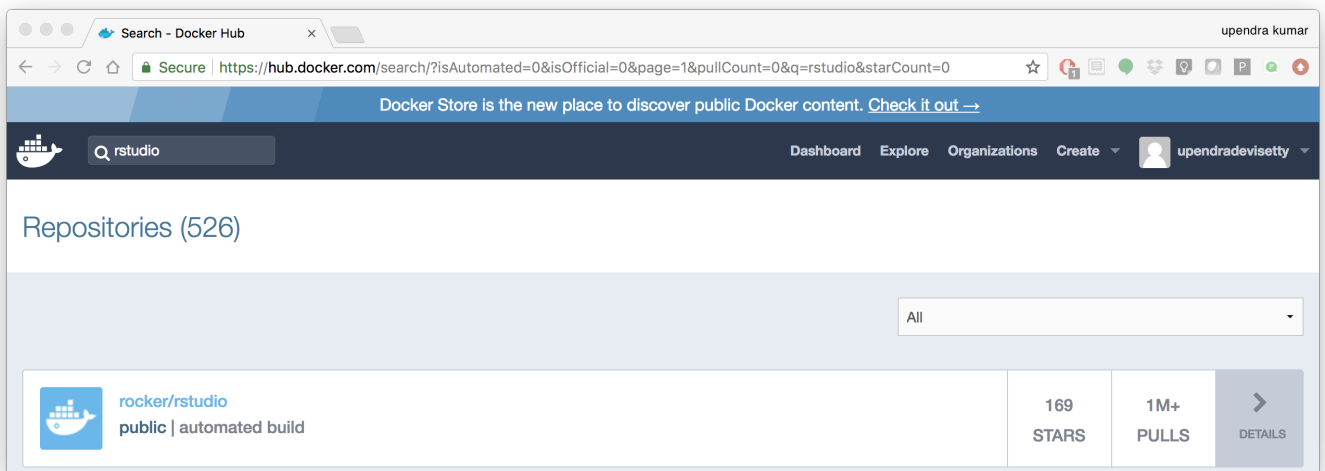
Tip: If you want to run *Jupyter-lab* instead of the default Jupyter notebook, you can do so by adding *jupyter-lab* at the end of the command.

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

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

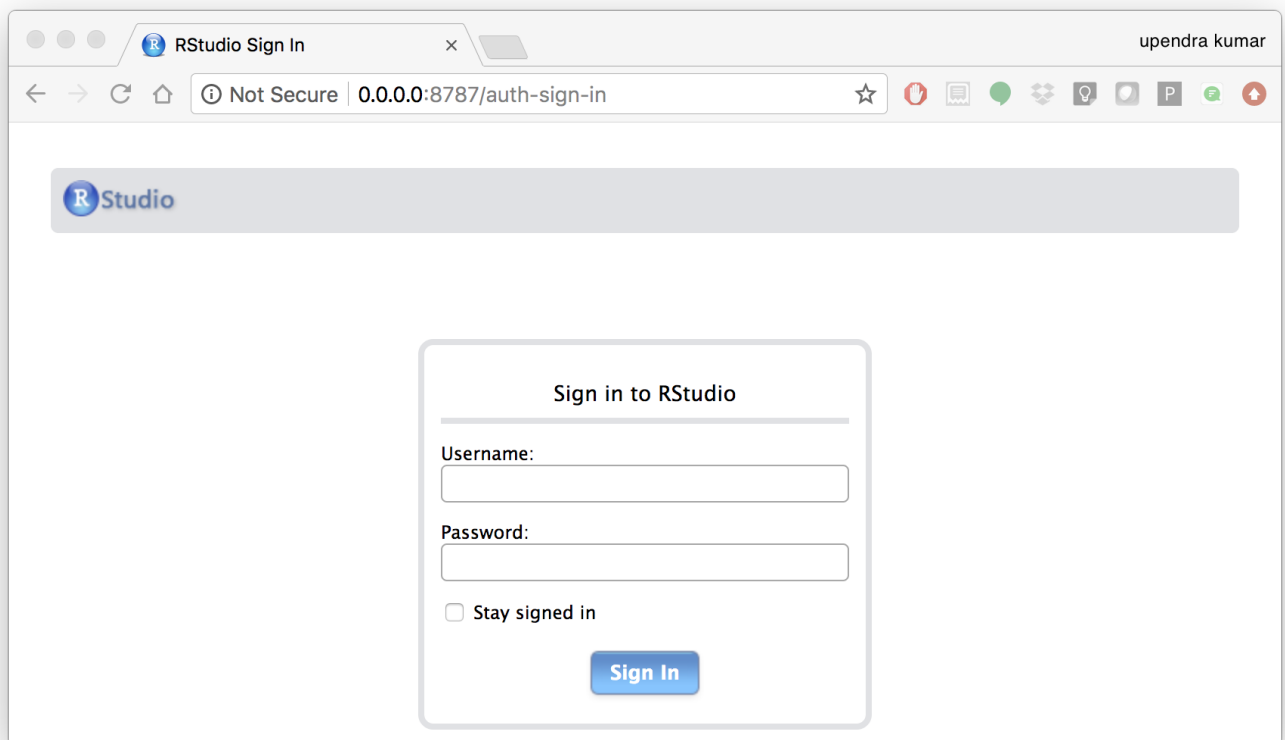
25.2 2. Launch a RStudio container

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



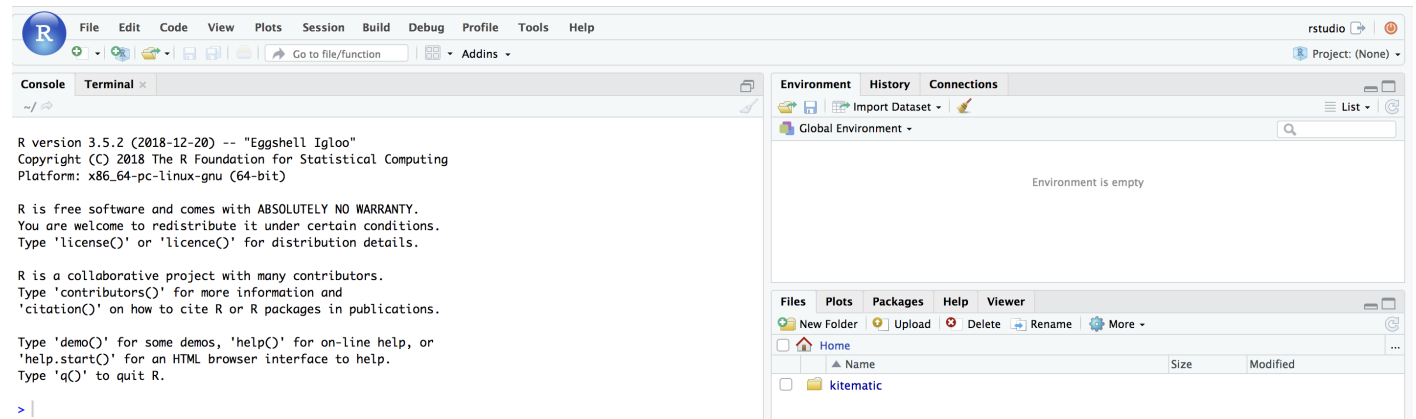
```
$ docker run --rm -d -e PASSWORD=rstudio1 -p 8787:8787 rocker/rstudio
```

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



Tip: For the current Rstudio container, the default username is *rstudio* and the password is *rstudio1*. However you

can override the disable the log-in with `-e DISABLE_AUTH=true` in place of `-e PASSWORD=rstudio1`.



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

```
$ docker run -v $PWD:/data -w /data -p 8787:8787 -e DISABLE_AUTH=true --rm rocker/
↪rstudio:3.6.2
```

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

An excellent R tutorial for reproducible research can be found [here](#)



[Home Icon](#) Learning Center Home

Booting a CyVerse Atmosphere instance

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

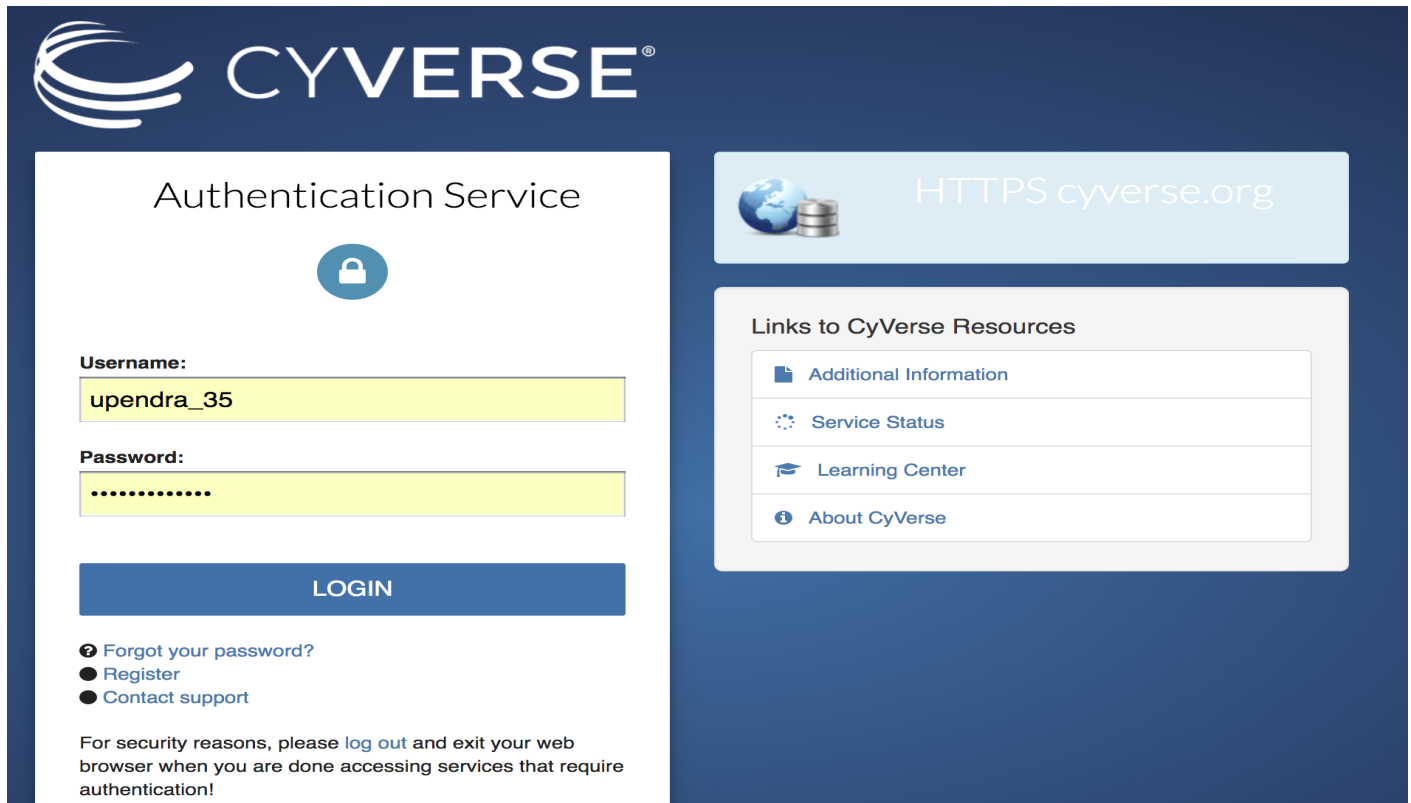
Below, we’ve provided screenshots of the whole process. You can click on them to zoom in a bit. The important areas to fill in are highlighted.

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

Important: You will need to have access to the Atmosphere workshop cloud. If you are unable to log-in for some reason, please let us know and we will fix it immediately.

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

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



The screenshot shows the CyVerse Authentication Service login interface. At the top left is the CyVerse logo. The main heading is "Authentication Service" with a lock icon below it. There are two input fields: "Username:" with the value "upendra_35" and "Password:" with masked characters. A blue "LOGIN" button is below the password field. To the right of the login form is a sidebar with a URL "HTTPS cyverse.org" and a list of links: "Additional Information", "Service Status", "Learning Center", and "About CyVerse". At the bottom left of the login form, there are links for "Forgot your password?", "Register", and "Contact support". A security notice at the bottom of the login form states: "For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!"

CYVERSE®

Authentication Service

Username:

upendra_35

Password:

.....

LOGIN

[Forgot your password?](#)

[Register](#)

[Contact support](#)

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

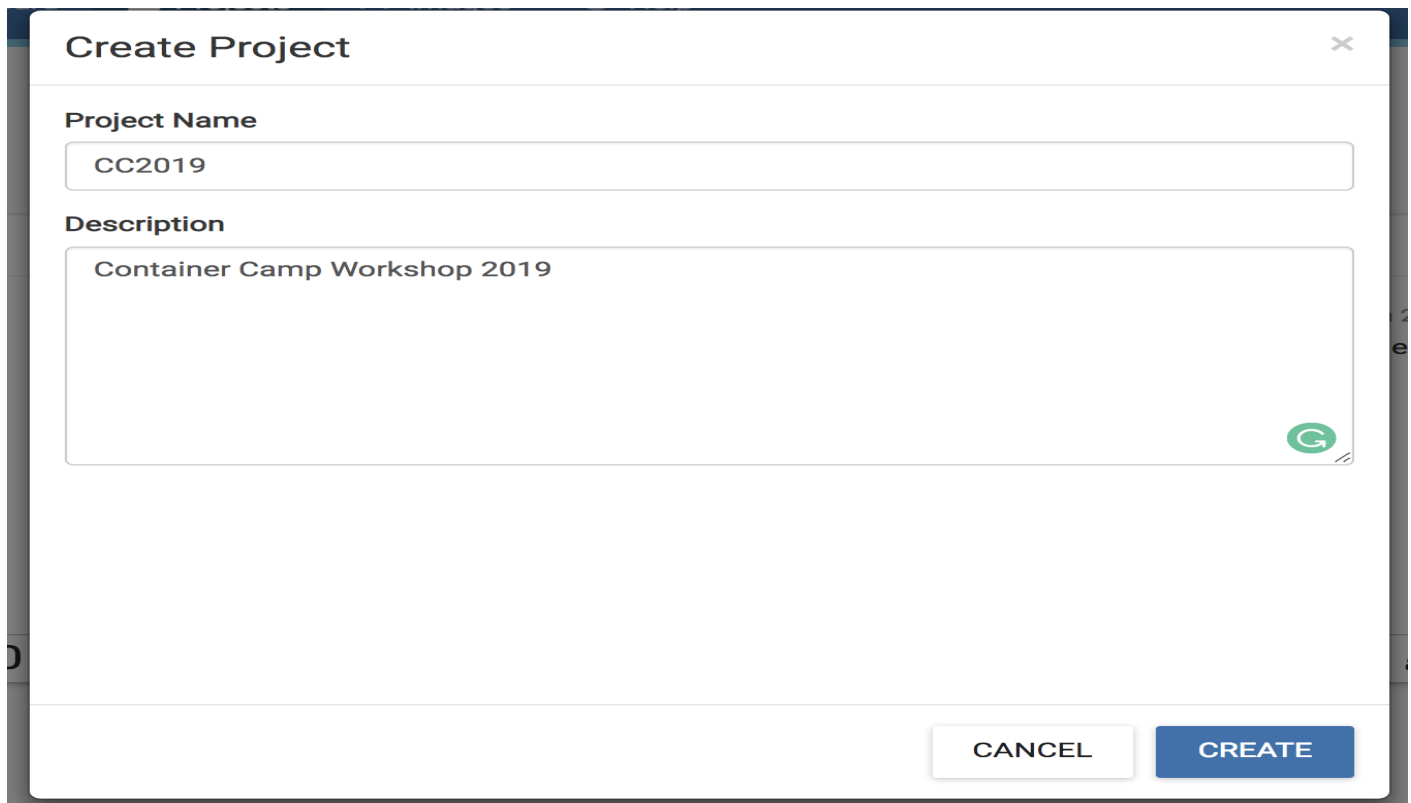
HTTPS cyverse.org

Links to CyVerse Resources

- [Additional Information](#)
- [Service Status](#)
- [Learning Center](#)
- [About CyVerse](#)

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

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

A screenshot of a 'Create Project' dialog box. The dialog has a title bar with 'Create Project' and a close button (X). It contains two input fields: 'Project Name' with the text 'CC2019' and 'Description' with the text 'Container Camp Workshop 2019'. The description field has a green circular icon with a 'G' in the bottom right corner. At the bottom right, there are two buttons: 'CANCEL' and 'CREATE'.

3. Select the newly created project

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

https://atmo.cyverse.org/application/projects/7950/resources

CYVERSE™ Dashboard Projects Images Help

RESOURCES DETAILS

CC2019

NEW ↻ ⬆

- Instance ←
- Volume
- Link

instances to this project.

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



Launch an Instance / Select an Image

First choose an image for your instance

Show Featured Show Favorites Show All

Ubuntu 18.04

Showing 2 image(s) for "Ubuntu 18.04"

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

[Advanced Options](#) CANCEL LAUNCH INSTANCE

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

- Leave rest of the fields as default.

Launch an Instance / Basic Options

Basic Info

Instance Name

workshop tutorial

Base Image Version

1.0

Project

CC2019

Resources

Allocation Source

upendra_35

Provider

CyVerse Cloud - Marana

Instance Size

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

Allocation Used

0% of 168 AUs from upendra_35

Resources Instance will Use

A total 14 of 32 allotted CPUs

A total 32 of 128 allotted GBs of Memory

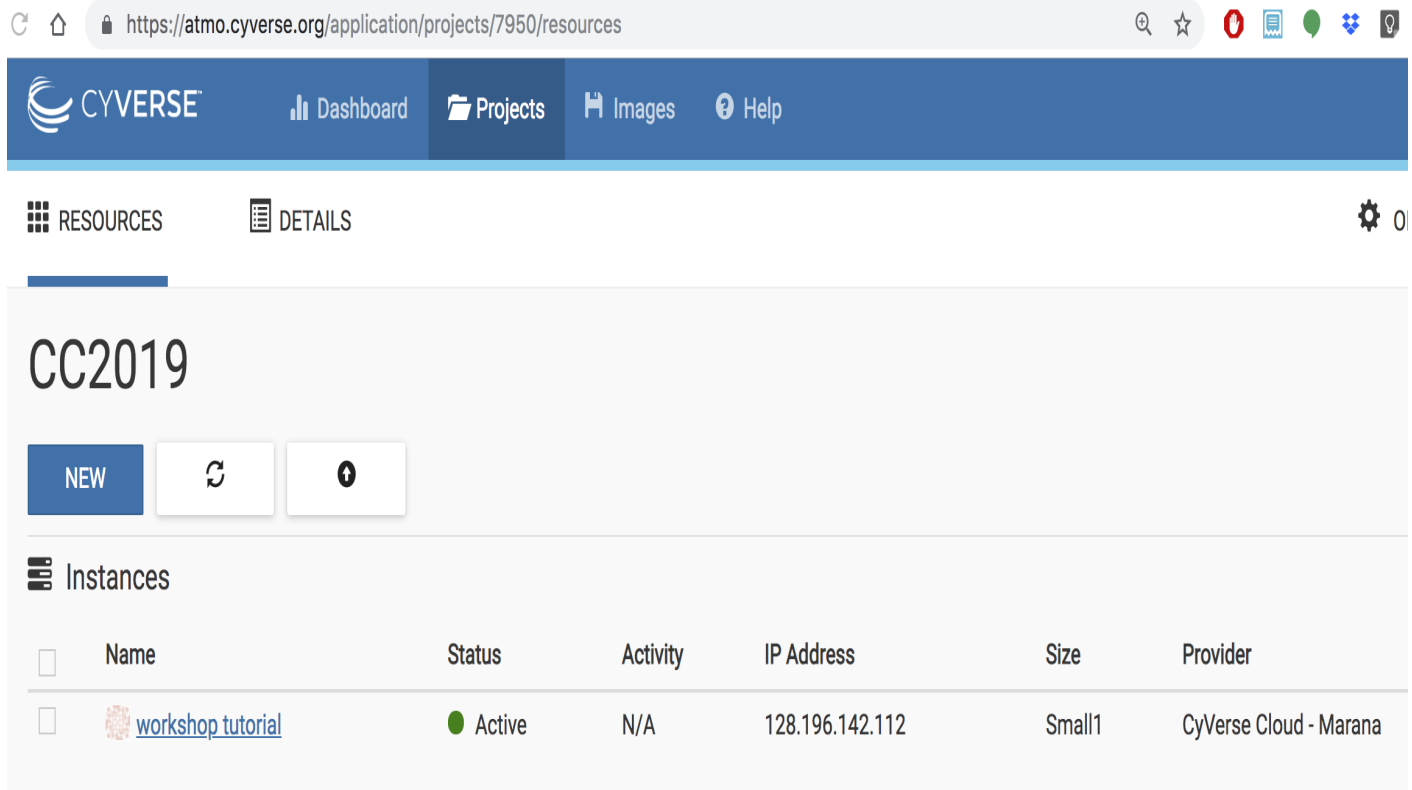
← Back

⚙️ Advanced Options

CANCEL

LAUNCH INSTANCE

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



https://atmo.cyverse.org/application/projects/7950/resources


CYVERSE Dashboard Projects Images Help

RESOURCES DETAILS

CC2019

NEW ↻ +

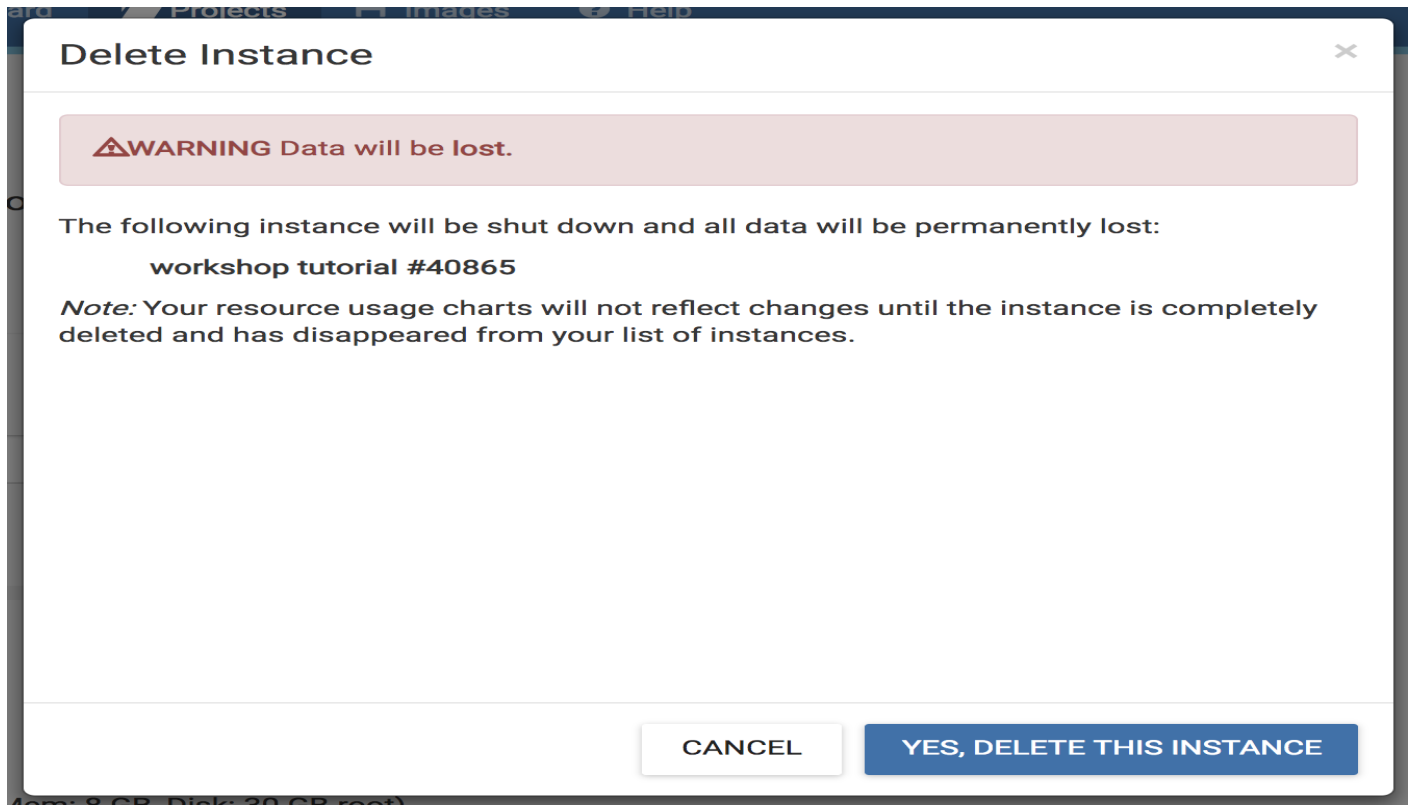
Instances

	Name	Status	Activity	IP Address	Size	Provider
<input type="checkbox"/>	 workshop tutorial	● Active	N/A	128.196.142.112	Small1	CyVerse Cloud - Marana

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

Deleting your instance


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



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

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

[↶](#) [🏠](#) <https://atmo.cyverse.org/application/projects/7950/resources>

 [Dashboard](#) [Projects](#) [Images](#) [Help](#)

[RESOURCES](#) [DETAILS](#)


CC2019

[NEW](#) [↺](#) [↑](#)

Instances

You have not added any instances to this project.

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



[!\[\]\(66d890a35b17e933e5c9cfad608118ca_img.jpg\) Learning Center Home](#)

Tool integration in the Discovery Environment (DE)

27.1 Why use the DE?

- Use hundreds of bioinformatics Apps without the command line (or with, if you prefer)
- Batch and interactive modes
- Seamlessly integrated with data and high performance computing – not dependent on your hardware
- Create and publish Apps and workflows so anyone can use them
- Analysis history and provenance – “avoid forensic bioinformatics”
- Securely and easily manage, share, and publish data

27.2 Types of apps

CyVerse tool: Software program that is integrated into the back end of the DE for use in DE apps

CyVerse app: graphic interface of a tool made available for use in the DE

- **Executable:** user starts an analysis and when the analysis finishes they can find the output files in their ‘Analyses’ folder
 - **DE:** run locally on our cluster
 - **HPC:** labeled as ‘Agave’ in the DE. Run on XSEDE resources at Texas Advanced Computing Center (TACC)
 - **OSG:** run on the Open Science Grid
- **Interactive:** also called Visual and Interactive Computing Environment (VICE). Allows users to open Integrated Development Environments (IDEs) including RStudio, Project Jupyter and RShiny and work interactively within them.

The (containerized) tool must be integrated into the Cyverse DE first. Then an app (interface) can be built for that tool.

27.3

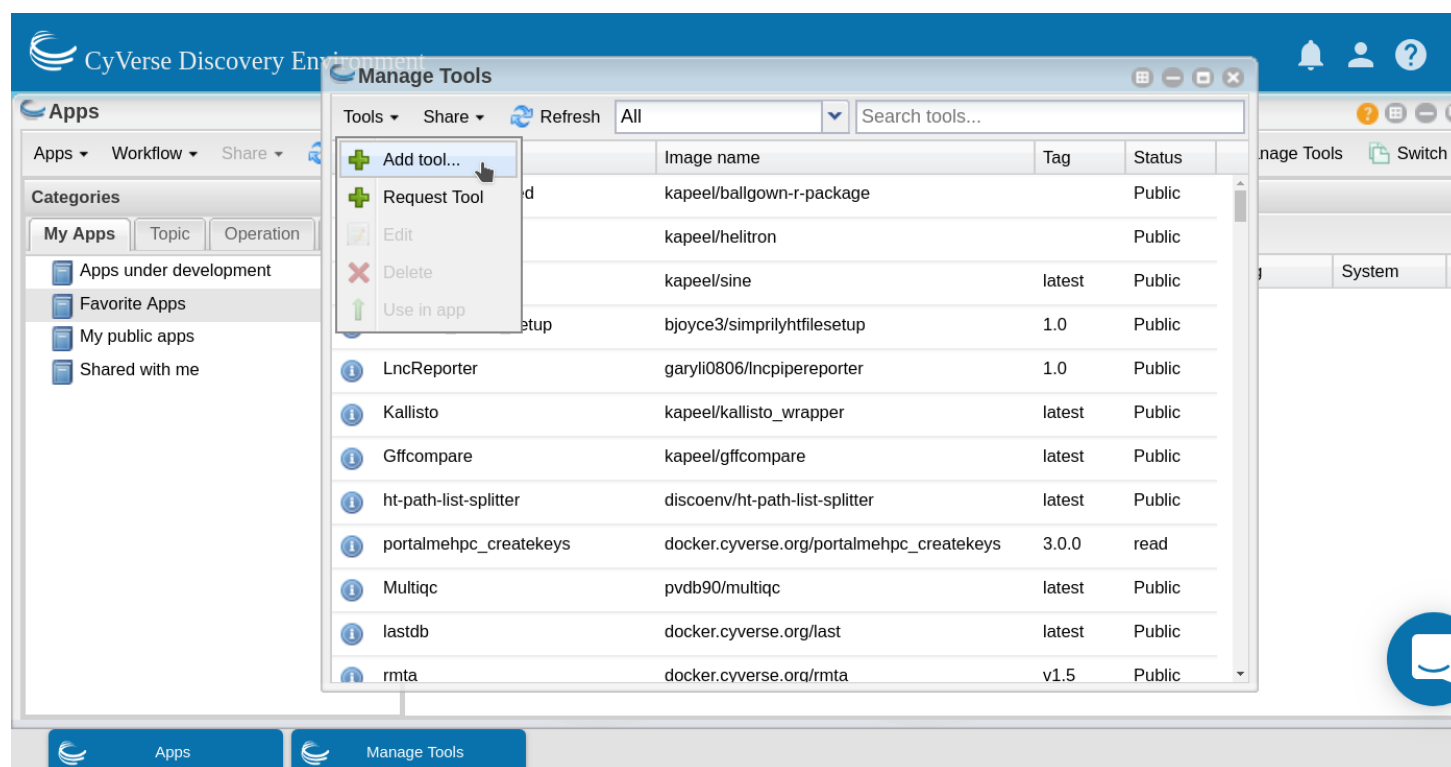
27.4 Building an App for Your Tool

You can build an app for any tool that:

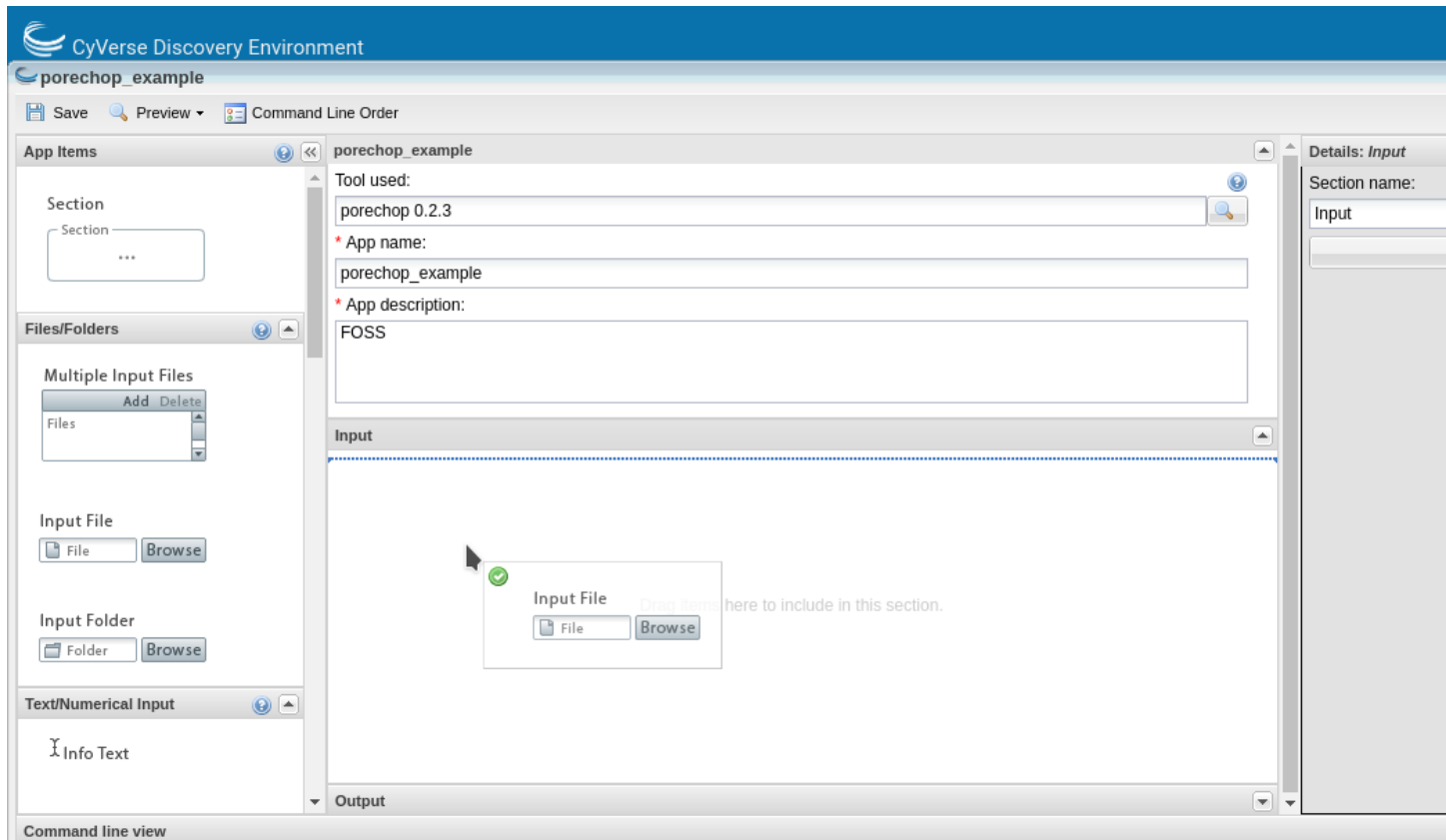
- is private to you
- is shared with you
- is public

Note: It is a good idea to check to see if the tool you want is already integrated before you start. The tool may be there already and you can build an app using it.

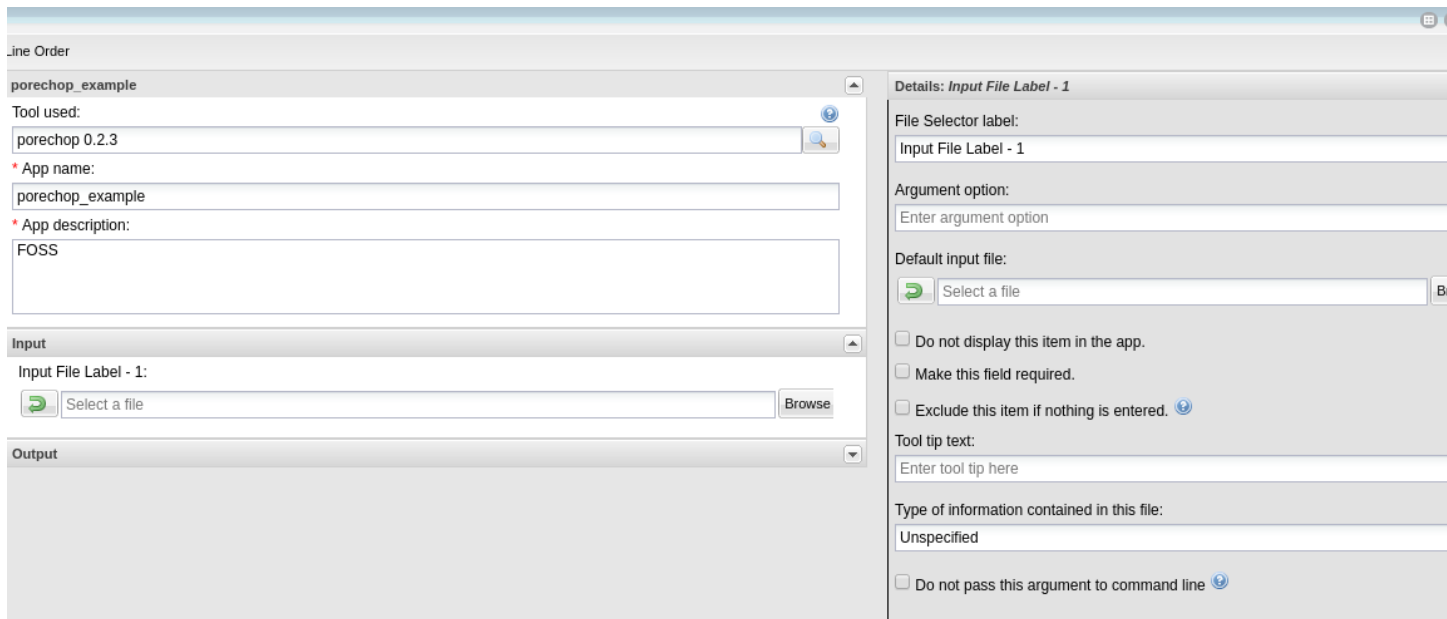
In the ‘Manage Tools’ window search for ‘porechop’ in the search bar at the top of the window. Select the porechop public tool and choose ‘Use in App’ from the ‘Tools’ menu



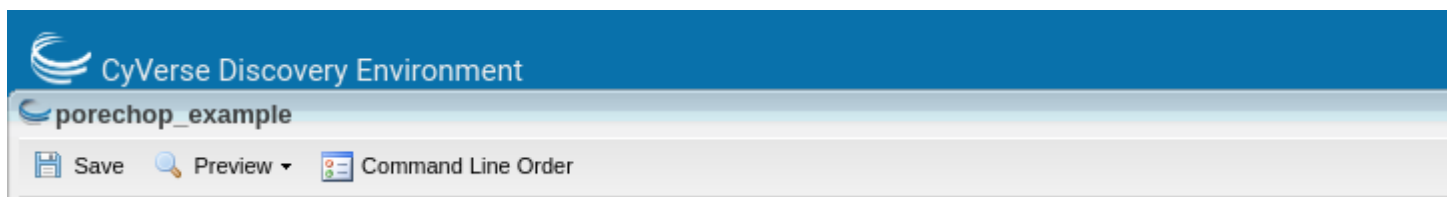
This will open the ‘Create App’ window. The tool to use will be pre-populated. Choose an informative app name and description (eg. tool name and version). Apps features can be added by dragging the feature from the left pane into the center pane.



You can edit the details of an app feature by selecting it in the center pane and editing in the right pane. Divide the app into sections appropriate for that tool (input, options and output are usually sufficient sections for simple apps).



For each option you add, you will need to specify what the option is, the flag (if there is one) and whether that option is required. If an option is not required be sure to check the 'exclude if nothing is entered' box. For tools that have positional arguments (no flags, eg. -z) you can modify the order of the commands by clicking the 'command line order' at the top of the window.

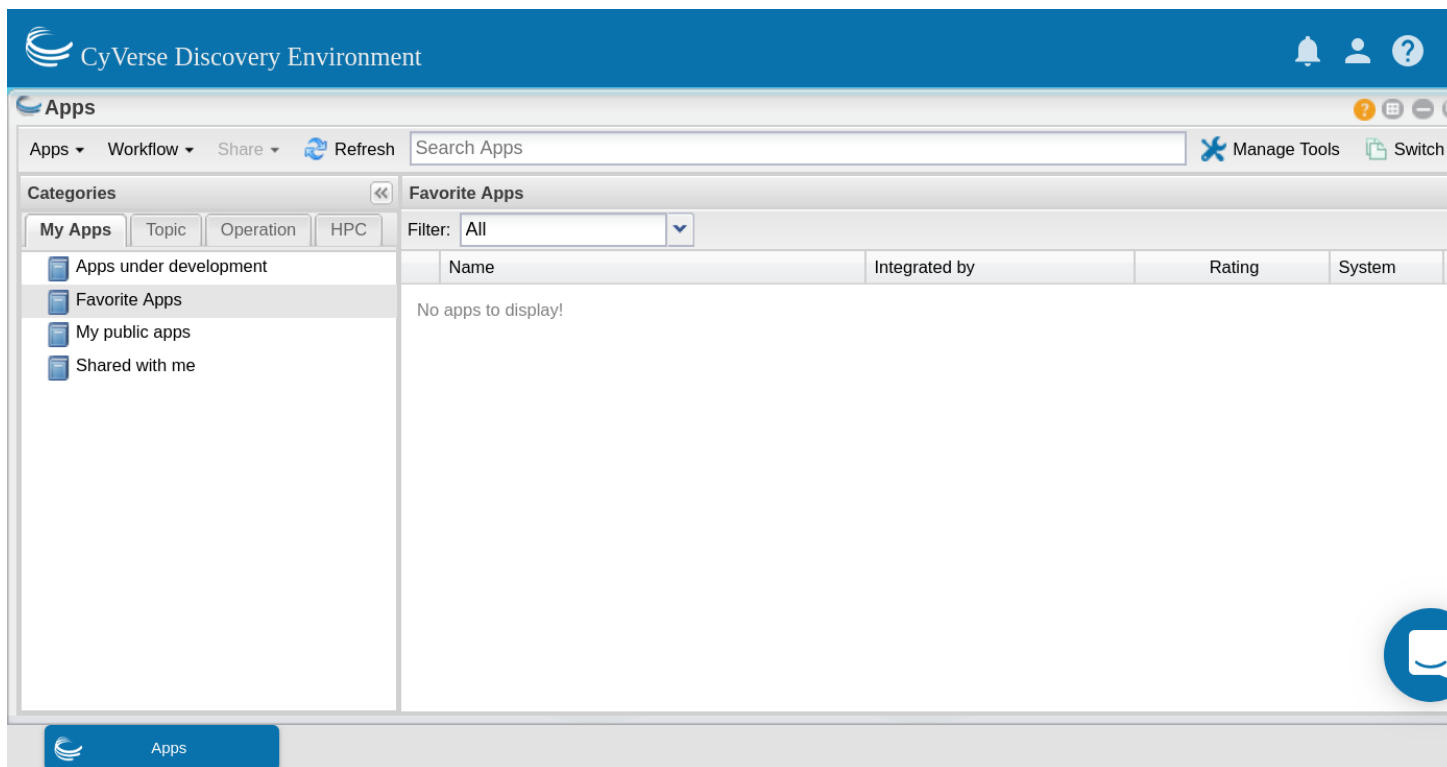


As you add options to your app you will see in the bottom pane (command line view) what the command would look like on the command line.



Although it is best to add all of the options for your tool, as it makes the app the most useful, you can expose as many or as few options as you like (as long as you add all the required options). Once you have finished adding options click save and close your app.

Now test your app with appropriate data. Your app can now be found in the 'My apps in development' category of the 'Apps' window (which displays by default).



Once you know your app works correctly you can share or publish it as you wish. Public apps must have example data located in an appropriately named folder here:

```
/iplant/home/shared/iplantcollaborative/example_data
```

All public apps also have a brief documentation page on the [CyVerse Wiki](#)

To publish your app click on 'Share' at the top of the 'Apps' window and select 'Make public'. You will need to

supply a:

- Topic (eg. genomics)
- Operation (eg. assembly)
- location of the example data
- brief description of inputs, required options and outputs
- link to CyVerse Wiki documentation page
- link to documentation for the tool (provided by the developers)

27.5 Additional resources

- [DE Guide](#)
- [DE Manual](#)
- [VICE Manual](#)
- [Using CyVerse for a shared project](#)

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



Deploying apps in CyVerse Discovery Environment

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

This [paper](#) will guide you to bring your dockerized tools into CyVerse DE.

F1000Research

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



SOFTWARE TOOL ARTICLE

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

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

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

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

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

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

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

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

1. Build and test your Docker images

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

For this tutorial I will use the `tensorflow image classifier` docker image that I built using this [code](#).

Building the Docker image from the Dockerfile

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

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

Testing Docker image with test data

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

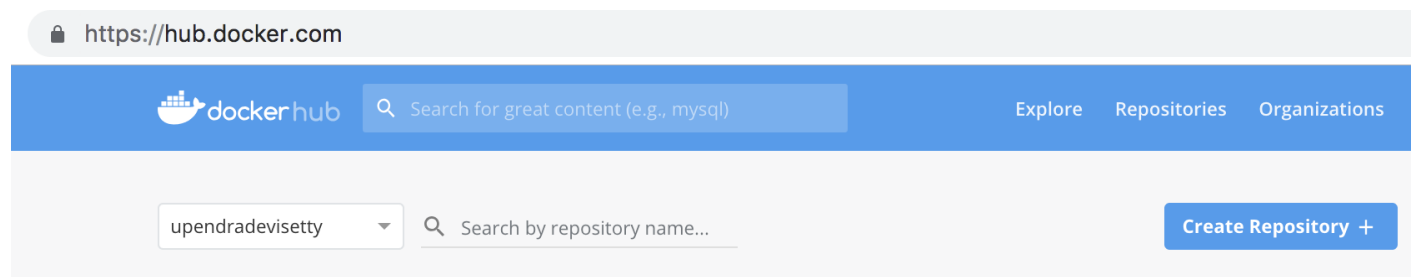
This generates a file called `16401288243_36112bd52f_m.out` that consists of classification percentages such as

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

2. Push your Docker image to public repositories

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


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



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

[Repositories](#)[Create](#)

Create Repository

 upendradevisetty ▼

tensorflow_image_classifier

Description

Visibility

Using 0 of 1 private repositories. [Get more](#)



Public 

Public repositories appear in Docker
Hub search results



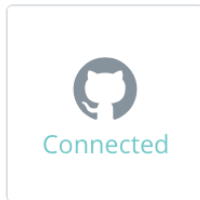
Private 

Only you can view private repositories

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

Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository. [Learn More.](#)



upendrak



tensorflow_image_classifier



▼ Click here to customize the build settings

BUILD RULES [+](#)

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

Source Type	Source	Docker Tag	Dockerfile location	Build Caching	
Branch ▼	master	1.0	Dockerfile	<input checked="" type="checkbox"/>	

► View example build rules

[Cancel](#)[Create](#)[Create & Build](#)

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

Automated Builds

Autobuild triggers a new build with every `git push` to your source code repository. [Learn More](#).

 [upendrak/tensorflow_image_classifier](#) | Use Docker Hub's infrastructure | Autotests: Off

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

Here is the docker image built using automated build for the tensorflow image classifier on [Dockerhub](#)

3. Add Docker images as tool in DE

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

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

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

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

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

Add Tool

tensorflow_image_classifier upendradevisetty/tensorflow_image_classifier

Tool Information

* Tool Name: tensorflow_image_classifier

Description: Tensorflow Image Classifier

* Version : 1.0

* Image name: upendradevisetty/tensorflow_image_classifier

Tag: 1.0

Docker Hub URL:

* Type: executable

* OSG Image Path:

Entrypoint:

Working Directory:

UID:

Max CPU Cores:

Memory Limit: 0 GB

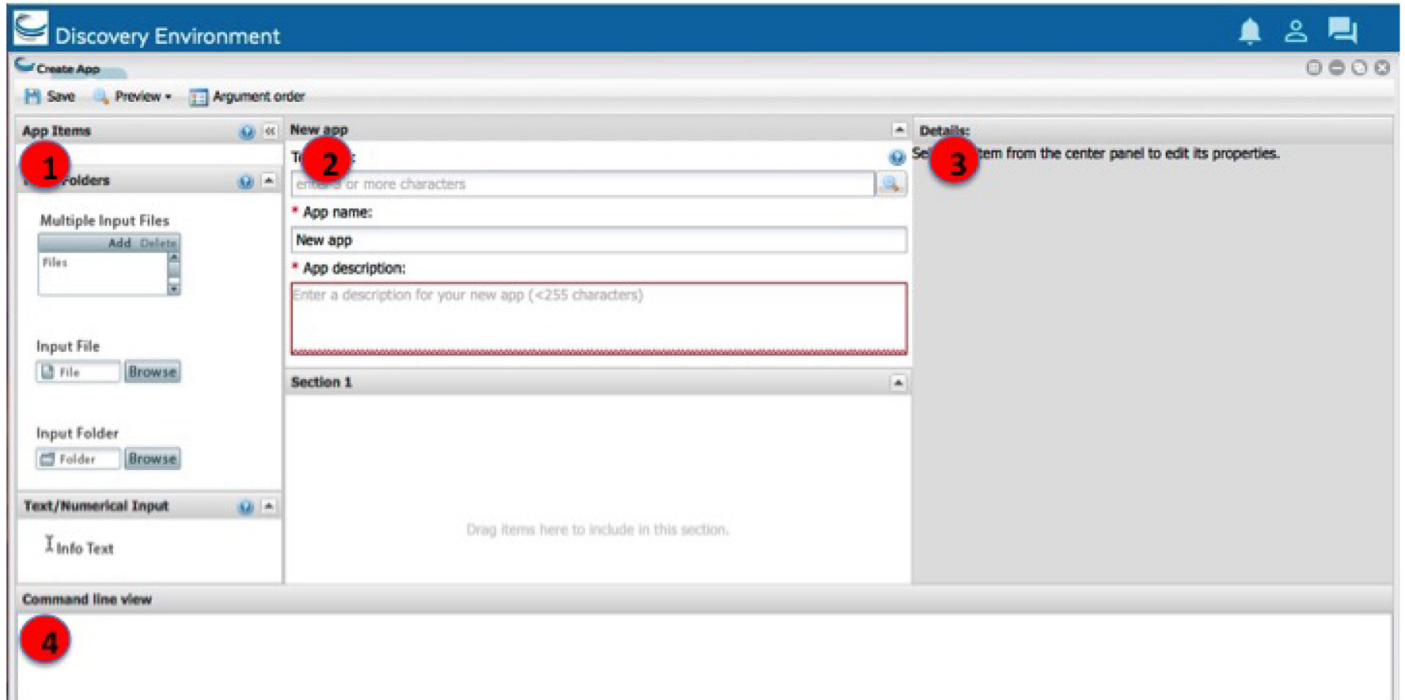
OK Cancel

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

4. Create an App UI for the tool in DE

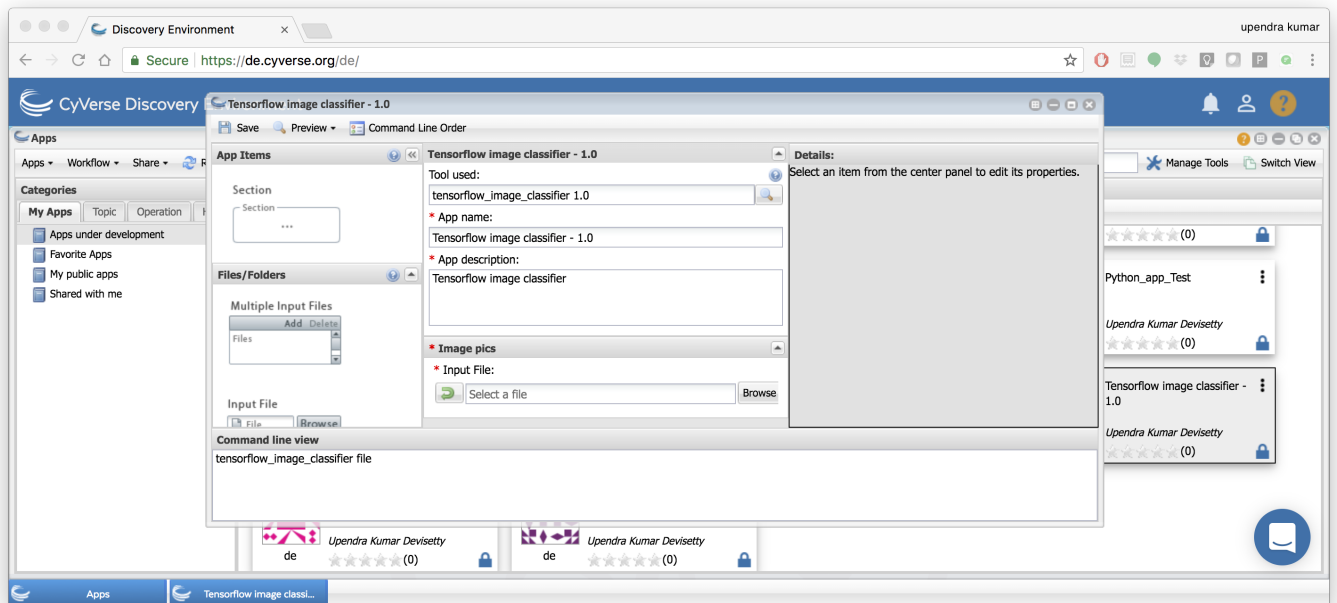
Once the Dockerized tool is added, you can create the app UI for the tool. The `Create App` window consists of four distinct sections:

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



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

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



5. Test the app using appropriate test data in DE

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

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

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



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

← → ↻ 🏠 🔒 <https://de.cyverse.org/de/>

CyVerse Discovery Environment

Apps

Apps ▾ Workflow ▾ Share ▾ ↻ Refresh

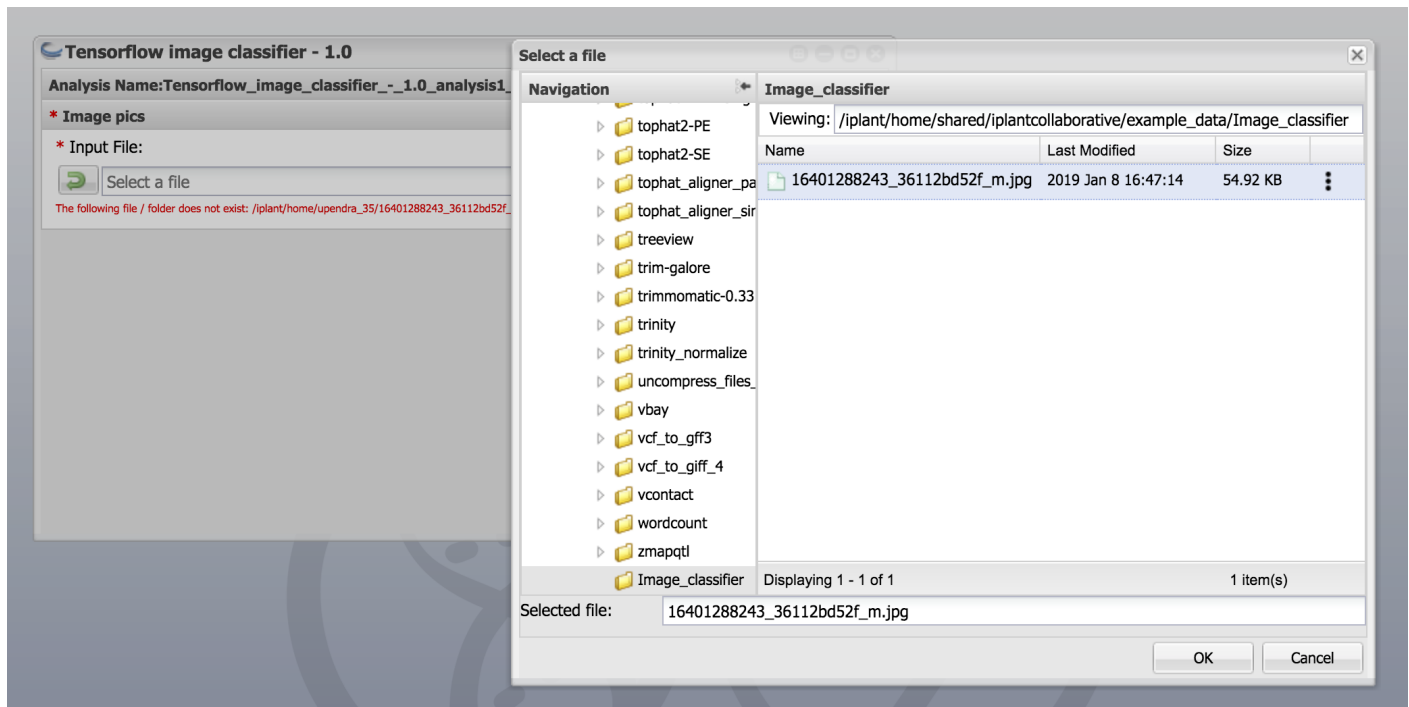
Categories << **Search results: 1 found for image classifier**

My Apps Topic Operation HPC Filter: All ▾

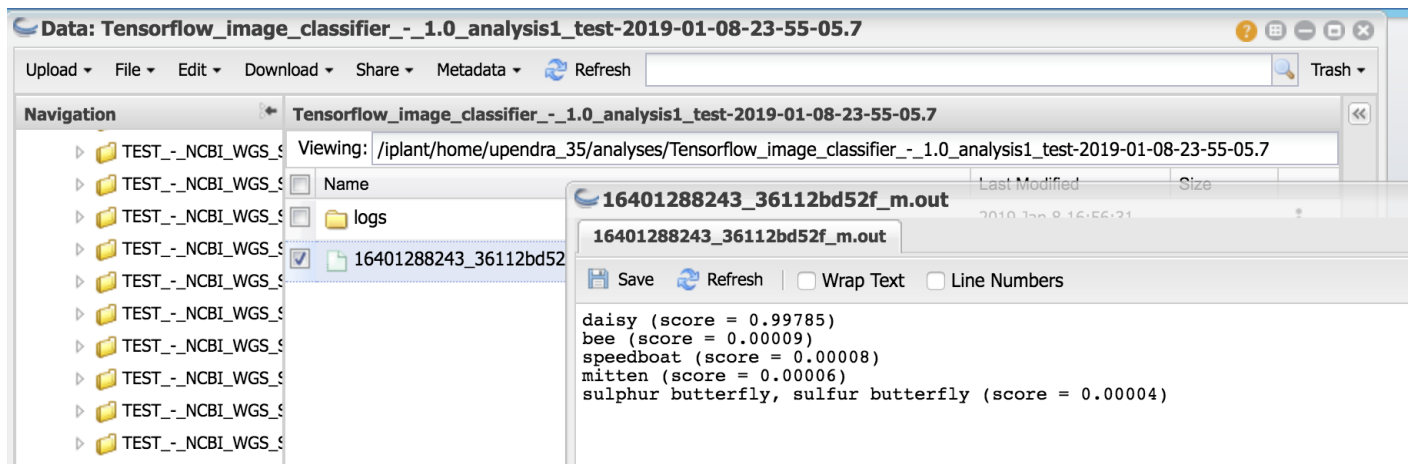
Name
🔒 Tensorflow image classifier - 1.0

Apps under development
Favorite Apps
My public apps
Shared with me

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



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



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

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



Deploying interactive apps in CyVerse Discovery Environment

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

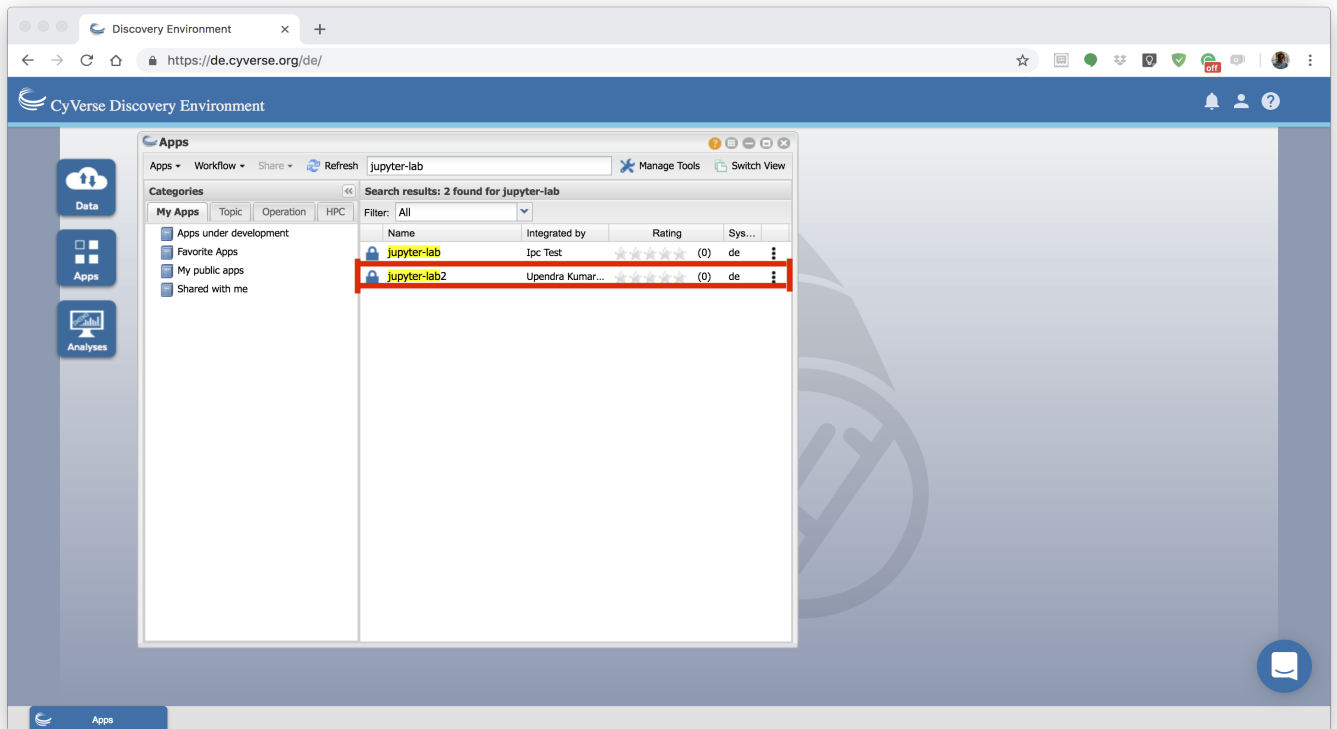
VICE (Visual Interactive Computing Environment) lets users interact with their data and do analyses in their favorite programming language in one place in an iterative way. Researchers can now explore their datasets interactively by easily changing parameters of selected analysis applications without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

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

First log-in [CyVerse DE](#)

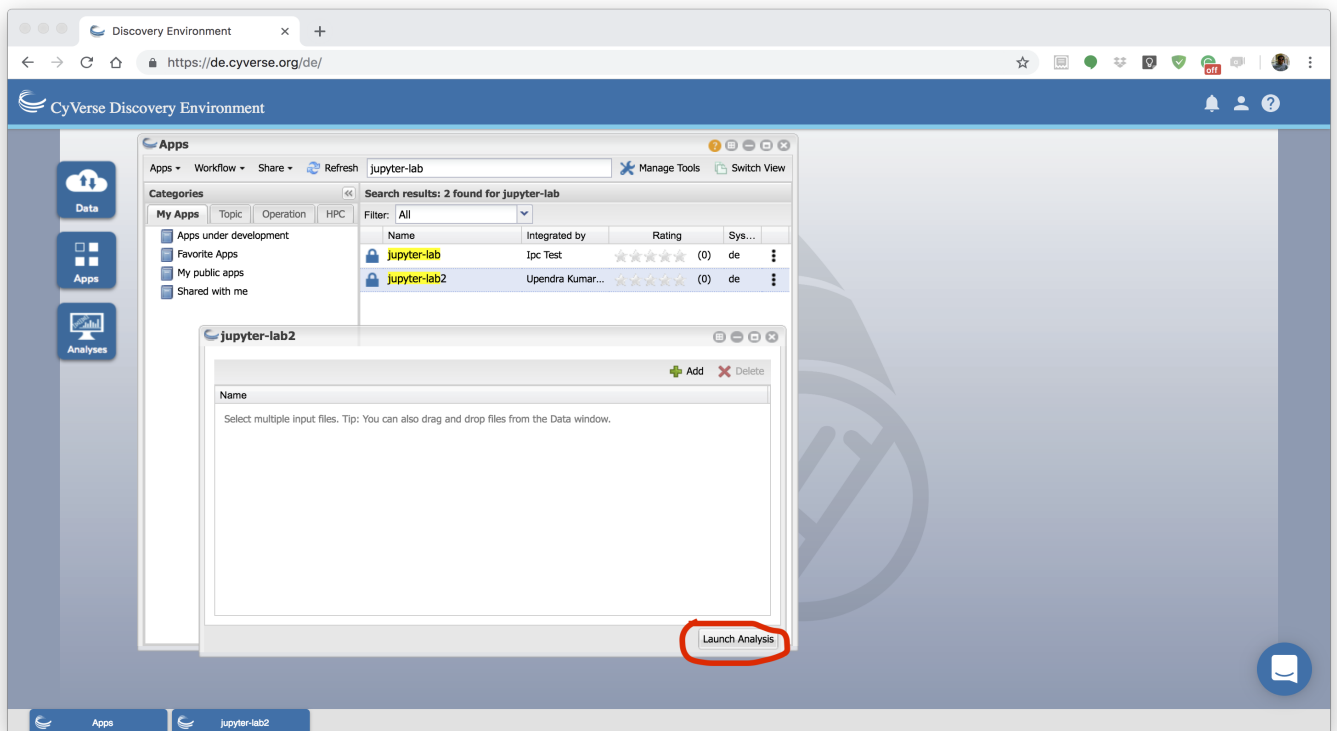
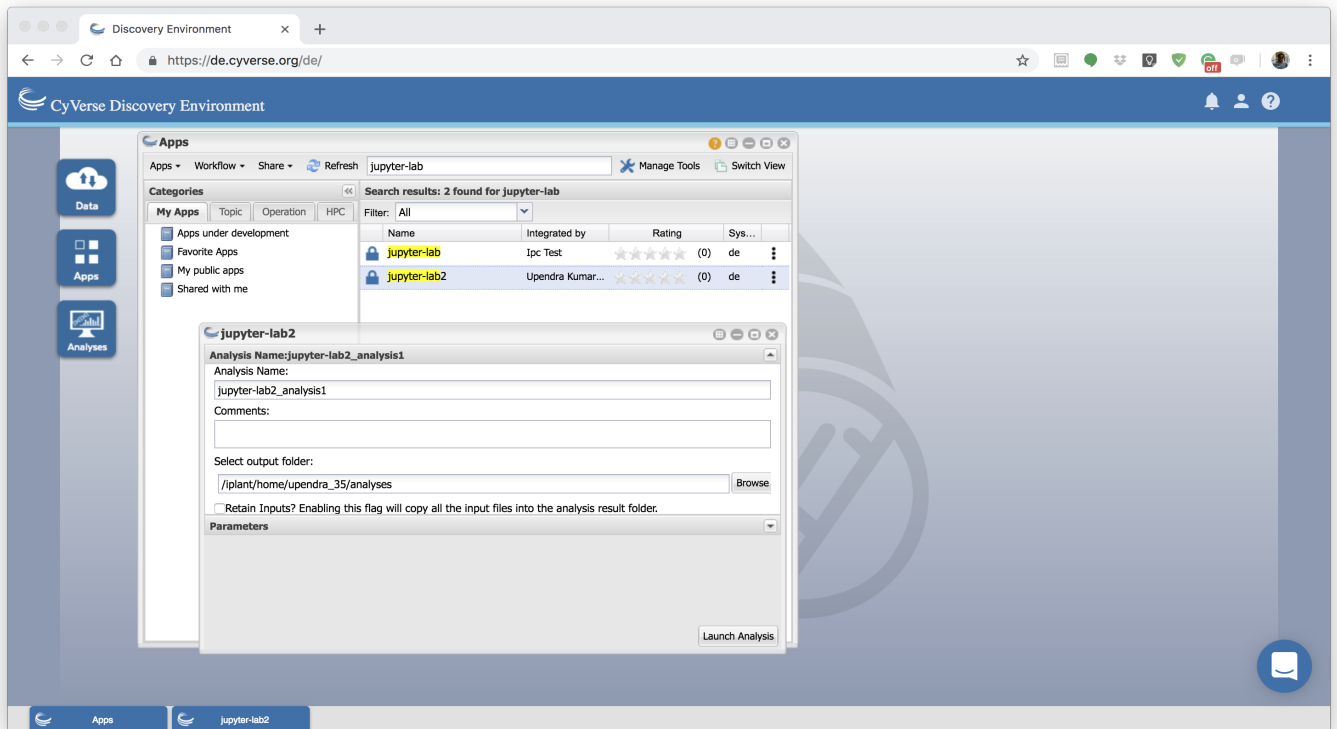
29.1 1. Search JupyterLab App

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



29.2 2. Launch analysis

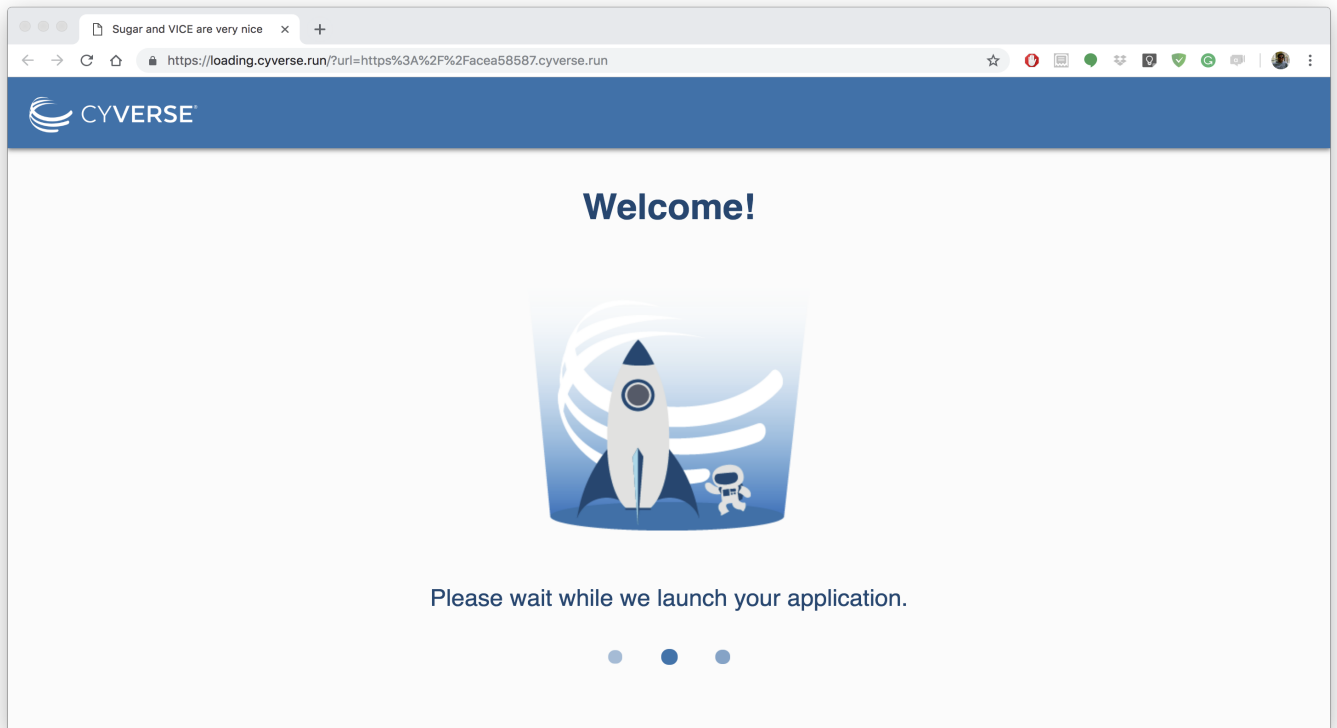
Launch the JupyterLab app by clicking **launch analysis**. Before you launch, you can either drag and drop or browse the files that you want to use with Jupyter-lab. There is currently no restriction of how many files and size of the files that can be launched along with JupyterLab app.



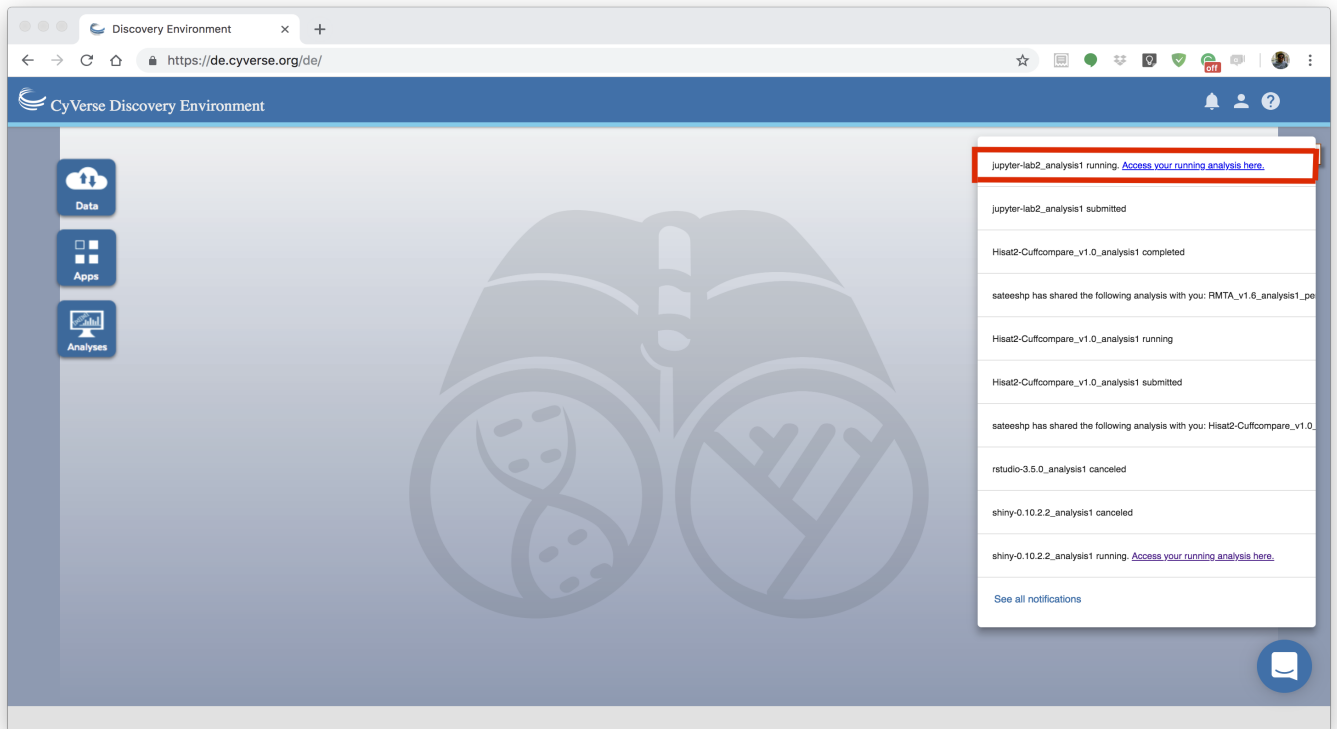
Note: The first two steps of launching apps are same as with other DE apps.

29.3 3. Navigate to JupyterLab url

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



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



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

More information about the JupyterLab can be found [here](#)

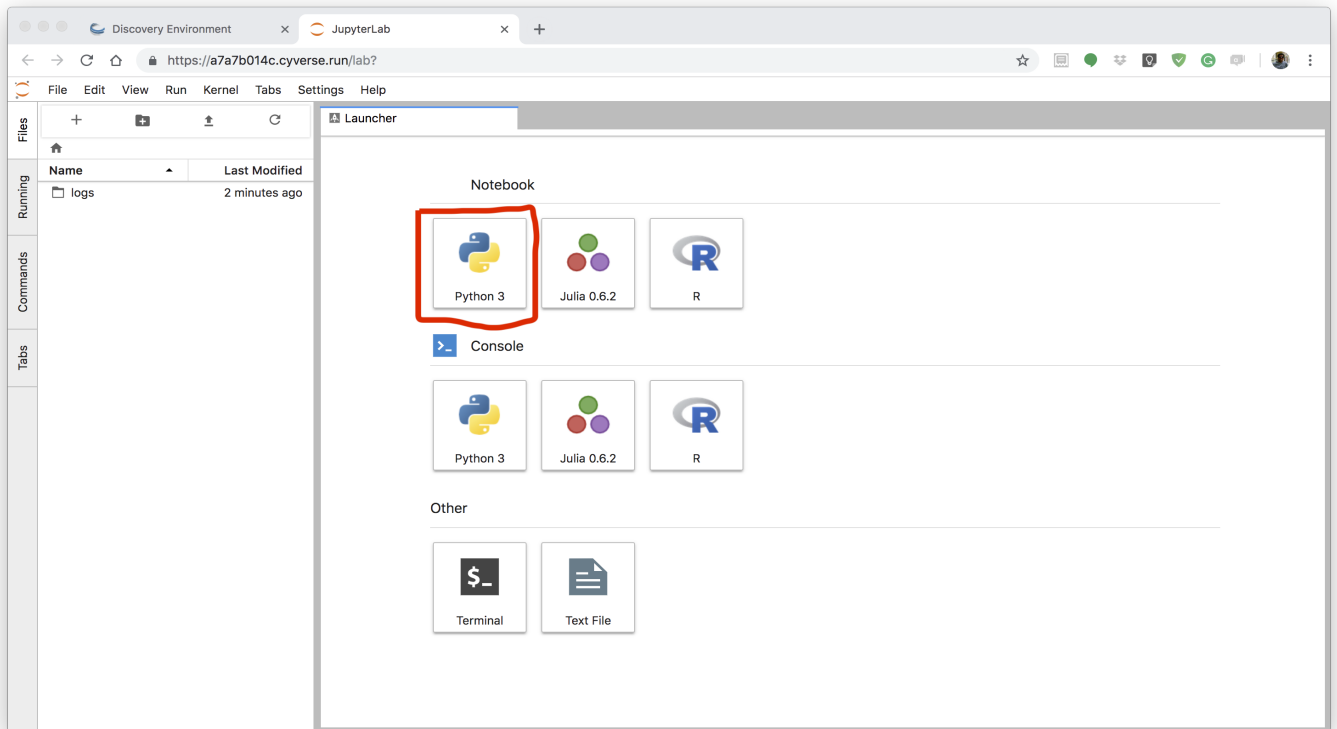
29.4 4. Create Jupyter notebook

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

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

To know more about notebooks in JupyterLab click [here](#)

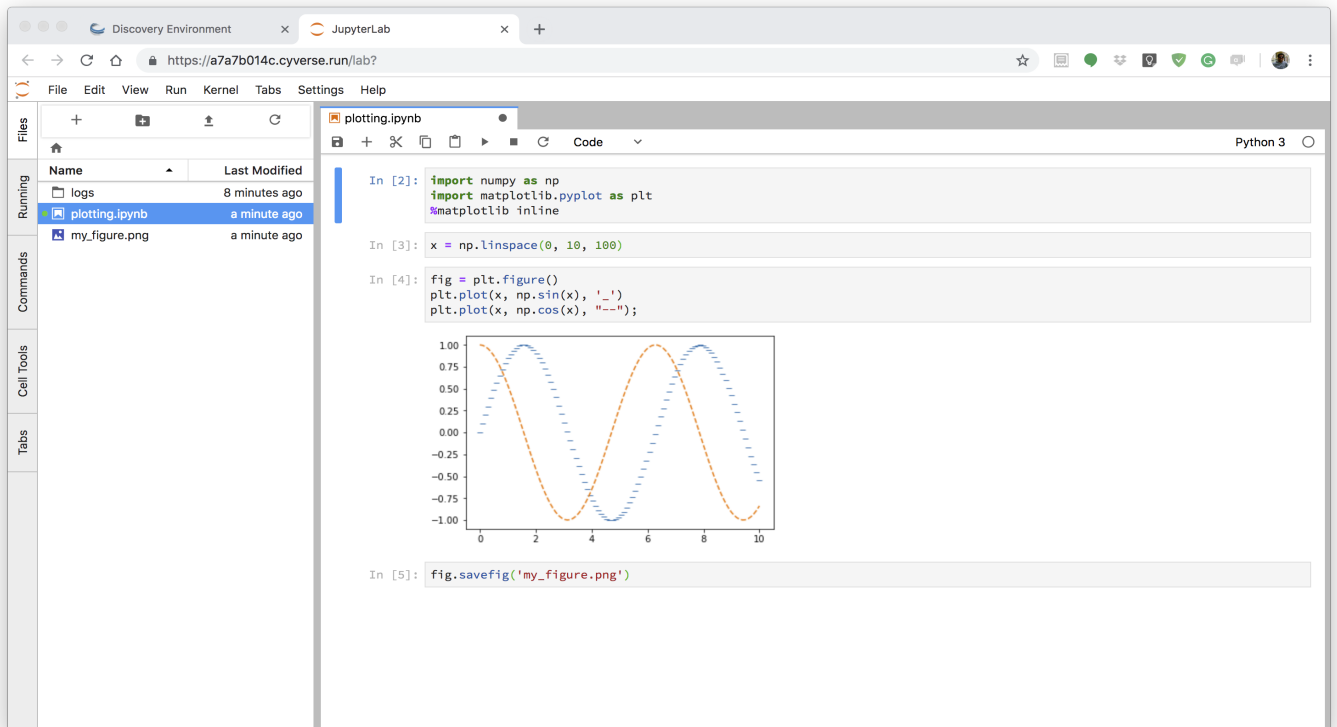
Tip: To open the classic Notebook from JupyterLab, select “Launch Classic Notebook” from the JupyterLab Help menu.



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

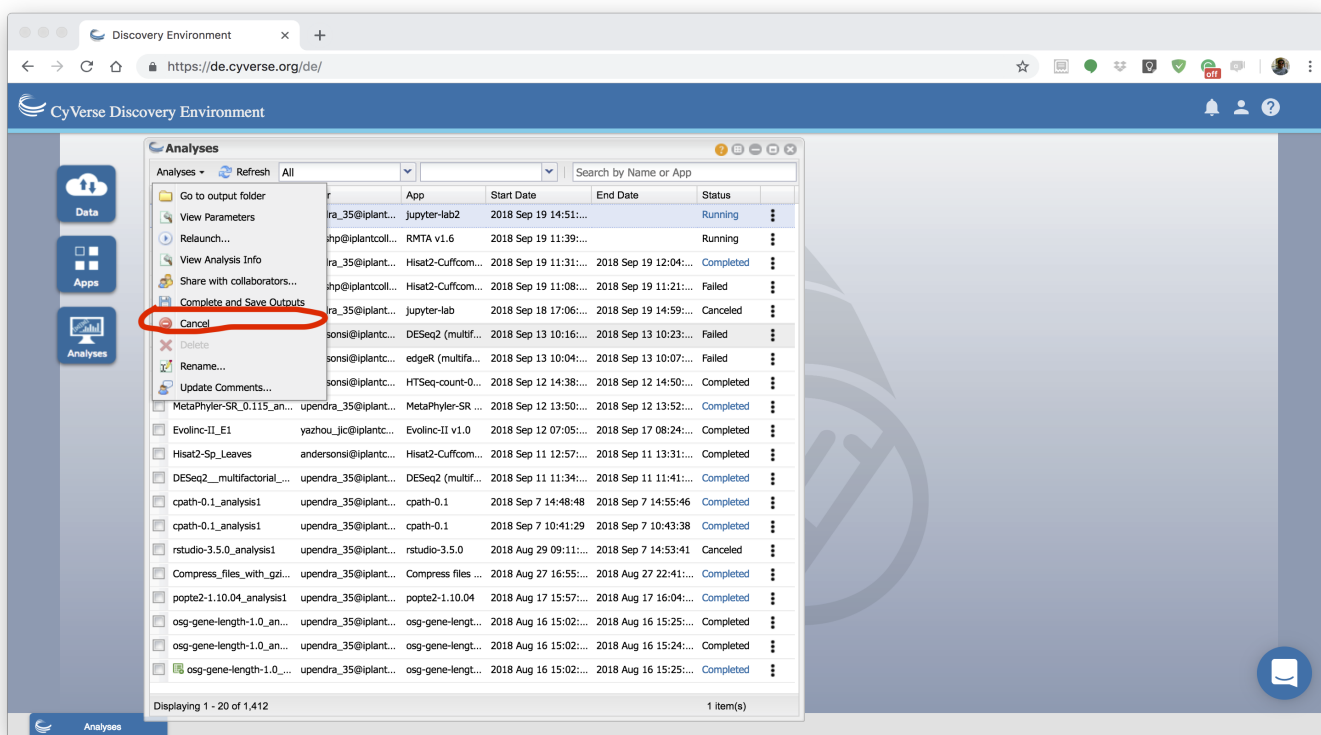
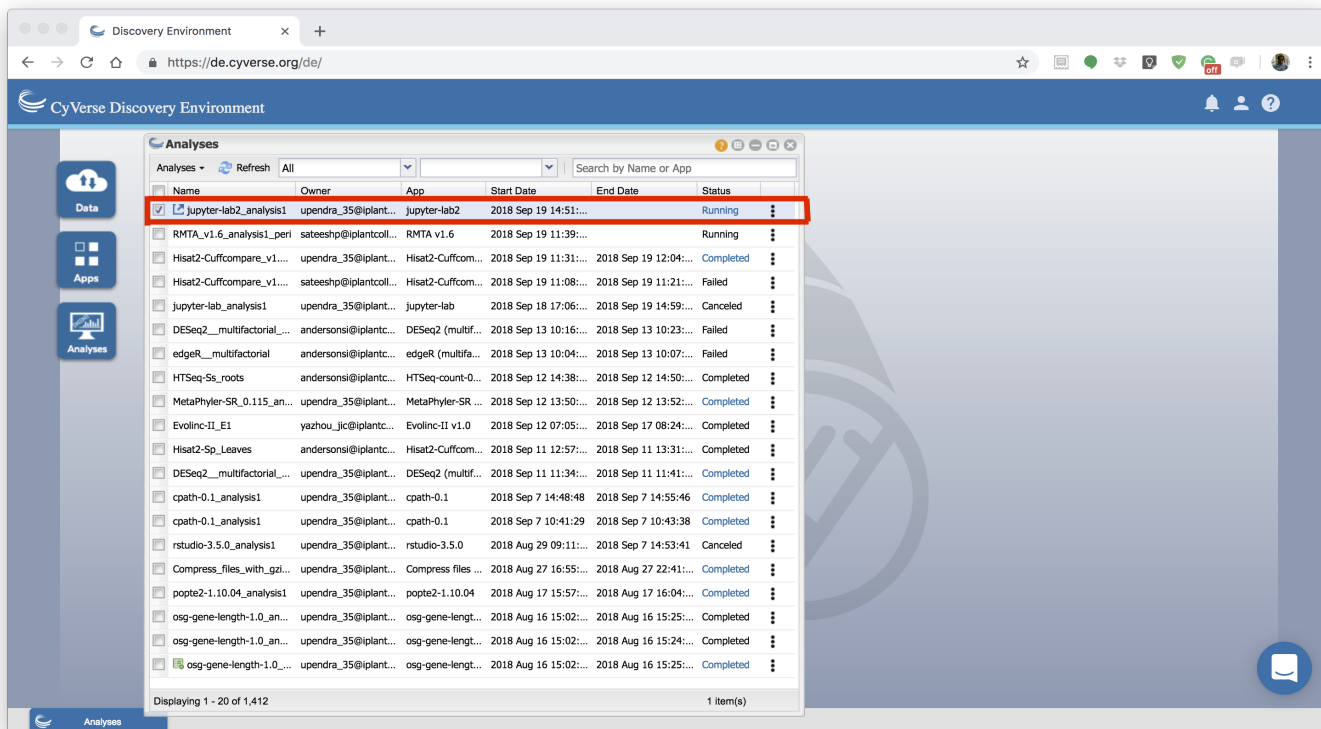
29.5 5. Write your code

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



29.6 6. Complet and Save Outputs

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



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

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

Docker related resources

[Awesome Docker](#)

[Docker labs](#)

[Docker Community Slack](#)

[Docker Community Forums](#)

[Docker hub](#)

[Docker documentation](#)

[Docker on StackOverflow](#)

[Docker on Twitter](#)

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

[Docker tips](#)

[Docker cloud](#)

[Docker store](#)

Interesting tutorials and blog posts:

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

Singularity related resources

[Singularity Homepage](#)

[Singularity Hub](#)

[University of Arizona Singularity Tutorials](#)

[NIH HPC](#)

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

31.1 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 32

Other resources

University of Arizona Campus Resources

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

CHAPTER 33

For instructors!

Coordinating Web site work

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

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

Technical info re adding content to the Web site

All the Container Camp workshop tutorials are stored on [GitHub](#).

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

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

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

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

Updating the “official” Web site.

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

Building a local copy of the Web site.

Briefly,

- clone the repo:

```
git clone https://github.com/CyVerse-learning-materials/container_camp_workshop_2019.  
↪ git`
```

- set up a virtualenv with python2 or python3:

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

- install the prerequisites:

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

- build site:

```
make html
```

- open / click on

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

Formatting, guidelines, etc.

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

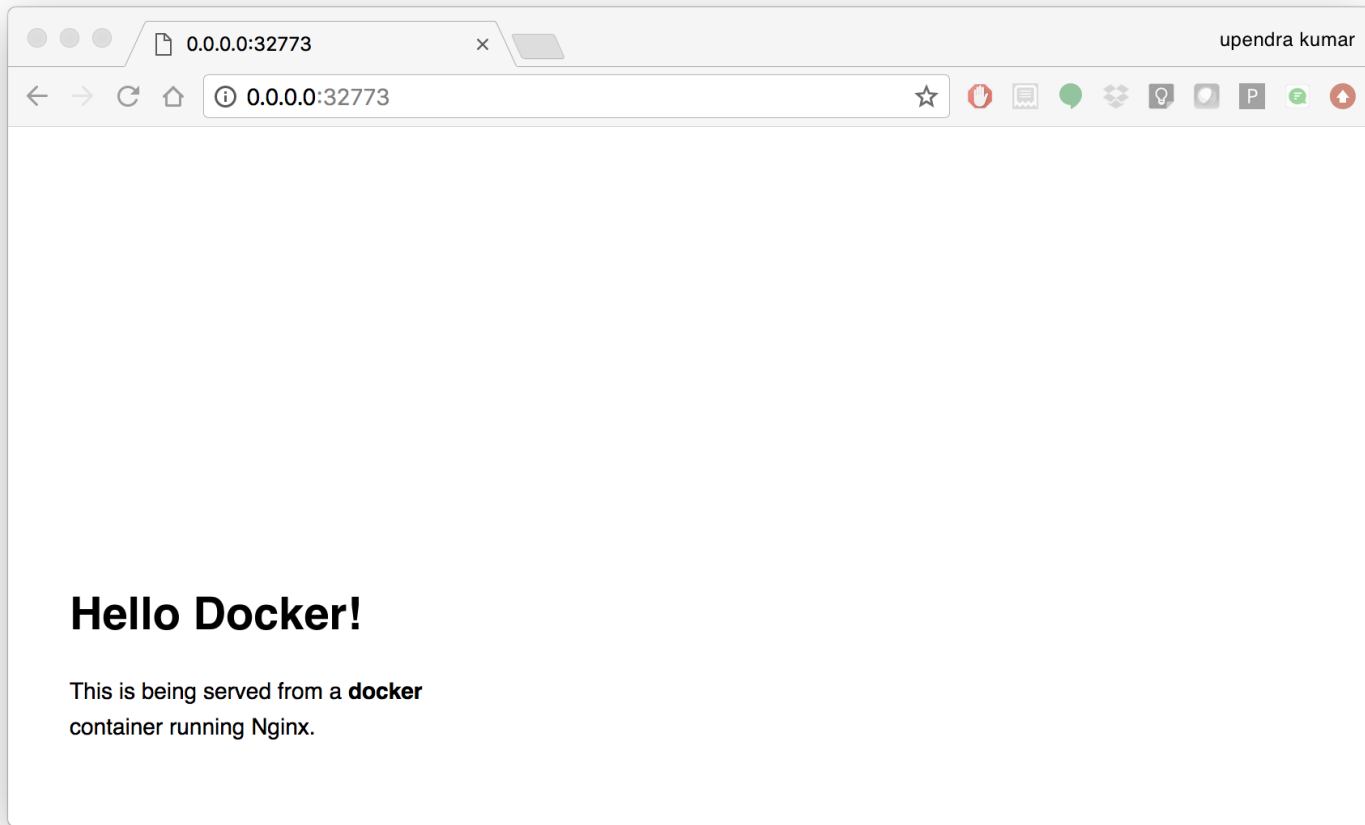
(Note that you can go visit the github repo and it will helpfully render *.rst* files for you if you click on them! They just won't have the full site template.)

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

https://cyverse-container-camp-workshop-2019.readthedocs-hosted.com/_static/filename

Images

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



The code that generates this image is this

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

Problems? Bugs? Questions?

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

Fix or improve this documentation

- On Github: [Repo link](#)
- Send feedback: support@cyverse.org