INTERNET ARCHIVE Wayback Machine https://docs.rc.uab.edu/workflow_solutions/getting_containers/ Go NOV DEC 04 2022 MAR 2024
7 captures 4 Dec 2022 - 25 Jan 2025 2021 About this capture

Research Computing Documentation
Home
Account Management
Data Management
Workflow Solutions
- Using the Shell
- Using Anaconda
- Using Workflow Managers
- Using Git
- Getting Containers
- Getting Software from GitHub and GitLab

Cheaha Guide
UAB Cloud (cloud.rc) Guide
UAB Kubernetes Guide
National Research Computing Cyberinfrastructure
Grants & Contracts
Education & Outreach
Policies
Reporting Documentation Errors
Contributing Content
Help

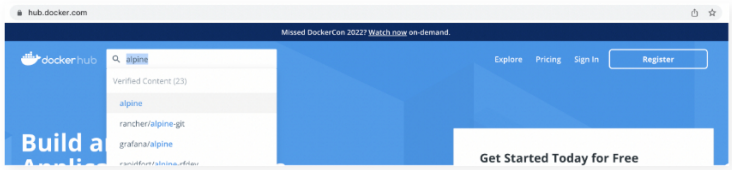
Software Containers

Containers help to manage software installations and all their dependencies in a single large image.

Docker is an open-source platform for building, deploying, running, updating, and managing containers.

Fantastic Containers and Where to Find Them

Docker containers are available in https://hub.docker.com/. This docker hub repository allows to share containers and use pre-existing docker images.



Using Containers on UAB RC Cloud (cloud.rc.uab.edu)

To access docker containers, install `Docker` in your system. To install docker desktop on your computer, follow this link: Docker Desktop Page.

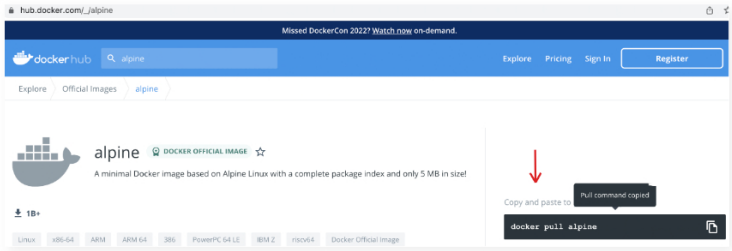
Docker Installation on UAB RC Cloud

Following are the installation instructions to install `Docker` on UAB RC Cloud with Ubuntu operating system. Tested the installation on Ubuntu 20.04. Setting up UAB RC Cloud account can be found in UAB RC Cloud.

```
sudo apt-get update
sudo apt install docker.io
```

Using a Docker Container from DockerHub

We can start pulling a container named `alpine` from the Docker hub. `alpine` is a general-purpose Linux distribution. Look for the container `alpine` in the docker hub, copy the pull command, and paste it into your terminal.




```
sudo docker pull alpine
```

```
(base) ubuntu@osg-instance:~$ sudo docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
2408cc74d12b: Pull complete
Digest: sha256:686d8c9dfa6f3ccfc8230bc3178d23f84eeaf7e457f36f271ab1acc53015037c
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Once the image is pulled, you can verify if the image exists using the below command. Note that if you do not specify the tag/version of the container, the recent version is built, and the tag is listed as `latest`.

```
sudo docker images
```

```
(base) ubuntu@osg-instance:~/alpine$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
alpine       latest    5274064a8e18   49 seconds ago   5.53MB
```

If you prefer to pull a particular version of the `alpine` container, you need to mention the tag details in your pull command. You can see the available tags/versions of `alpine` from the Docker hub.



Table of contents
Fantastic Containers and Where to Find Them
Using Containers on UAB RC Cloud (cloud.rc.uab.edu)
- Docker Installation on UAB RC Cloud
- Using a Docker Container from DockerHub

Create Your Own Docker Container
- Create a Dockerfile that has Miniconda Installed
- Running the Built Miniconda Docker Container Interactively
- Mounting Data Onto a Container

Sharing Containers Using UAB GitLab Container Registry
- Push Alpine Container from your System to UAB GitLab Container Registry



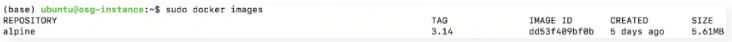
To pull particular version of `alpine` container, use the below syntax.

```
sudo docker pull container_name:tag
```

Here the `container_name` is `alpine`, and the tag is `3.14`.

```
sudo docker pull alpine:3.14
```

The existing image looks like,



Create Your Own Docker Container

You can create your own Docker container, build it, and upload/share them in the Docker hub or UAB GitLab container registry.

Let us take a synthetic python code and formulate the packages/dependencies required to build your software container. Below is a python script that requires packages, namely, numpy, scipy, and matplotlib. Next, the steps to create a `Dockerfile` is illustrated. Let us name this script `python_test.py`.

```
import numpy as np
import matplotlib
import pylab
import matplotlib.pylab as plt
import scipy.integrate as integrate

a = np.array([0, 10, 20, 30, 40])
print(a)

b = np.arange(-5, 5, 0.5)
print(b)

t = np.arange(0,20.5,0.5)
print(t)

result = integrate.quad(np.sin, 0, np.pi)
print(result)

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
plt.savefig('testing.png')
```

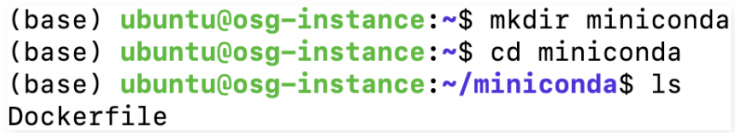
Create a Dockerfile that has Miniconda Installed

We require numpy, scipy, and matplotlib libraries to execute the above Python script. Following are the steps to create a specification file and build a container image.

1. Create an empty directory `miniconda`.

   ```
   mkdir miniconda
   ```

2. Create a `Dockerfile` within the `miniconda` directory with the following contents. The file name `Dockerfile` is case-sensitive.



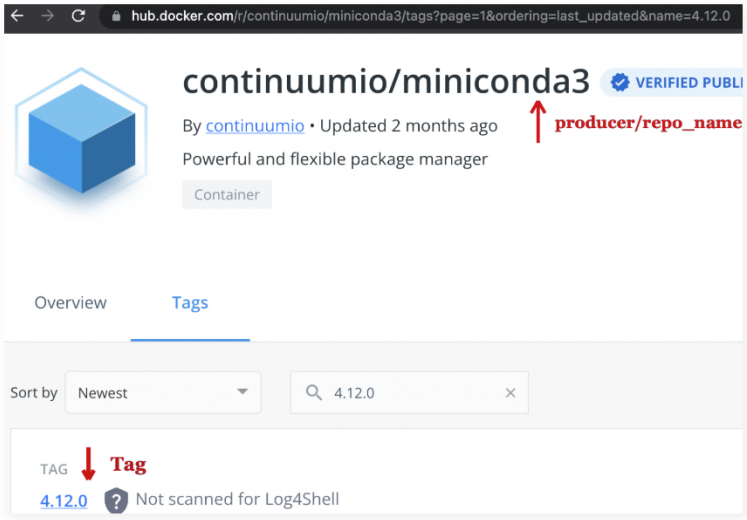

```
# You may start with a base image
# Always use a specific tag like "4.10.3", never "latest"!
# The version referenced by "latest" can change, so the build will be
# more stable when building from a specific version tag.
FROM continuumio/miniconda3:4.12.0

# Use RUN to execute commands inside the miniconda image
RUN conda install -y numpy">=1.16.5, <1.23.0"

# RUN multiple commands together
# Last two lines are cleaning out the local repository and removing the state
# information for installed package
RUN apt-get update \
&& conda install -y scipy=1.7.3 \
&& conda install -y matplotlib=3.5.1 \
&& apt-get --yes clean \
&& rm -rf /var/lib/apt/lists/*
```

This is the specification file. It provides Docker with the software information, and versions, it needs to build our new container. See the Docker Container documentation for more information https://docs.docker.com/engine/reference/builder/.

In the Dockerfile, we start with an existing container `continuumio/miniconda3:4.12.0`. This container is obtained from Dockerhub; here, `continuumio` is the producer, and the repo name is `continuumio/miniconda3`.



You may specify the required version from the `Tag` list. Here the tag/version is `4.12.0`. Also its a very good practice to specify the version of packages for numpy, scipy, and matplotlib for better reproducibility.

> **Containers and Reproducibiliy**
>
> Always include version numbers for Anaconda, package managers, software you are installing, and the dependencies for those software. Containers are not by nature scientifically reproducible, but if you include versions for as much software in the container as possible, they can be reproducible years later.

3. To build your container, change the directory to `miniconda` and use the below syntax to build the `Dockerfile`. Here we use `.` to say "current directory." This will only work if you are in the directory with the `Dockerfile`.

```
sudo docker build -t repository_name:tag .
```

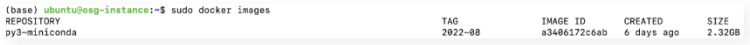
Here the repository_name is `py3-miniconda` and the tag is `2022-08`.

```
cd miniconda
sudo docker build -t py3-miniconda:2022-08 .
```



> **Note**
>
> The `.` at the end of the command! This indicates that we're using the current directory as our build environment, including the Dockerfile inside. Also, you may rename the `repository_name` and `tag` as you prefer.

```
sudo docker images
```



Running the Built Miniconda Docker Container Interactively

To run docker interactively and execute commands inside the container, use the below syntax. Here `run` executes the command in a new container, and `-it` starts an interactive shell inside the container. After executing this command, the command prompt will change and move into the bash shell.

```
sudo docker run -it repository_name:tag /bin/bash
```

To execute your container `py3-miniconda` interactively, run this command with the tag `2022-08'.

```
sudo docker run -it py3-miniconda:2022-08 /bin/bash
cd /opt/conda/bin/
```

The `python` executables to execute our synthetic python script are within the directory structure `/opt/conda/bin`.

```
(base) ubuntu@osg-instance:~$ sudo docker run -it py3-miniconda:2022-08 /bin/bash
(base) root@a9d745c2a4f4:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
(base) root@a9d745c2a4f4:/# /opt/conda/bin/python
Python 3.9.12 (main, Apr  5 2022, 06:56:58)
```

Path to Python executable

```
(base) root@a9d745c2a4f4:~# cd /opt/conda/bin/
(base) root@a9d745c2a4f4:/opt/conda/bin# ls python*
python  python3  python3-config  python3.9  python3.9-config
```

Mounting Data Onto a Container

Before we mount data onto a container, remember you initially created the python script `python_test.py` when creating your own container. Move `python_test.py` within `miniconda` directory. Now you have your `miniconda/python_test.py` outside the container. To access the files outside the container you should mount the file path along with the `docker run` command.

```
(base) ubuntu@osg-instance:~/miniconda$ mv ../python_test.py .
(base) ubuntu@osg-instance:~/miniconda$ ls
Dockerfile  python_test.py
```

To mount a host directory into your docker container, use the `-v` flag.

```
sudo docker run -v /host/directory/:/container/directory  -other-options
```

So the command for our example will be,

```
sudo docker run -v /home/ubuntu/:/home  -it py3-miniconda:2022-08 /bin/sh
```

Here we are mounting the $HOME directory `/home/ubuntu` from a host into containers' $HOME directory. Note that you may mount a particular directory according to your preference. The following shows the list of files in containers' $HOME directory with and without mounting.

Before mounting, there are no files found within the $HOME directory.

```
(base) ubuntu@osg-instance:~$ sudo docker run -it py3-miniconda:2022-08 /bin/bash
(base) root@e59ee96d6967:/# ls home
(base) root@e59ee96d6967:/#
```

No files within $HOME directory

After mounting using `-v` flag, files show up within the $HOME directory. The highlighted `miniconda` is our working directory with python script.

```
(base) ubuntu@osg-instance:~$ sudo docker run -v /home/ubuntu/:/home  -it py3-miniconda:2022-08 /bin/sh
# ls home
 Anaconda3-2021.11-Linux-x86_64.sh   anaconda3       hmmer                              rosetta_cont
'Example IO.zip'                     anvio_docker    miniconda                          singularity-
```

We can now execute the script, python_test.py using this command.

```
python python_test.py
```

```
# python python_test.py
[ 0 10 20 30 40]
[-5.  -4.5 -4.  -3.5 -3.  -2.5 -2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5
  2.   2.5  3.   3.5  4.   4.5]
[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5  5.   5.5  6.   6.5
  7.   7.5  8.   8.5  9.   9.5 10.  10.5 11.  11.5 12.  12.5 13.  13.5
 14.  14.5 15.  15.5 16.  16.5 17.  17.5 18.  18.5 19.  19.5 20. ]
(2.0, 2.220446049250313e-14)
```

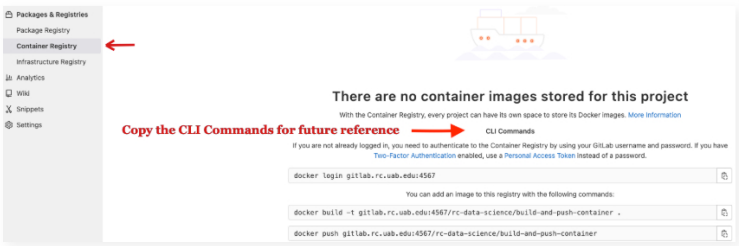
More lessons on Docker can be found in this link: Introduction to Docker and Docker Documentation.

Sharing Containers Using UAB GitLab Container Registry

If you prefer to share your container with a particular team/group, then the UAB GitLab container registry is the best and most secure option.

The following steps help you to create a container registry in UAB GitLab:

1. Create a UAB Gitlab account following the guidelines from the UAB GitLab page.
2. Create a `new_project` on UAB GitLab and click `Package and Registries`, and then go to `Container Registry`. Initially, the container registry looks empty because there are no container images in the registry.



> **Note**
>
> Copy these CLI commands for future reference. It contains commands (1) to login to your project UAB GitLab container registry (2) Add an image to the registry using the push/build command. We will use the `push` command as we already have the existing container in our system.

3. Login to UAB GitLab Registry using your `registry_name:ID` .

```
sudo docker login gitlab.rc.uab.edu:4567
```

> **Note**
>
> The registry_name and ID shown in this examples are for understanding and not meant for testing. Please use your GitLab registry name and ID for testing.

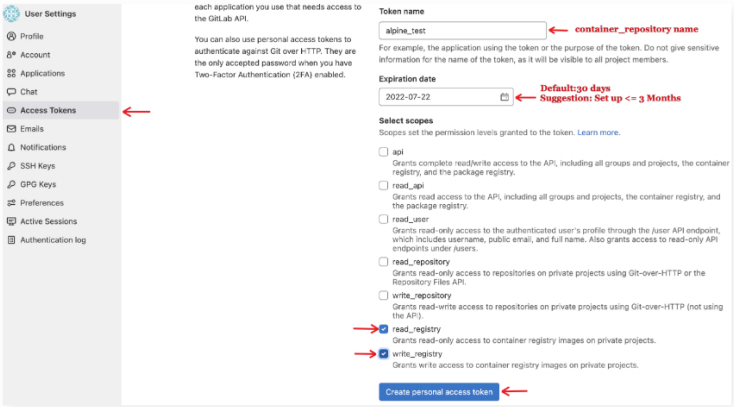
Note: For securing concerns, use an access token to log in. Create an access token in UAB GitLab to push/pull the docker container in the container registry (Secure token and guidelines to follow are shown next).

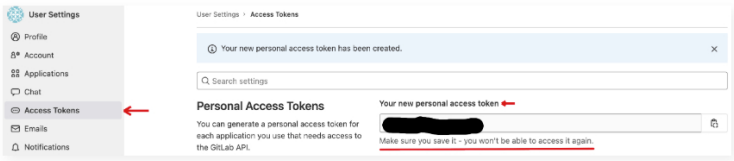
```
sudo docker login gitlab.rc.uab.edu:4567 -u username -p access_token
```

4. Creating an Access Token: From the UAB GitLab page, you can create an access token instead of using a password to log in to the UAB GitLab registry. Goto Edit profile -> Click `Access Tokens` . Then enter:

- **Token name**.:Suggestion: "container"_"repository-name"
- **Expiry date.** Suggestion: Default is 30 days. You can set your expiry date 3 months from the date you are making it.
- Under select scopes, check read and write registry (to push images to the registry) -> Then click `create personal access token` .

Once you create the token, copy the new personal access token since it's a one-time step and hard to retrieve after a refresh. Use the personal access token for login.






```
(base) ubuntu@osg-instance:~$ sudo docker login gitlab.rc.uab.edu:4567 -u [redacted] -p [redacted]
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

> **Warning**
>
> Running `docker login` leads to a warning message that your password is stored unencrypted in `/root/.docker/config.json` (or) `$HOME/.docker/config.json` . To ignore this warning, follow the instructions in this Github page or the Docker credentials store page.

Push Alpine Container from your System to UAB GitLab Container Registry

- List the docker images on your local computer using the `docker images` command. An `alpine` image exists already on this computer. Your container will likely have a different name.

```
sudo docker images
```

```
(base) ubuntu@osg-instance:~$ sudo docker images
REPOSITORY   TAG              IMAGE ID       CREATED        SIZE
alpine       latest           5274064a8e18   4 days ago     5.53MB
```

- Tag `alpine` to push into UAB GitLab registry. We need to have the UAB GitLab registry name to push. It will show the default command on the container registry page. Copy these commands for future reference. The tag is `test` here.

```
sudo docker tag alpine:latest gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
```

You can see the tag `test` associated with the `alpine` image.

```
sudo docker images
```

```
(base) ubuntu@osg-instance:~$ sudo docker images
REPOSITORY                                                        TAG       IMAGE ID       CREATED        SIZE
alpine                                                            latest    5274064a8e18   4 days ago     5.53MB
gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/...gitlab   test      5274064a8e18   4 days ago     5.53MB
```

The below first command is the syntax to push the Docker image to the UAB GitLab container registry from your computer. The second command is an example of pushing a Docker image to the UAB GitLab container registry from your computer.

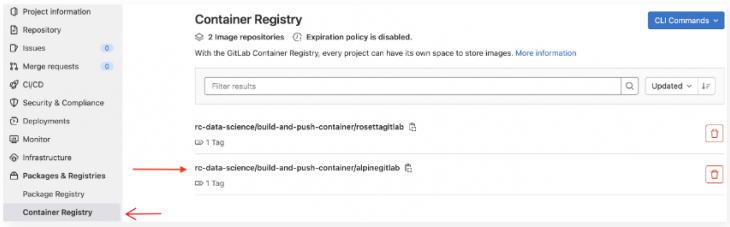
> **Note**
>
> The registry_name,ID, and gitlab_group_name shown in this examples are for understanding and not meant for testing. Please use your GitLab registry name and ID for testing.

```
sudo docker push gitlab_registry_name:ID/gitlab_group_name/project_name:tag

sudo docker push gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
```

```
(base) ubuntu@osg-instance:~$ sudo docker push gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
The push refers to repository [gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab]
24302eb7d908: Pushed
test: digest: sha256:1051244293a6e6f7d3916b03ec087e7a689bd6f9b5a4d254e87276e15d430f6d size: 528
```

In your GitLab's page container registry, refresh to view the `alpine` container is pushed to the registry.

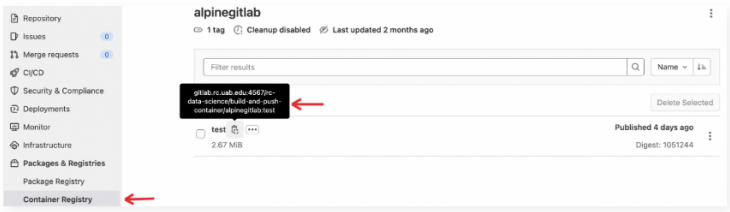


Now lets pull the `alpine` container from GitLab's page container registry to your system. Before that, remove the previous image from the system, which already has a `test` tag, to avoid discrepancies.

```
sudo docker rmi -f image_id
```

```
(base) ubuntu@osg-instance:~$ sudo docker images
REPOSITORY                                                        TAG       IMAGE ID       CREATED        SIZE
alpine                                                            latest    5274064a8e18   4 days ago     5.53MB
gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab   test      5274064a8e18   4 days ago     5.53MB
rosetta                                                           latest    803a01764166   6 weeks ago    12.3GB
<none>                                                            <none>    665449e49362   2 months ago   5.58MB
ubuntu                                                            bionic-20220315   b67d6ac264e4   4 months ago   63.2MB
(base) ubuntu@osg-instance:~$ sudo docker rmi -f 5274064a8e18
Untagged: alpine:latest
Untagged: gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
Untagged: gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab@sha256:1051244293a6e6f7d3916b03ec087e7a689bd6f9b5e4d254e
87276e15d430f6d
Deleted: sha256:5274064a8e18061e98553f01bca107d4d410bf2e4bc3756ca42277a67a58c4c
Deleted: sha256:8e51834b2b2cf1f60481113f79dc0d2ff3abe8138c3b6e960083fa013de19a75a
Deleted: sha256:24302eb7d9085da80f016e7e4ae55417e412fb7e0a8021e95e3b60c67cde557d
(base) ubuntu@osg-instance:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
rosetta      latest    803a01764166   6 weeks ago    12.3GB
<none>       <none>    665449e49362   2 months ago   5.58MB
ubuntu       bionic-20220315   b67d6ac264e4   4 months ago   63.2MB
```

`alpine` image removed



In your GitLab's page container registry, copy the pull command from the `test` container registry, and use it to pull the docker container to your system. You can see the image is reflected in the image list.

```
sudo docker pull gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
```

```
(base) ubuntu@osg-instance:~$ sudo docker pull gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
test: Pulling from rc-data-science/build-and-push-container/alpinegitlab
2408cc74d12b: Pull complete
Digest: sha256:1051244293a6e6f7d3916b03ec087e7a689bd6f9b5e4d254e87276e15d430f6d
Status: Downloaded newer image for gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab:test
```

```
(base) ubuntu@osg-instance:~$ sudo docker images
REPOSITORY                                                        TAG       IMAGE ID       CREATED        SIZE
gitlab.rc.uab.edu:4567/rc-data-science/build-and-push-container/alpinegitlab   test      5274064a8e18   4 days ago     5.53MB
```

Last update: October 19, 2022

IPR2025-01084
Orca Security EX1045 Page 6


Using Git
Getting Software from GitHub and GitLab
Copyright © 2021-2022 The University of Alabama at Birmingham.
Still stuck after reading? Email us at support@listserv.uab.edu
Made with Material for MkDocs