Introduction to Containerization (Docker and Docker Compose)

Ritu Arora

Nov 14, 2022

With Contributions from Charlie Dey from Texas Advanced Computing Center

WHAT IS A CONTAINER?

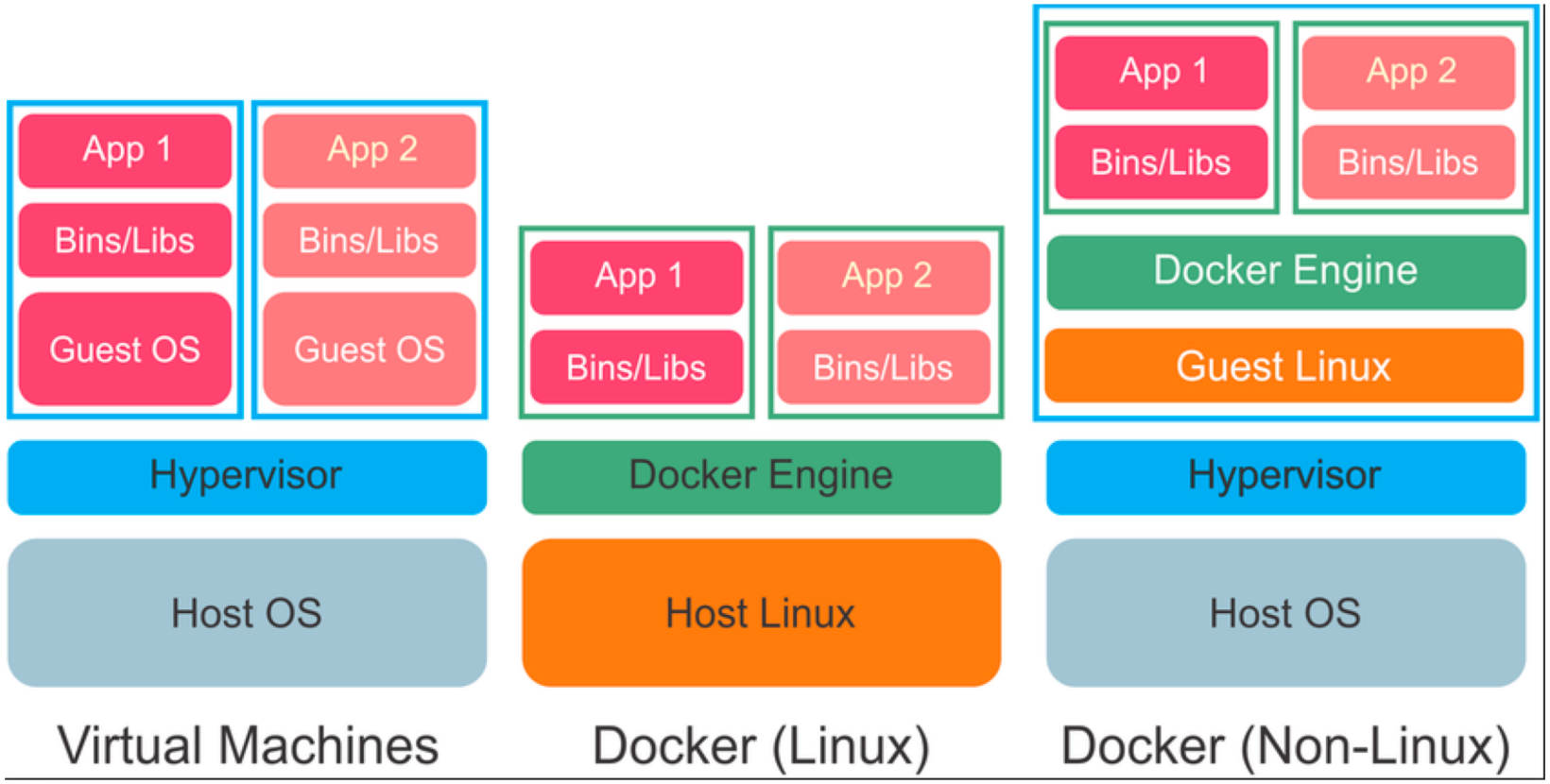
Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Examples: Docker, Singularity

ADVANTAGES OF CONTAINERIZATION

- Help in **deploying future-proof applications** by creating packages that are almost self-contained
- Make it **convenient to distribute production and trial versions of the code** that can run on the customers' devices without requiring an application-specific installation and configuration, hence, helps in developing **a scalable software distribution model**
- **Save time in complicated installs** - build a Docker application - push it to Docker Hub - and **reuse** it on any number of systems as you desire
- **Mitigate the portability issues related to the applications** - Dockerized applications can be ported conveniently across different cloud computing service providers - of course, you may need to install Docker and additional tools to run the Docker container depending upon the system that you are on
- **Help in doing reproducible science**

CONTAINER VERSUS VIRTUAL MACHINES



Source: GUIdock: Using Docker Containers with a Common Graphics User Interface to Address the Reproducibility of Research - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/A-comparison-of-the-architecture-of-virtual-machines-and-Docker-software_fig4_299771559 [accessed 24 Sep, 2018]

WHAT IS DOCKER?

- Docker is an open-source project designed for creating, deploying, and running applications by using containers.
- Three important concepts related to Docker that we will cover today:
    - Docker container
        - As mentioned on slide # 1: containers are used to package an application along with all its dependencies (such as runtime system, libraries, and data) into a single unit
        - These containers are portable, self-sufficient and can be run on any infrastructure in the cloud or on-premises on which Docker is installed/supported
    - Docker image
    - Dockerfile

HOW POPULAR IS DOCKER?

Source: docker.com [accessed November 11, 2022]

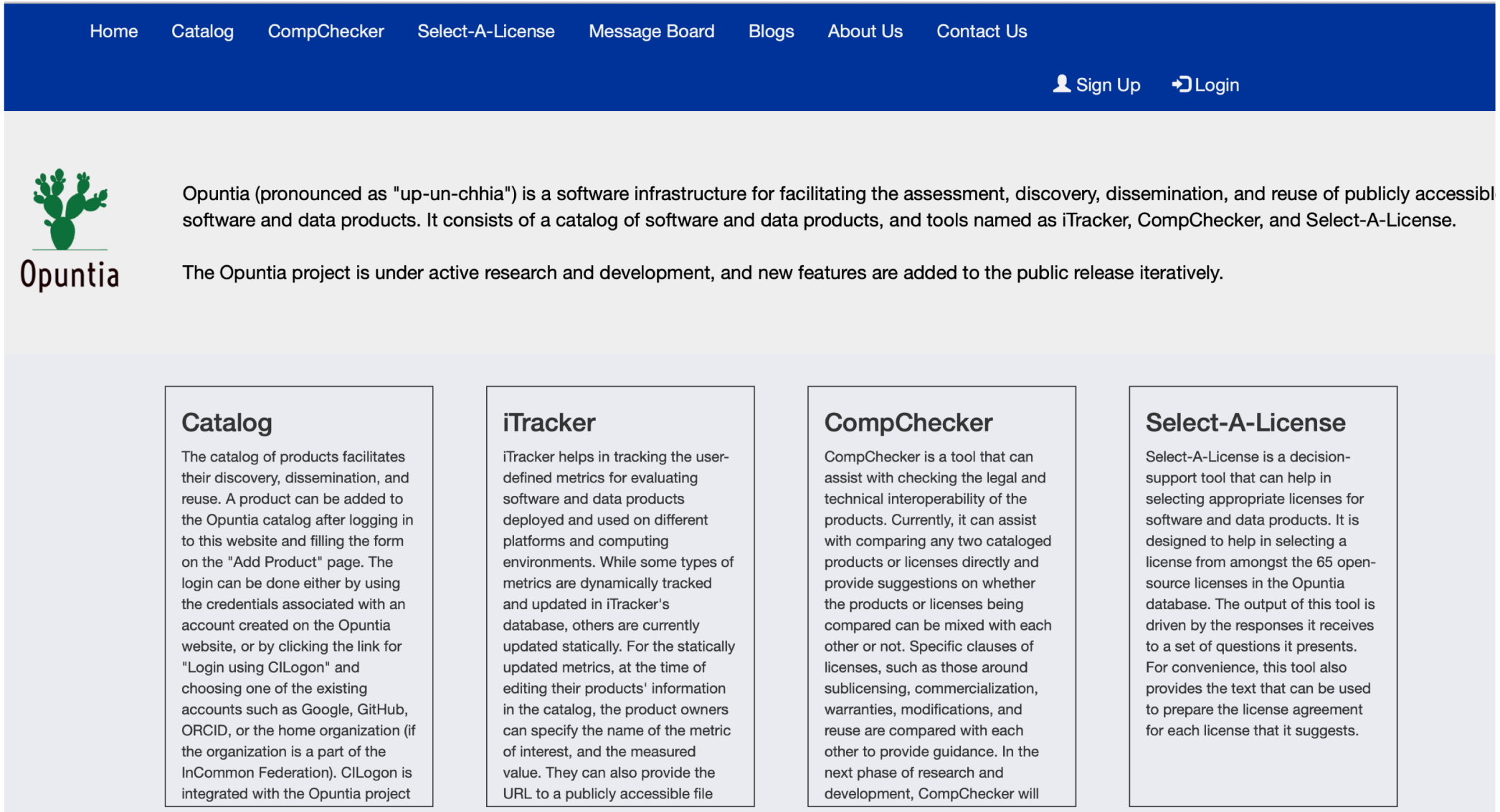
WHAT IS THE PROCESS FOR BUILDING DOCKER IMAGES?

1. Install Docker
2. Write a Dockerfile
    + **Dockerfile** is a text file that contains instructions, using which, Docker can build images automatically
    + Name of the file: Dockerfile
3. Build the Docker image and tag it
    + A file containing the snapshot of a container is known as a **Docker image**
    + It is created using the build command, and produces a container when it starts running
    + You can add tags to the images during the build step or while saving it to a repository - the default tag is “latest”
4. Run the image that you built
5. Register on DockerHub - use the credentials to push the image
    + Images are stored in a Docker registry such as registry.hub.docker.com
    + Pull Images later on any system that you want to run the application on

HOW CAN CONTAINERS BE CREATED FROM DOCKER IMAGES?

- You can use Docker commands such as “docker pull” or “docker run”
  - + “Pull” will always fetch the latest version of the image from Docker Hub before running
  - + “Run” will search for an image locally and run it, and if there is nothing available locally, it will go to Docker Hub.
- We will learn more about this topic during the hands-on session

SOME PROJECTS USING DOCKER (1)



SOME PROJECTS USING DOCKER(2)

Getting Docker

A Docker Image is a container template from which one or more containers can be run. It is a rooted filesystem that, by definition, contains all of the file dependencies needed for whatever application(s) will be run within the containers launched from it. The image also contains metadata describing options available to the operator running containers from the image.

One of the great things about Docker is that a lot of software has already been packaged into Docker images. One source of 100s of thousands of public images is the official **Docker Hub**:

https://hub.docker.com

The Docker Hub contains images contributed by individual users and organizations as well as "official images".

Hands-On Session

STEP 0: SET-UP

Create a Docker ID: https://hub.docker.com/signup

Pick one of the following options for proceeding:

Option # 1: Use your own laptop with Docker installed on it

Option # 2: Use an instance of a VM, provisioned with Docker, on a cloud computing platform

Option # 3: Use Docker's online platform for learning about Docker: https://tinyurl.com/y9sg7eq7

STEP 1: RUNNING PRE-BUILT DOCKER CONTAINERS (1)

Please go to “Step 1” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.38 ~
$ docker run alpine date
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
050382585609: Pull complete
Digest: sha256:6a92cd1fcdc8d8cdec60f33dda4db2cb1fcdcacf3410a8e05b3741f44a9b5998
Status: Downloaded newer image for alpine:latest
Thu Jul 25 21:14:59 UTC 2019
[node1] (local) root@192.168.0.38 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               N
AMES
[node1] (local) root@192.168.0.38 ~
$ docker container ls --all
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
       NAMES
041179d7c93b        alpine              "date"              41 seconds ago      Exited (0) 39 seconds ago
       funny_dirac
[node1] (local) root@192.168.0.38 ~
$
```

STEP 1: RUNNING PRE-BUILT DOCKER CONTAINERS (2)

Please go to “Step 1” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.38 ~
$ docker run --interactive --tty alpine /bin/sh
/ #
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # exit
[node1] (local) root@192.168.0.38 ~
$ docker container ls --all
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
      NAMES
b12478fb3f94        alpine              "/bin/sh"           15 seconds ago      Exited (0) 7 seconds ago
      determined_lalande
041179d7c93b        alpine              "date"              9 minutes ago       Exited (0) 9 minutes ago
      funny_dirac
[node1] (local) root@192.168.0.38 ~
$ docker start -a -i b12478fb3f94
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # exit
[node1] (local) root@192.168.0.38 ~
$
```

STEP 1: RUNNING PRE-BUILT DOCKER CONTAINERS (3)

Please go to “Step 1” at the following link: https://tinyurl.com/5n939rhm

```
$ docker run --interactive --tty --rm ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
7413c47ba209: Pull complete
0fe7e7cbb2e8: Pull complete
1d425c982345: Pull complete
344da5c95cec: Pull complete
Digest: sha256:c303f19cfe9ee92badbbbd7567bc1ca47789f79303ddcef56f77687d4744cd7a
Status: Downloaded newer image for ubuntu:latest
root@cd2f17c662c2:/# exit
exit
[node1] (local) root@192.168.0.38 ~
$ docker container ls --all
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
       NAMES
b12478fb3f94        alpine              "/bin/sh"           15 minutes ago      Exited (0) 14 minutes ago
       determined_lalande
041179d7c93b        alpine              "date"              25 minutes ago      Exited (0) 25 minutes ago
       funny_dirac
[node1] (local) root@192.168.0.38 ~
$
```

STEP 1: RUNNING PRE-BUILT DOCKER CONTAINERS (4)

Please go to “Step 1” at the following link: https://tinyurl.com/5n939rhm

```
RRC02C92WZMD6NIP01:democontainer vrv207$ docker run --interactive --tty --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
e96e057aae67: Pull complete
Digest: sha256:4b1d0c4a2d2aaf63b37111f34eb9fa89fa1bf53dd6e4ca954d47caebca4005c2
Status: Downloaded newer image for ubuntu:latest
root@f82f11b92a0d:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@f82f11b92a0d:/# date
Fri Nov 11 18:45:48 UTC 2022
root@f82f11b92a0d:/# exit
exit
RRC02C92WZMD6NIP01:democontainer vrv207$ docker start -a -i f82f11b92a0d
Error: No such container: f82f11b92a0d
```

STEP 2: PACKAGE AND RUN AN APPLICATION USING DOCKER (1)

Please go to “Step 2” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.38 ~
$ vi Dockerfile
[node1] (local) root@192.168.0.38 ~
$ cat Dockerfile
FROM alpine
CMD ["echo", "hello world!"]
[node1] (local) root@192.168.0.38 ~
$ docker build -t app1 .
Sending build context to Docker daemon  1.748MB
Step 1/2 : FROM alpine
 ---> b7b28af77ffe
Step 2/2 : CMD ["echo", "hello world!"]
 ---> Using cache
 ---> 020925a69856
Successfully built 020925a69856
Successfully tagged app1:latest
[node1] (local) root@192.168.0.38 ~
$
```

STEP 2: PACKAGE AND RUN AN APPLICATION USING DOCKER (2)

Please go to “Step 2” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.38 ~
$ docker run --name hello_world app1
hello world!
[node1] (local) root@192.168.0.38 ~
$ docker start -a -i hello_world
hello world!
[node1] (local) root@192.168.0.38 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
app1                latest              020925a69856        14 minutes ago      5.58MB
alpine              latest              b7b28af77ffe        2 weeks ago         5.58MB
[node1] (local) root@192.168.0.38 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
          NAMES
[node1] (local) root@192.168.0.38 ~
$
```

STEP 3: ADDING VOLUME (1)

Please go to “Step 3” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.43 ~
$ vi Dockerfile
[node1] (local) root@192.168.0.43 ~
$ cat Dockerfile
FROM ubuntu
RUN mkdir -p ubuntu1 && cd ubuntu1 && echo "hello hello bye bye" >> file
VOLUME /ubuntu1
CMD /bin/sh
[node1] (local) root@192.168.0.43 ~
$
```

STEP 3: ADDING VOLUME (2)

Please go to “Step 3” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.43 ~
$ docker build .
Sending build context to Docker daemon  1.748MB
Step 1/4 : FROM ubuntu
latest: Pulling from library/ubuntu
7413c47ba209: Pull complete
0fe7e7cbb2e8: Pull complete
1d425c982345: Pull complete
344da5c95cec: Pull complete
Digest: sha256:c303f19cfe9ee92badbbbd7567bc1ca47789f79303ddcef56f77687d4744cd7a
Status: Downloaded newer image for ubuntu:latest
 ---> 3556258649b2
Step 2/4 : RUN mkdir -p ubuntu1 && cd ubuntu1 && echo "hello hello bye bye" >> file
 ---> Running in 11cec2531d99
Removing intermediate container 11cec2531d99
 ---> 57a0103c4f45
Step 3/4 : VOLUME /ubuntu1
 ---> Running in 087ca99ff0b6
Removing intermediate container 087ca99ff0b6
 ---> be1a8b07b358
Step 4/4 : CMD /bin/sh
 ---> Running in f92efa146fa4
Removing intermediate container f92efa146fa4
 ---> 239c5b4b5ed6
Successfully built 239c5b4b5ed6
[node1] (local) root@192.168.0.43 ~
$
```

STEP 3: ADDING VOLUME (3)

Please go to “Step 3” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.43 ~
$ docker build .
Sending build context to Docker daemon  1.748MB
Step 1/4 : FROM ubuntu
latest: Pulling from library/ubuntu
7413c47ba209: Pull complete
0fe7e7cbb2e8: Pull complete
[node1] (local) root@192.168.0.43 ~
$ docker run --rm -it 239c5b4b5ed6
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  ubuntu1  usr  var
# exit
[node1] (local) root@192.168.0.43 ~
$ mkdir src
[node1] (local) root@192.168.0.43 ~
$ cd src
[node1] (local) root@192.168.0.43 ~/src
$ vi Hello.java
[node1] (local) root@192.168.0.43 ~/src
$ cat Hello.java
public class Hello { public static void main(String... ignored) { System.out.println("Hello, World!"); } }
[node1] (local) root@192.168.0.43 ~/src
$ vi Dockerfile
[node1] (local) root@192.168.0.43 ~/src
$ cat Dockerfile
FROM openjdk:8u131-jdk-alpine
WORKDIR /src
ENTRYPOINT javac Hello.java && java Hello
[node1] (local) root@192.168.0.43 ~/src
$
```

STEP 3: ADDING VOLUME (4)

Please go to “Step 3” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.43 ~/src
$ docker build -t my-openjdk .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM openjdk:8u131-jdk-alpine
8u131-jdk-alpine: Pulling from library/openjdk
1160f4abea84: Pull complete
b1b3e089ad5b: Pull complete
4220f7d94f04: Pull complete
Digest: sha256:01655aeb8f29002d40e75d25144d0b61b6e455f9d6469b4016eb56c5f43dbb99
Status: Downloaded newer image for openjdk:8u131-jdk-alpine
 ---> a99736768b96
Step 2/3 : WORKDIR /src
 ---> Running in 1c3eacc0ed61
Removing intermediate container 1c3eacc0ed61
 ---> fe69b6d65e3f
Step 3/3 : ENTRYPOINT javac Hello.java && java Hello
 ---> Running in e758c39eeb44
Removing intermediate container e758c39eeb44
 ---> 13a1987288fa
Successfully built 13a1987288fa
Successfully tagged my-openjdk:latest
```

STEP 3: ADDING VOLUME (5)

Please go to “Step 3” at the following link: https://tinyurl.com/5n939rhm

```
[node1] (local) root@192.168.0.43 ~/src
$ cd ..
[node1] (local) root@192.168.0.43 ~
$ docker run --rm -it -v $(pwd)/src:/src my-openjdk
Hello, World!
[node1] (local) root@192.168.0.43 ~
$ vi src/Hello.java
[node1] (local) root@192.168.0.43 ~
$ cat src/H
Hello.class  Hello.java
[node1] (local) root@192.168.0.43 ~
$ cat src/Hello.java
public class Hello { public static void main(String... ignored) { System.out.println("Hello, World from GHC18!"); } }
[node1] (local) root@192.168.0.43 ~
$ docker run --rm -it -v $(pwd)/src:/src my-openjdk
Hello, World from GHC18!
[node1] (local) root@192.168.0.43 ~
$
```

What is Docker Compose?

- Compose is a tool for defining and running multi-container Docker applications
- With Compose, you use a YAML file to configure your application’s services
- Then, with a single command, you create and start all the services from your configuration

Using Docker Compose

Using Compose is a three-step process:

- Define images with Dockerfiles
- Define the services in a `docker-compose.yml` files as containers with all of your options (image, port mapping, links, etc.)
- Run `docker-compose up` and Compose starts and runs your entire app

Three step process to use … a bit more to actually build

Hands-On Session

We're going to run two containers:

- Redis

- Flask

You will need two terminal sessions - one for running the containers, one for executing commands like `curl`

Step 1, The Docker File

- We are going to need a Docker Container to run Redis CLI as well as a Flask application

- We are going to build from the `python:3-onbuild` image

```
FROM python:3-onbuild
```

Step 1, The Docker File

- We are going to install the Redis CLI

```
RUN apt-get update
RUN apt-get install -y redis-tools
```

Step 1, The Docker File

- We are going to expose port 5000

```
EXPOSE 5000
```

Step 1, The Docker File

- and we're going to execute `main.py`

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

The complete Dockerfile

```
FROM python:3-onbuild
RUN apt-get update
RUN apt-get install -y redis-tools
EXPOSE 5000
CMD ["python", "./main.py"]
```

Create a requirements.txt file

Add the following to the file:

```
flask
redis
```

The file should look like this:

```
$ cat requirements.txt

flask
redis
```

Create a main.py file with the contents below

```
from flask import Flask
from redis import Redis

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

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


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

Build the Docker image with this command

```
docker build -t compose-flask .
```

Create a docker-compose.yml file

```
version: '3'
services:
  flask:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Run the docker-compose command

```
docker-compose up
```

You would see something like this in your terminal window after running docker-compose

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix the
m
[+] Running 3/3
 ⠿ Network democontainer_default    Created                                                    0.0s
 ⠿ Container democontainer-redis-1  Created                                                    0.1s
 ⠿ Container democontainer-flask-1  Created                                                    0.1s
Attaching to democontainer-flask-1, democontainer-redis-1
democontainer-redis-1  | 1:C 28 Oct 2022 00:25:00.079 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0
Oo
democontainer-redis-1  | 1:C 28 Oct 2022 00:25:00.079 # Redis version=7.0.5, bits=64, commit=000000
00, modified=0, pid=1, just started
democontainer-redis-1  | 1:C 28 Oct 2022 00:25:00.079 # Warning: no config file specified, using th
e default config. In order to specify a config file use redis-server /path/to/redis.conf
democontainer-redis-1  | 1:M 28 Oct 2022 00:25:00.080 * monotonic clock: POSIX clock_gettime
democontainer-redis-1  | 1:M 28 Oct 2022 00:25:00.081 * Running mode=standalone, port=6379.
democontainer-redis-1  | 1:M 28 Oct 2022 00:25:00.081 # Server initialized
democontainer-redis-1  | 1:M 28 Oct 2022 00:25:00.082 * Ready to accept connections
democontainer-flask-1  |  * Serving Flask app 'main' (lazy loading)
democontainer-flask-1  |  * Environment: production
democontainer-flask-1  |    WARNING: This is a development server. Do not use it in a production de
ployment.
democontainer-flask-1  |    Use a production WSGI server instead.
democontainer-flask-1  |  * Debug mode: on
democontainer-flask-1  |  * Running on all addresses.
democontainer-flask-1  |    WARNING: This is a development server. Do not use it in a production de
ployment.
democontainer-flask-1  |  * Running on http://172.18.0.3:5000/ (Press CTRL+C to quit)
democontainer-flask-1  |  * Restarting with stat
democontainer-flask-1  |  * Debugger is active!
democontainer-flask-1  |  * Debugger PIN: 135-698-799
```

Testing

- Look at your container names
- Log in to the Flask container with the following command after replacing `<container>` with your actual container id

```
docker exec -it <container> bash
```

- Try pinging the Redis container from there with: `ping redis`

Open a second terminal window & run these commands

docker ps

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS
NAMES
4b3cb98c9d25   democontainer-flask    "python ./main.py"       15 minutes ago   Up 15 minutes
0.0.0.0:5000->5000/tcp   democontainer-flask-1
d763e83d07cc   redis:alpine           "docker-entrypoint.s…"   15 minutes ago   Up 15 minutes   6379/tcp
democontainer-redis-1
```

docker exec -it democontainer-flask-1 bash

```
root@4b3cb98c9d25:/usr/src/app# ping redis
PING redis (172.18.0.2) 56(84) bytes of data.
64 bytes from democontainer-redis-1.democontainer_default (172.18.0.2): icmp_seq=1 ttl=64 time=1.74 ms
64 bytes from democontainer-redis-1.democontainer_default (172.18.0.2): icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from democontainer-redis-1.democontainer_default (172.18.0.2): icmp_seq=3 ttl=64 time=0.118 ms
64 bytes from democontainer-redis-1.democontainer_default (172.18.0.2): icmp_seq=4 ttl=64 time=0.113 ms
64 bytes from democontainer-redis-1.democontainer_default (172.18.0.2): icmp_seq=5 ttl=64 time=0.110 ms
64 bytes from democontainer-redis-1.democontainer_default (172.18.0.2): icmp_seq=6 ttl=64 time=0.117 ms
```

Stop and Restart the Containers

- You can stop your running containers with Ctrl-C
- You can restart them with “docker-compose up”
- You can also rebuild them if necessary with “docker-compose build”

Now when you run the curl command in the second terminal window, you will see the counter reset for page views

```
$ curl localhost:5000
This Compose/Flask demo has been viewed '1' time(s).
```

Run the curl command in the second terminal window to communicate with the Flask container

```
$ curl localhost:5000
This Compose/Flask demo has been viewed '1' time(s).

$ curl localhost:5000
This Compose/Flask demo has been viewed '2' time(s)

$ curl localhost:5000
This Compose/Flask demo has been viewed '3' time(s)
```

You will see the messages as follows printed in the first terminal window that has the server running

```
democontainer-flask-1  | 172.18.0.1 - - [28/Oct/2022 00:37:46] "GET / HTTP/1.1" 200 -
democontainer-flask-1  | 172.18.0.1 - - [28/Oct/2022 00:37:50] "GET / HTTP/1.1" 200 -
democontainer-flask-1  | 172.18.0.1 - - [28/Oct/2022 01:02:27] "GET / HTTP/1.1" 200
```

To stop the container and reset the Redis database before restarting try the following

```
$ docker-compose down
[+] Running 3/2
 ⠿ Container democontainer-redis-1  Removed
0.2s
 ⠿ Container democontainer-flask-1  Removed
0.2s
 ⠿ Network democontainer_default    Removed
0.1s


$  docker-compose up -d
[+] Running 3/3
 ⠿ Network democontainer_default    Created
0.0s
 ⠿ Container democontainer-redis-1  Started
0.4s
 ⠿ Container democontainer-flask-1  Started
0.4s
```

Docker and Singularity

- Docker has become extremely popular for both applications and services, but using the Docker daemon requires elevated privileges, making it a security risk for shared servers.
- Giving users direct access to the “docker” command on a host allows users to trivially use docker to obtain root-level access on the host
- Singularity was designed to run without root privileges while also providing access to host devices, making it a good fit for traditional HPC environments.

Singularity vs Docker

		Singularity	Docker
1	• Edit and run containers • Interact with host devices and filesystems	✓	✓
2	• Runs without sudo	✓	✗
3	• Runs as host user	✓	✗
4	• Can become root in containers	✗	✓
5	• Control network interfaces	✗	✓
6	• Configurable for advanced security	✓	✗

Source: *https://tacc.github.io/CSC2018Institute/docs/day5/singularity.html*

Singularity File

```
Bootstrap: docker
From: ubuntu:16.04

%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
```

Source: https://singularity.lbl.gov/docs-docker

Singularity from Docker

```
singularity shell docker://ubuntu:latest
singularity run docker://ubuntu:latest
singularity exec docker://ubuntu:latest echo "Hello Dinosaur!"

singularity pull docker://ubuntu:latest
singularity build ubuntu.img docker://ubuntu:latest
```

Source: https://singularity.lbl.gov/docs-docker

Thanks!

Any questions, comments, or concerns?

https://github.com/ritua2/Basil/tree/main/training