Introduction to Containerization (Docker, Docker Compose, and Singularity)

Ritu Arora

Nov 14, 2022

With Contributions from Charlie Dey, Carlos Redondo, and other colleagues from the
Texas Advanced Computing Center

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 .
```

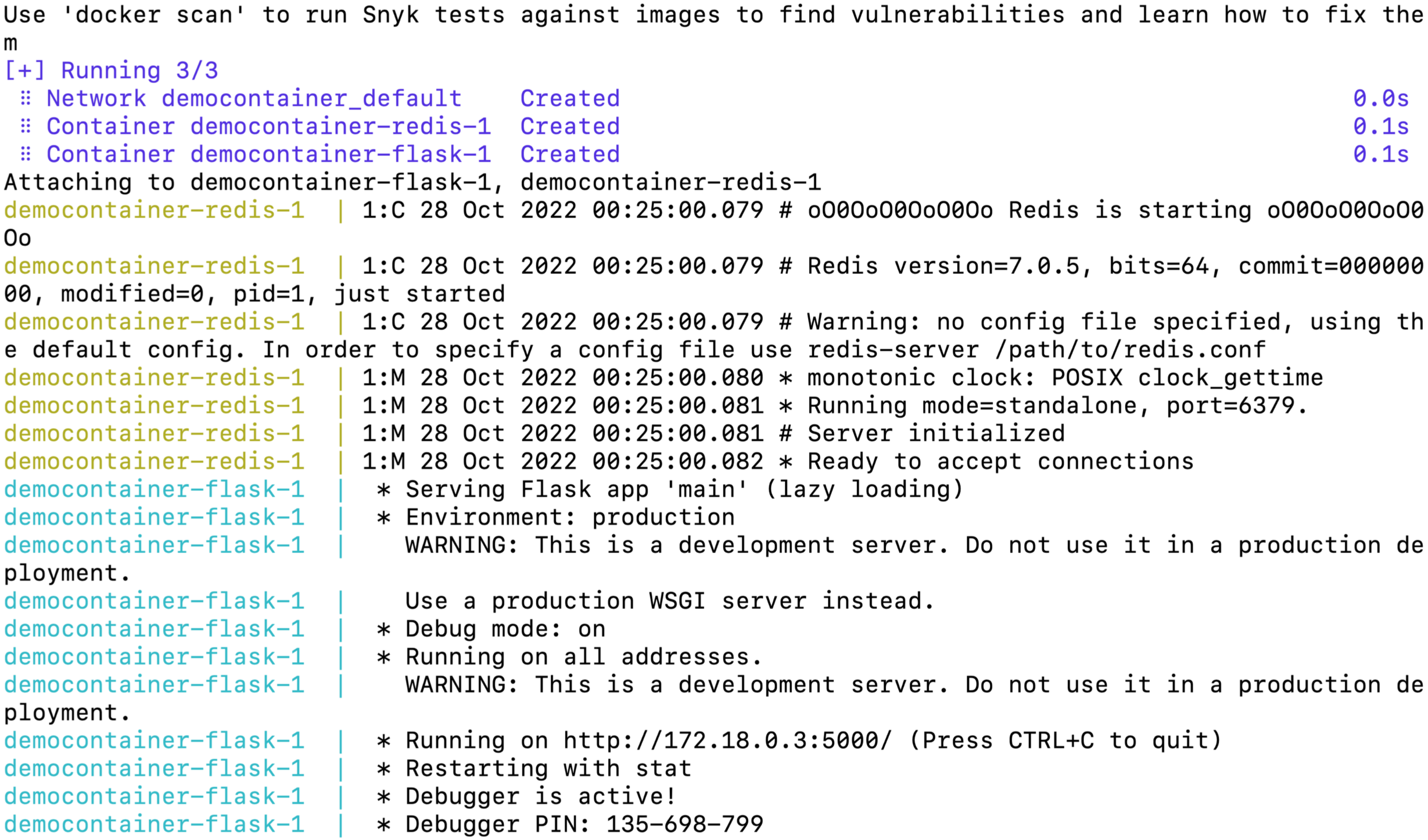
Create a docker-compose.yml file

```
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 (root) privileges, making it a security risk for shared servers.
- Giving users root access to run “docker” commands on a host allows them to 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
- You can use Singularity to pull Docker images and convert them into Singularity Image Format (SIF) for running on HPC systems

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*

Using Singularity on an HPC System (1)

- Switch to a compute node and load the Singularity module
  - Use the "**srun**" command on Arc

```
[login002]$ ml singularity
Lmod has detected the following error:

 This module is not available on the login nodes.
 To use this module please connect directly to a node using the "srun"
command.

[login002]$ srun -p compute1 -n 1 -t 02:00:00 --pty bash

[c034]$ ml singularity
The singularity  module version 3.10.3  is loaded
```

Using Singularity on an HPC System (2)

```
$ singularity run docker://godlovedc/lolcow

INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Getting image source signatures
Copying blob 9fb6c798fa41 done
Copying blob 8e860504ff1e done
Copying blob d010c8cf75d7 done
Copying blob 9d99b9777eb0 done
Copying blob 3b61febd4aef done
Copying blob 7fac07fb303e done
Copying config 73d5b1025f done
Writing manifest to image destination
Storing signatures
2022/11/13 13:58:44  info unpack layer:
sha256:9fb6c798fa41e509b58bccc5c29654c3ff4648b608f5daa67c1aab6a7d02c118
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Using Singularity on an HPC System (3)

```
   $ singularity run docker://godlovedc/lolcow

   INFO:    Converting OCI blobs to SIF format
   INFO:    Starting build...
   Getting image source signatures
   Copying blob 9fb6c798fa41 done
   Copying blob 8e860504ff1e done
   Copying blob d010c8cf75d7 done
   Copying blob 9d99b9777eb0 done
   Copying blob 3b61febd4aef done
   Copying blob 7fac07fb303e done
   Copying config 73d5b1025f done
   Writing manifest to image destination
   Storing signatures
   2022/11/13 13:58:44  info unpack layer:
sha256:9fb6c798fa41e509b58bccc5c29654c3ff4648b608f5daa67c1aab6a7d02c118
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Using Singularity on an HPC System (4)

```
    $ singularity run docker://godlovedc/lolcow

    INFO:    Converting OCI blobs to SIF format
    INFO:    Starting build...
    Getting image source signatures
    Copying blob 9fb6c798fa41 done
    …
    Writing manifest to image destination
    Storing signatures
    2022/11/13 13:58:44  info unpack layer:
sha256:9fb6c798fa41e509b58bccc5c29654c3ff4648b608f5daa67c1aab6a7d02c118
    …
    _______________________________________
    / I must have a prodigious quantity of  \
    | mind; it takes me as much as a week   |
    | sometimes to make it up.              |
    |                                       |
    \ -- Mark Twain, "The Innocents Abroad" /
     ---------------------------------------
            \   ^__^
             \  (oo)\_______
                (__)\       )\/\
                    ||----w |
                    ||     ||
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Using Singularity on an HPC System (5)

```
$ /home/abc123/.singularity/cache/oci-tmp/sha256.a692b57abc43035b197b10390ea2c12855d21649f2ea2cc28094d18b93360eeb
 _____________________________________
/ Beware of a tall black man with one \
\ blond shoe.                         /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||


$ singularity exec docker://godlovedc/lolcow fortune
INFO:    Using cached SIF image
After your lover has gone you will still have PEANUT BUTTER!
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Using Singularity on an HPC System (6)

- Using interactive shell session with Singularity

```
$ singularity shell docker://godlovedc/lolcow
INFO:    Using cached SIF image
Singularity> cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.3 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.3 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
Singularity>
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Using Singularity on an HPC System (7)

- Using the pull command, a local copy of the SIF file is created

```
$  singularity pull docker://godlovedc/lolcow
INFO:    Using cached SIF image

[c034:]$ ls
lolcow_latest.sif

[c034:]$ ./lolcow_latest.sif
 _________________________
/ Don't Worry, Be Happy. \
|                         |
\ -- Meher Baba           /
 -------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Using Singularity on an HPC System (8)

- Singularity definition file example

```
$ cat testingsifdef.def
Bootstrap: library
       From: alpine:latest
%runscript
   echo "Running the container - hellow world!"

%post
   echo "Now inside the container"
   yum -y install vim-minimal
```

- You will need to build this on a system on which you have “root” access

```
$ sudo singularity build testingsifdef.sif testingsifdef.def
```

Source: https://docs.sylabs.io/guides/3.0/user-guide/singularity_and_docker.html

Thanks!

Any questions, comments, or concerns?

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