GETTING STARTED WITH DOCKER AND SWARM

Prepared by Matthew Cengia

- Email: mattcen@mattcen.com
- Twitter: @mattcen
- Mastodon: @mattcen@aus.social
- Matrix: mattcen:matrix.org

ACKNOWLEDGEMENT OF COUNTRY

WHY ARE WE TALKING ABOUT DOCKER?

WHAT IS DOCKER?

WAIT, WHAT ARE CONTAINERS?

- Not virtual machines, but similar in some ways
- A combination of:
  - A disk image/filesystem chroot
  - Namespaces (what you can see, e.g. filesystem, processes)
  - Cgroups (what you can do, e.g. access to system resources like devices)
- Can be created with a handful of standard Linux commands

More detail in Liz Rice's talk "*Building a container from scratch in Go*":
https://youtu.be/Utf-A4rODH8

HOW CAN DOCKER AND CONTAINERS HELP?

- Consistent environment
- Segregation between apps
- Sandboxing and limiting privileges
- Isolating code and data
- Simplify deployment
- Note: "containerising" apps needn't be a big unmanageable project

HUH?

INSTALLING DOCKER

On most Redhat- or Debian-based systems, you can do this:

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
$ sudo usermod -aG docker "$USER
```

Remember to log out and back in for the user group change to take effect

IF YOU WANT TO PLAY ALONG

1. Create a Docker ID at https://hub.docker.com/signup
2. Browse to https://labs.play-with-docker.com/ and start a session
3. Click "Add new instance"
4. `git clone https://github.com/mattcen/lca2021-docker-talk`
5. `cd lca2021-docker-talk/demos`

RUNNING OUR FIRST CONTAINER

Once Docker is running, we can do something like this:

```
$ docker run --rm -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
Digest: sha256:c95a8e48bf88e9849f3e0f723d9f49fa12c5a00cfc6e60d2bc99d87555295e4c
Status: Downloaded newer image for ubuntu:latest
root@6e4f7117321d:/# ps -efa
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:13 pts/0    00:00:00 bash
root         8     1  0 02:13 pts/0    00:00:00 ps -efa
root@6e4f7117321d:/# exit
$
```

HOW ARE CONTAINER IMAGES CREATED?

- Filesystems that are usually built using a `Dockerfile`
- Can be based off other images
  - Creates "layers"
  - encouraging reusability
- Some images, like Ubuntu, aren't based off other images, but built from scratch

DOCKERFILE

A basic `Dockerfile` might be:

```
FROM ubuntu:20.04
CMD echo "Hello World!"
```

`docker build` looks for files called `Dockerfile` by default:

```
$ cd ~/lca2021-docker-talk/demos/docker_file
$ ls -l
total 4
-rw-r--r--    1 docker   staff           43 Jan 20 02:38 Dockerfile
$
```

BUILDING A DOCKER IMAGE

The `-t` option to `docker build` 'tags' the image with the name 'helloworld'

```
$ docker build . -t helloworld
Sending build context to Docker daemon  25.09kB
Step 1/2 : FROM ubuntu:20.04
 ---> f643c72bc252
Step 2/2 : CMD echo "Hello World!"
 ---> Running in aca2515e7c8a
Removing intermediate container aca2515e7c8a
 ---> 2e1213f6db18
Successfully built 2e1213f6db18
Successfully tagged helloworld:latest
```

We then run a container based on that image with `docker run` again

```
$ docker run --rm -it helloworld
Hello World!
$
```

ANOTHER DOCKERFILE

A more useful container might be:

```
$ cd ~/lca2021-docker-talk/demos/another_docker_file
$ ls -l
total 12
-rw-r--r--    1 docker   staff           70 Jan 20 02:57 Dockerfile
-rw-r--r--    1 docker   staff           25 Jan 20 02:46 index.html
-rw-r--r--    1 docker   staff          243 Jan 20 02:52 webserver.py
```

Dockerfile:

```
FROM python:3.9.1-buster
COPY . .
EXPOSE 8000
CMD python webserver.py
```

index.html:

```
<!DOCTYPE html>
<html><body>
<p>Hello World from Python!</p>
</body></html>
```

webserver.py:

```
import http.server
import socketserver

PORT = 8000
Handler = http.server.SimpleHTTPRequestHandler
with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    httpd.serve_forever()
```

We can put all this together into an image called 'hellopython' with:

```
$ docker build . -t hellopython
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM python:3.9.1-buster
 ---> da24d18bf4bf
Step 2/4 : COPY . .
 ---> 2e0b56c961fb
Step 3/4 : EXPOSE 8000
 ---> Running in 3e08636e7cb6
Removing intermediate container 3e08636e7cb6
 ---> c4079aa09149
Step 4/4 : CMD python webserver.py
 ---> Running in e4fb064f6438
Removing intermediate container e4fb064f6438
 ---> 74fb999e2f10
Successfully built 74fb999e2f10
Successfully tagged hellopython:latest
```

And then when we run it:

```
$ docker run --rm -d -p 8000:8000 hellopython
a2c890dd24c63852d373598ed934fca0f66b4605f8b8e19a244f2011925d3016
```

And test it:

```
$ curl http://localhost:8000
<!DOCTYPE html>
<html><body>
<p>Hello World from Python!</p>
</body></html>
```

And then we can stop (and, because of `--rm` above, remove) the container:

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
a2c890dd24c6        hellopython         "/bin/sh -c 'python …"   15 seconds ago      Up 15
$ docker container stop a2c
a2c
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
$
```

DOCKER SUMMARY

- Runs application in environment with known disk image
- Consistent between development, testing, and production environments
- Keeps app separate and "contained" so it doesn't affect host OS
- Makes management, upgrading, and removal trivial

RUNNING MULTI-CONTAINER APPLICATIONS

- Multiple services, e.g. a web app + database?
- We can group these together
- Can be done manually with Docker, but Docker Compose is more elegant

INSTALLING `docker-compose`

```
$ sudo curl -L \
  "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(
  -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   651  100   651    0     0    698      0 --:--:-- --:--:-- --:--:--   697
100 11.6M  100 11.6M    0     0  2231k      0  0:00:05  0:00:05 --:--:-- 3433k
$ sudo chmod +x /usr/local/bin/docker-compose
```

USING `docker-compose`

- Docker Compose looks for files called `docker-compose.yml` in current directory
- YAML file containing configurations for groups of related containers
- We'll make a simple web app that stores data in Redis

REALLY SIMPLE WEB APP

```
$ cd ~/lca2021-docker-talk/demos/simple_web_app
$ ls -l
total 16
-rw-r--r--    1 docker   staff          507 Jan 20 04:34 Dockerfile
-rw-r--r--    1 docker   staff          111 Jan 20 04:35 docker-compose.yml
-rw-r--r--    1 docker   staff           12 Jan 20 04:35 requirements.txt
-rw-r--r--    1 docker   staff          277 Jan 20 04:35 webapp.py
```

webapp.py:

```
import redis
from flask import Flask

hello = Flask(__name__)
cache = redis.Redis(host='redis')

def refresh_count():
    while True:
        return cache.incr('refresh_count')

@hello.route('/')
def hi():
    return 'Hello Flask! Refresh count: {}.\n'.format(refresh_count())
```

requirements.txt:

```
flask
redis
```

Dockerfile:

```
FROM python:3.7-alpine
WORKDIR /app
ENV FLASK_APP=webapp.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

docker-compose.yml:

```
version: "3.1"
services:
  web:
    build: .
    image: mattcen/simplewebapp
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

LET'S RUN IT!

```
$ docker-compose up -d
Creating network "simple_web_app_default" with the default driver
Building web
Step 1/10 : FROM python:3.7-alpine
 ---> 72e4ef8abf8e
[...]
Step 8/10 : EXPOSE 5000
 ---> Running in b86b56ac23cd
Removing intermediate container b86b56ac23cd
 ---> 71041c9a3620
Step 9/10 : COPY . .
 ---> d5a38cf1e575
Step 10/10 : CMD ["flask", "run"]
 ---> Running in b8fcb16313bd
Removing intermediate container b8fcb16313bd
 ---> 9f793cb0105a

Successfully built 9f793cb0105a
Successfully tagged mattcen/simplewebapp:latest
WARNING: Image for service web was built because it did not already exist. To rebuild this
Creating simple_web_app_web_1   ... done
Creating simple_web_app_redis_1 ... done
```

AND TEST IT

```
$ for redo in x y z; do curl http://localhost:5000; done
Hello Flask! Refresh count: 1.
Hello Flask! Refresh count: 2.
Hello Flask! Refresh count: 3.
```

Check it's actually modifying Redis:

```
$ docker-compose exec redis redis-cli get refresh_count
"3"
```

AND CLEAN UP

```
$ docker-compose down
Stopping simple_web_app_redis_1 ... done
Stopping simple_web_app_web_1   ... done
Removing simple_web_app_redis_1 ... done
Removing simple_web_app_web_1   ... done
Removing network simple_web_app_default
```

PUSHING A DOCKER IMAGE

- If you have a Docker Hub account, you can store images there
- … after authenticating to your Docker Engine with `docker login`
- These images can then be used by you or others
- I'll push this `mattcen/simplewebapp` image now for later use

```
$ docker push mattcen/simplewebapp
The push refers to repository [docker.io/mattcen/simplewebapp]
2c4609c155a4: Pushed
5cf997de0e7c: Pushed
43be206b777e: Pushed
0f309bc31411: Pushed
072957566e0f: Pushed
e2e42bb5b297: Mounted from library/python
e8f104f729a5: Mounted from library/python
5b2ca0c5db87: Mounted from library/python
b688d33030ff: Mounted from library/python
777b2c648970: Mounted from library/redis
latest: digest: sha256:abbd355bf17a52e2fc2df75374fa5642309a50f8e988b888b506ffc649c99e6d siz
```

DOCKER COMPOSE SUMMARY

- Abstracts away Docker's functionality
- Automates creation, config, and management, of groups of containers

SCALING WITH DOCKER SWARM

- Swarm mode is built into Docker Engine
- A Container Orchestrator (like Kubernetes, but easier)
- Can control Docker Engines on multiple machines
- Suitable for runnning redundant containers in production
- Swarm "Stacks" are groups of related containers
- "Stack" files are the same format as `docker-compose.yml`. Yay reuse!

CREATING A DOCKER SWARM

Swarm is build directly into the standard Docker Engine, so we can trivially create a single-node swarm like this:

node1 (manager)

```
docker@node1:~$ docker swarm init
Swarm initialized: current node (q0xlwpksgfxt7s3un5vqjjsnc) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5synm2ahj941bc1u44c8t1ms0oozze0u6qxotdcrlm9dkf8m6x-

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instru
```

(May need `docker swarm init --advertise-addr eth0`)

ADDING NODES TO A SWARM

node2 (worker)

```
docker@node2:~$ docker swarm join --token SWMTKN-1-5synm2ahj941bc1u44c8t1ms0oozze0u6qxotdc
This node joined a swarm as a worker.
```

node3 (worker)

```
docker@node3:~$ docker swarm join --token SWMTKN-1-5synm2ahj941bc1u44c8t1ms0oozze0u6qxotdc
This node joined a swarm as a worker.
```

LISTING SWARM NODES

node1 (manager)

```
docker@node1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY
q0xlwpksgfxt7s3un5vqjjsnc *   node1               Ready               Active
o5hau890eda7nzwqgrq1fc7if     node2               Ready               Active
p5gqdt2k09up8rlrr4nj2o3l0     node3               Ready               Active
docker@node1:~$
```

DEPLOYING A STACK

We can use our existing `docker-compose` file for our simple web app to deploy it as a stack:

```
docker@node1:~$ docker stack deploy -c docker-compose.yml webapp
Ignoring unsupported options: build

Creating network webapp_default
Creating service webapp_web
Creating service webapp_redis
```

- `build` unsupported in Swarm; needs pre-built image
- We pushed the image to Docker Hub earlier
- Because image is on Docker Hub, *only* local file we need is `docker-compose.yml`

CHECKING STACK STATUS

List configured stacks:

```
docker@node1:~$ docker stack ls
NAME                SERVICES            ORCHESTRATOR
webapp              2                   Swarm
```

List services within that stack:

```
docker@node1:~$ docker stack services webapp
ID                  NAME                MODE                REPLICAS            IMAGE
oquv4i6jc947        webapp_redis        replicated          1/1                 redis:alpir
w4iyas5q6nwo        webapp_web          replicated          1/1                 mattcen/sim
$
```

REMOVING OUR STACK

```
docker@node1:~$ docker stack rm webapp
Removing service webapp_redis
Removing service webapp_web
Removing network webapp_default
```

RECAP

- Docker makes it easy to run software in a contained, secure, consistent environment
- Docker Compose makes it easy to group several pieces of software together into an app
- Docker Swarm can run containers in production, across multiple physical machines if desired

MIGRATING TO CONTAINERS

- Production servers can run Docker Engine in single-node Swarm
- No worse than non-Docker config
- Can slowly migrate apps to containers
- Still get some wins, like consistent deployment across environments
- Makes it easy for developers to start working on your code

CONCLUSION

Give Docker a shot; it encourages good software design discipline, and gives lots of flexibility!