Docker

Michael Green

```
$ git clone https://github.com/1mikegrn/docker-2021.git
```

Pets: my dog Cooper

- My dog is unique
- My dog is irreplaceable
- Things wouldn’t be the same if something were to happen to him

In contrast: things you should not be saying about your server environments

In contrast: things you should not be saying about your server environments

- My environment is unique

In contrast: things you should not be saying about your server environments

- My environment is unique
- My environment is irreplaceable

In contrast: things you should not be saying about your server environments

- My environment is unique
- My environment is irreplaceable
- Things wouldn’t be the same if something were to happen to it

In contrast: things you should not be saying about your server environments

Pets are unique. There is only one, and will ever only be one.

This uniqueness paradigm has historically been a major pain point in application development

- Differences b/t dev and prod envs often lead to application issues
- Environment configurations previously needed to be curated on a per-server, per application basis
- Apps suffered from scalability/portability issues with being so tethered to the machines where they lived

Containers

Docker wraps applications in containers

A container is...

- An environment that encapsulates an application and its dependencies within a kernel namespace
  - The kernel is shared b/t the host and the containers
  - The namespace isolates the container’s filesystem, networking, etc., from the host

Containers

Docker wraps applications in containers

A container is...

- A lightweight abstraction that doesn’t require a full OS
  - The containers don’t waste resources trying to emulate a full server environment

Containers

Docker wraps applications in containers

A container is...

- A portable instance which can be run on the docker engine
  - Containers are plug-and-play on machines which run the docker engine
  - This allows containerized applications to work regardless of the host OS

Containers are built from Images

Images to containers can be thought of as synonyms to objects and instances

- A container is the instantiation of an image

```
class BaseImage:
    def __init__(self) -> None:
        self.fizz = "fizz"
class DerivedImage(BaseImage):
    def __init__(self) -> None:
        super().__init__()
        self.buzz = "buzz"
class MyImage(DerivedImage):
    def __init__(self) -> None:
        super().__init__()
        self.fizzbuzz = self.fizz + self.buzz
```

```
In [1]: container = MyImage()
In [2]: container.fizzbuzz
Out[2]: "fizzbuzz"
```

Containers are built from Images

Images to containers can be thought of as synonyms to objects and instances

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker image pull python
$ docker images python:latest
$ docker run -it python:latest /bin/bash
$ docker ps -al
$ docker start $(docker ps -aql)
$ docker exec -it $(docker ps -ql) python
$ docker rm $(docker ps -ql)
```

```
 -i  >>> interactive
 -t  >>> tty
 -a  >>> all
 -l  >>> latest
 -q  >>> quiet (only show ID)
--rm >>> remove
```

Containers should be considered Ephemeral

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run -it python /bin/bash
> cd && touch this

$ docker start -i $(docker ps -aql)
> cd && ls

$ docker run --rm -it python /bin/bash
> cd && ls
```

...though you can commit a container into an image

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run -it python /bin/bash
> cd && touch this

$ docker ps -al
$ docker commit $(docker ps -alq) my_container
$ docker images my_container
$ docker run -it $(docker images my_container -q) /bin/bash
> cd && ls
```

Images are built from Dockerfiles

Most of what we typically do with docker containers is codified in Dockerfiles.

- IaC lends itself to reproducibility

Start in a directory with a Dockerfile

```
Kubuntu [michaelgrn@x1-crunch] ~/Documents/Crunch/Presentations
$ touch Dockerfile.one
```

Images are built from Dockerfiles

Base Image

- Starting point for creating a container
  - python:latest was a base image
- **FROM** keyword

Current User

- Sets the user in the current image.
  - UID’s are the same b/t host and container
- **USER** keyword

Images are built from Dockerfiles

Environment Variables

- Sets environment variables inside image
- **ENV** keyword

Current working directory

- Sets the working directory for subsequent commands
- **WORKDIR** keyword

Images are built from Dockerfiles

Add files to image

- Copy files from source to destination
- **ADD**/**COPY** keywords (prefer **COPY** as it's more compact)

Specify Volumes

- Volumes link the container's filesystem to the host's filesystem
- **VOLUME** keyword

Images are built from Dockerfiles

Terminal Commands

- Runs instructions and commits resultant to new image
- **RUN** keyword

Executables

- Sets protocols for `docker run`
- **CMD** is the instruction set
- If **ENTRYPOINT** is defined, then **CMD** is passed to the **ENTRYPOINT** executable

Now build it!

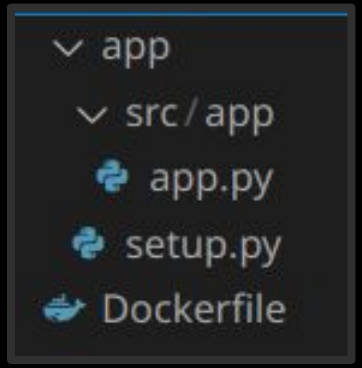
```
$ docker build -t my_image:latest .
```

```
 -f  >>> filename (default Dockerfile)
 -t  >>> tag (name:tag)

  .  >>> build context
```

Example

```
FROM python:latest
USER root
RUN useradd -u 1001 michaelgrn
WORKDIR /home/michaelgrn
RUN python -m venv venv \
      && chown -R michaelgrn /home/michaelgrn
USER michaelgrn
RUN . venv/bin/activate
COPY --chown=michaelgrn app/ /code/app
WORKDIR /code
RUN python -m pip install -e /code/app
CMD python -m app
```



Docker-Compose

Docker compose is a wrapper tool that provides management across a set of containers.

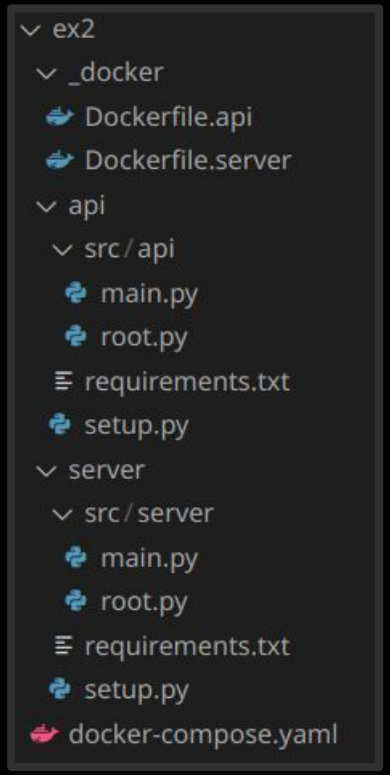
It allows developers to express the type of commands we've previously seen as yaml.

Orchestrates the build process across a set of containers.

Docker-Compose

```yaml
version: '3.8'

services:
 server:
   image: ex2/server:latest
   build:
     context: .
     dockerfile: _docker/Dockerfile.server
   ports:
     - "8081:8080"
 api:
   image: ex2/api:latest
   build:
     context: .
     dockerfile: _docker/Dockerfile.api
   ports:
     - "8080:8080"
```



Docker-Compose

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker-compose build
$ docker images | head -10
$ docker-compose up -d
$ docker-compose ps
$ docker-compose down
```

Troubleshooting build issues

Docker images are layered, and as such if there is an issue that occurs during the build process, it’s possible to enter a container of the previous layer to attempt to diagnose.

Appendix I: Docker Volumes

Docker volumes are directories which aren't associated with the containers UFS

Bind mounted to the host

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run --rm -it -v /home/michaelgrn/Desktop:/code python /bin/bash
```

Appendix II: Logging

By default Docker logs everything sent to STDOUT and STDERR

Logs can be retrieved with the logs command

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run --name exlog python sh -c 'echo "LOG: OUT"'-f
$ docker logs -t exlog
$ docker rm exlog
```

Appendix II: Logging

Logs can also be streamed from containers using the `-f` flag

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run --name exstream python
> sh -c 'while true; do echo "ping"; sleep 1; done;'

$ docker logs -f exstream
$ docker rm exstream
```

Appendix III: Networking

Docker containers are exposed to the outside world through the process of publishing ports

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run --rm -d -p 9000:80 nginx

> http://localhost:9000
```

Appendix III: Networking

Docker containers can talk to each other over an internal docker network using

`--link <CONTAINER>:<ALIAS>`

Using docker links will automatically add the alias and link container ID to /etc/hosts

```
Kubuntu [michaelgrn@x1-crunch] ~/crunch/zoom
$ docker run --rm -d --name exmongo mongo
$ docker run --rm --link exmongo:mongo -it python /bin/bash
> env
```

Appendix III: Networking

More complex networking setups make the use of “ambassador” containers which serve as proxy containers which forward traffic to actual services

Their advantage is that it allows production network architecture to differ from a dev environment w/o necessitating any changes to application code