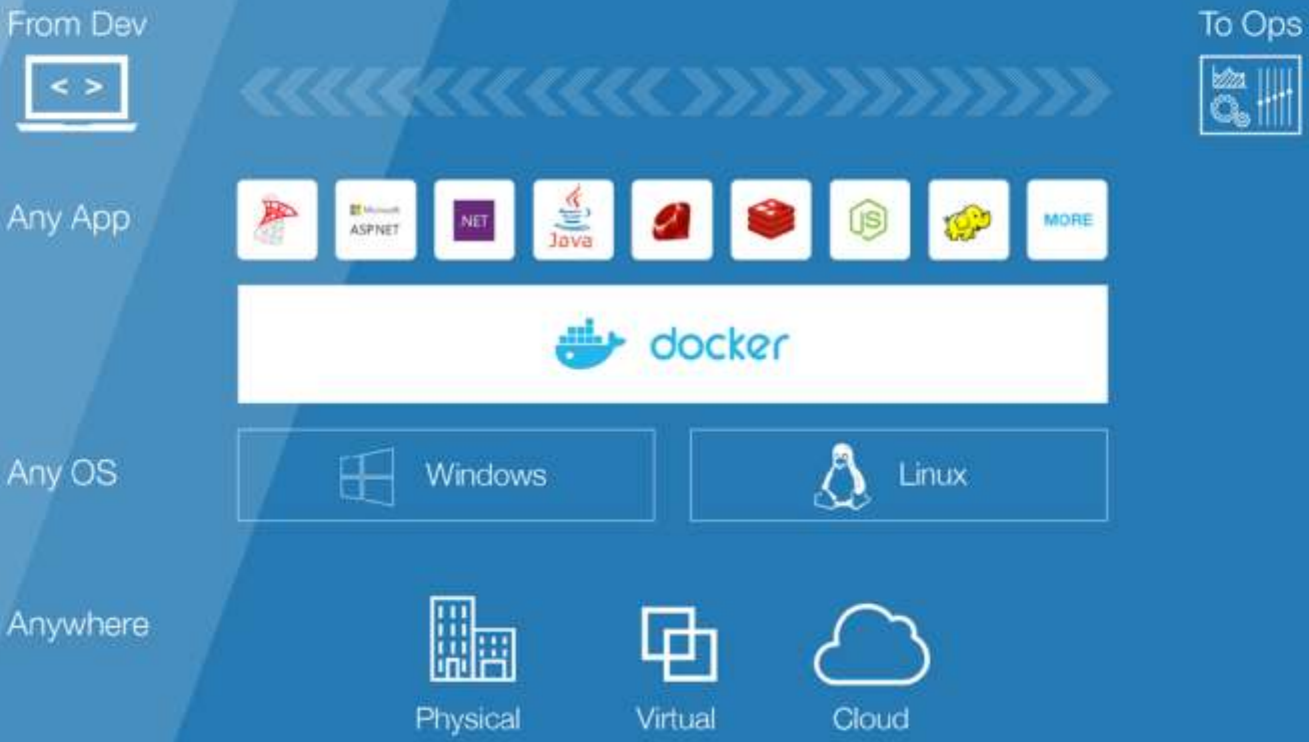
Build, Ship, and Run

Build, Ship, Run, Any App Anywhere



Some Docker vocabulary

Docker Image

The basis of a Docker container. Represents a full application

Docker Container

The standard unit in which the application service resides and executes

Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

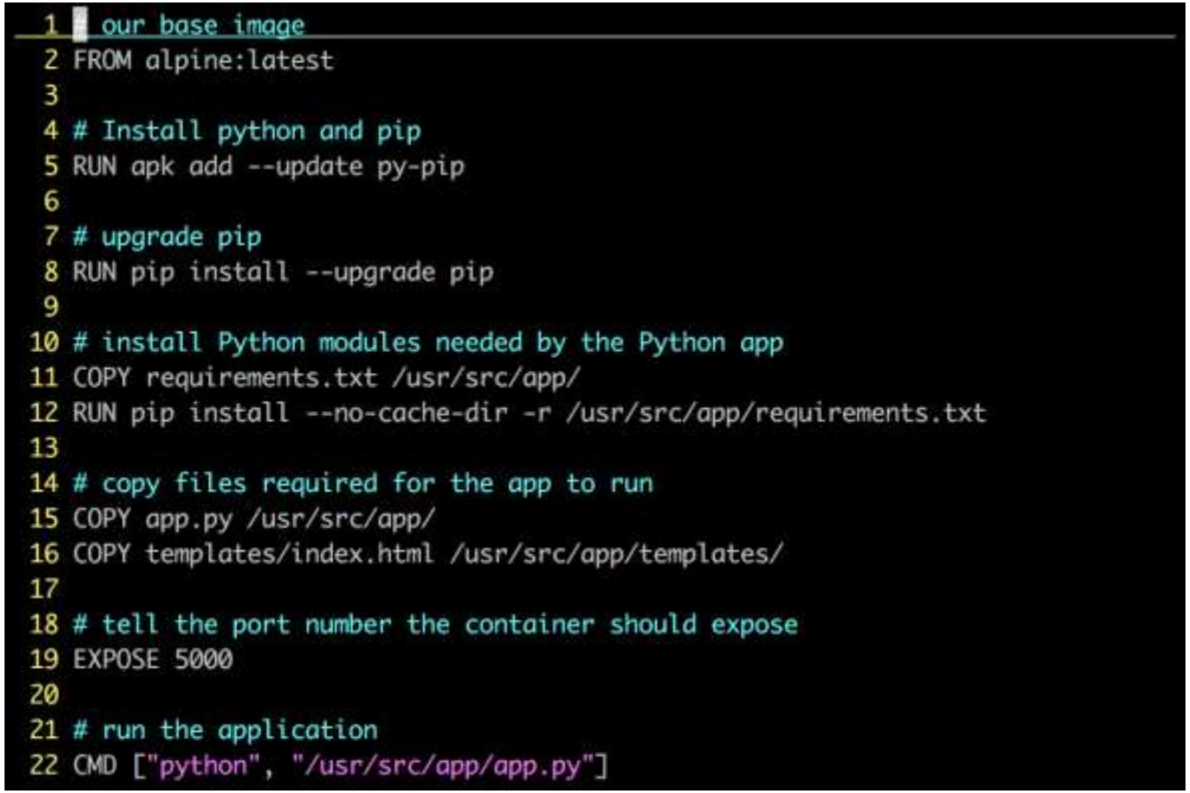
Registry Service (Docker Hub or Docker Trusted Registry)

Cloud or server based storage and distribution service for your images

Basic Docker Commands

```
$ docker pull mikegcoleman/catweb:latest

$ docker images

$ docker run -d -p 5000:5000 --name catweb mikegcoleman/catweb:latest

$ docker ps

$ docker stop catweb (or <container id>)

$ docker rm catweb (or <container id>)

$ docker rmi mikegcoleman/catweb:latest (or <image id>)
```

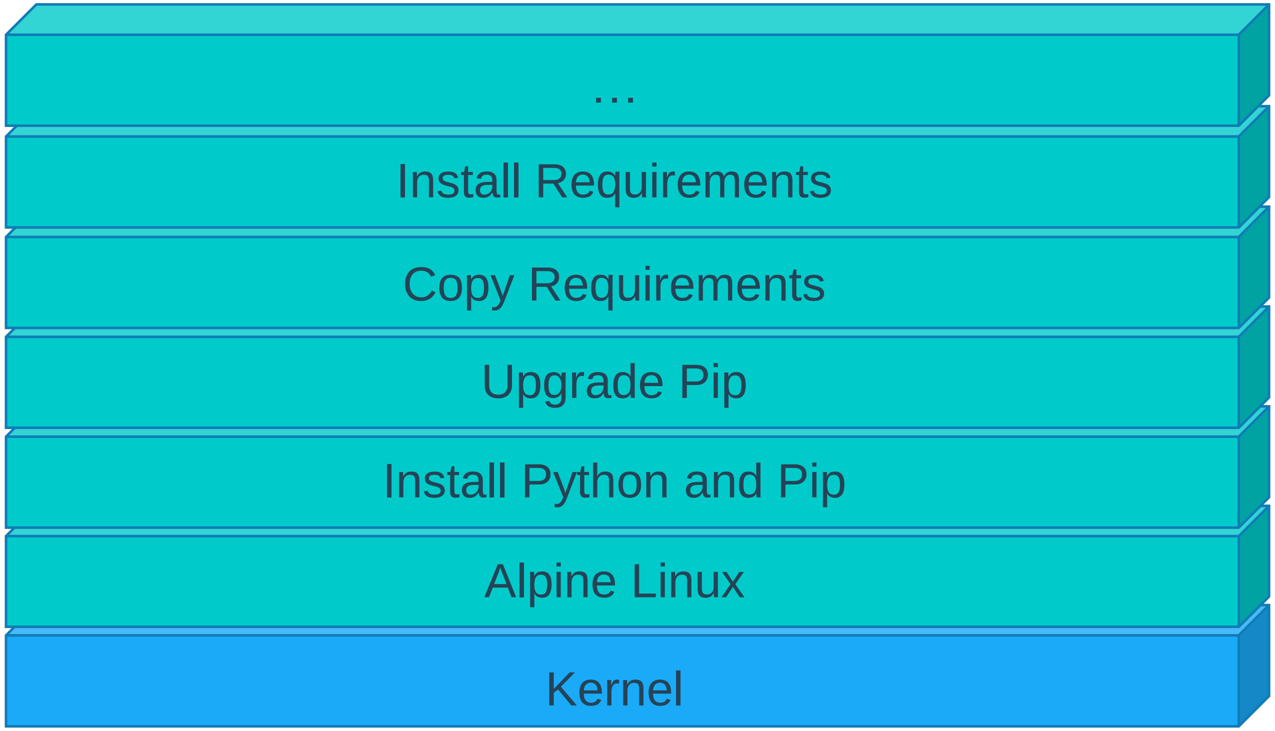
Dockerfile – Linux Example



```
# our base image
FROM alpine:latest

# Install python and pip
RUN apk add --update py-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

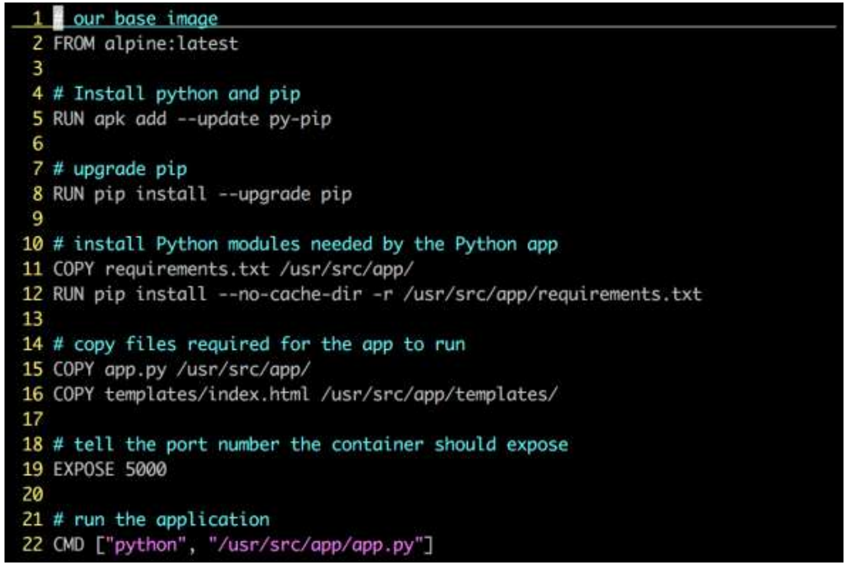
- Instructions on how to build a Docker image
- Looks very similar to “native” commands
- Important to optimize your Dockerfile

Image Layers



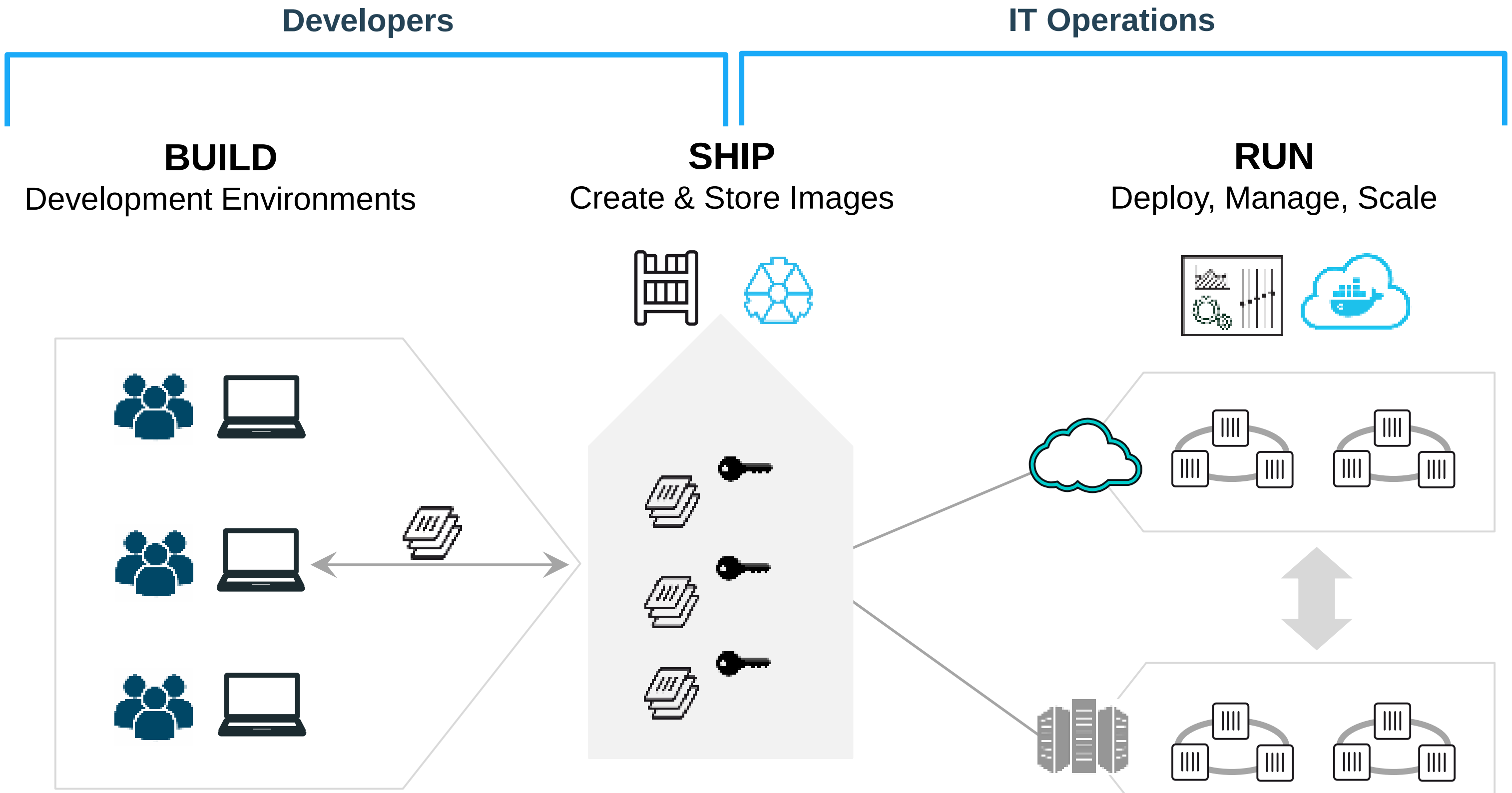
Basic Docker Commands

```
$ docker build -t mikegcoleman/catweb:2.0 .
$ docker push mikegcoleman/catweb:2.0
```



```
# our base image
FROM alpine:latest

# Install python and pip
RUN apk add --update py-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

Put it all together: Build, Ship, Run Workflow



What about data persistence?

- Volumes allow you to specify a directory in the container that exists outside of the docker file system structure

- Can be used to share (and persist) data between containers

- Directory persists after the container is deleted
  - Unless you explicitly delete it

- Can be created in a Dockerfile or via CLI

WHAT IS DOCKER

- Allows you ship code along with all its dependencies in a self-contained manner
- Dockerfile like a manifest allows you to describe these dependencies and steps to set it up
- Spin up many instances of this image as you want (container)
- Cloud ready

WHY USE IT

- So many many libraries, so many many versions
- Dependency Install nightmare, be shielded from inadvertent upgrades
- Simplify and speed up focus on actual ML problem not supporting infrastructure

STEP 1

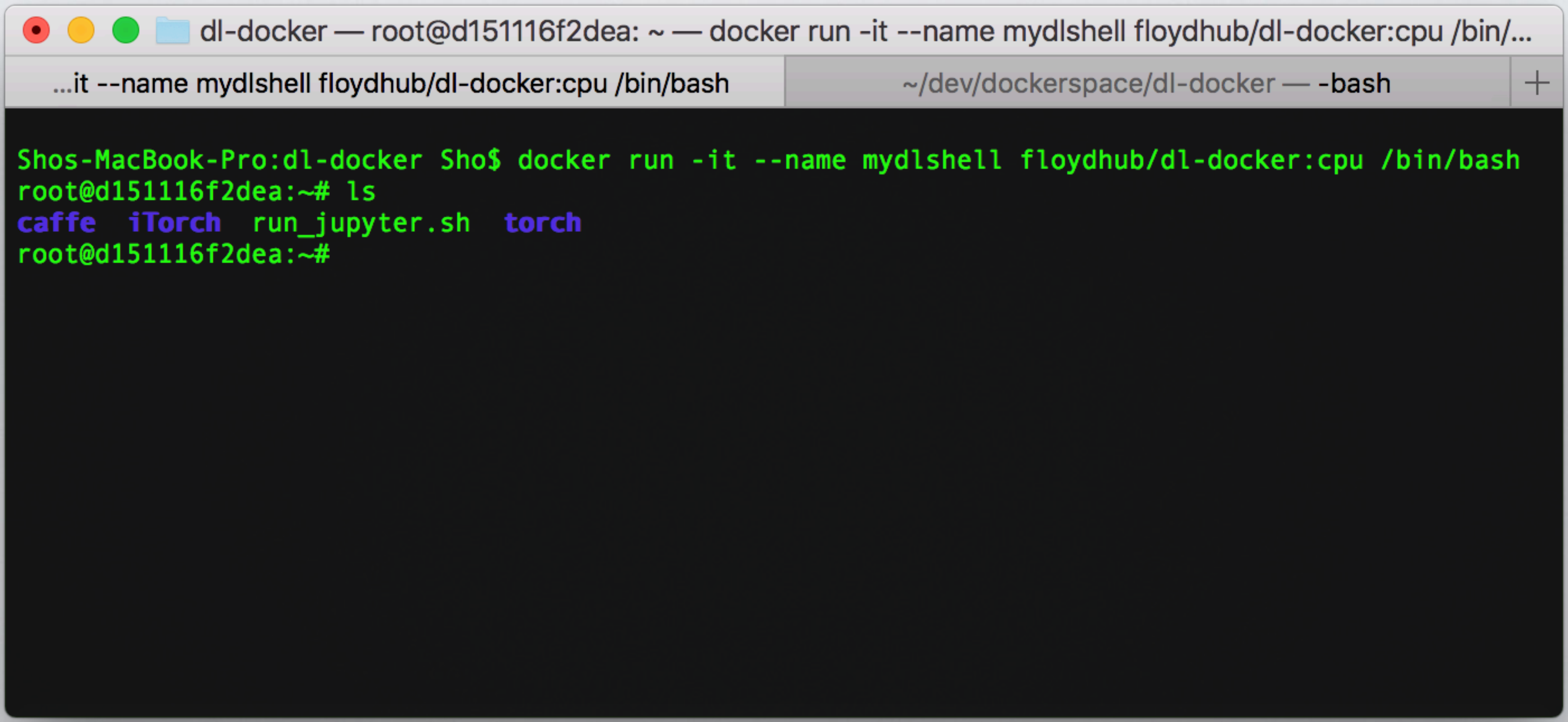
Download the image of choice from Docker Hub

```
$ docker pull floydhub/dl-docker:cpu
```

STEP 2

Start container with that image

$: docker run -it --name mydlshell floydhub/dl-docker:cpu /bin/bash



dl-docker — root@d151116f2dea: ~ — docker run -it --name mydlshell floydhub/dl-docker:cpu /bin/...
...it --name mydlshell floydhub/dl-docker:cpu /bin/bash
~/dev/dockerspace/dl-docker — -bash

```
Shos-MacBook-Pro:dl-docker Sho$ docker run -it --name mydlshell floydhub/dl-docker:cpu /bin/bash
root@d151116f2dea:~# ls
caffe  iTorch  run_jupyter.sh  torch
root@d151116f2dea:~#
```

STEP 2B

Another Way to Start Container … Using Assigned Label

```
$: docker start -ia mydlshell
```

STEP 3

Interact with the container to perform various tasks

Approach 1: Copy files into Container

```
$: docker cp ~/dev/dockerspace/census_keras.py dl-docker/ mydlshell:/root/test/census_keras.py
```

STEP 3B

Or Share a Volume (my preferred method)

```
$: docker run -it -v ~/dev/dockerspace/dl-docker:/projects/dl-docker --name
mydlspace floydhub/dl-docker:cpu
```

```
$: docker start mydlspace
```

```
$: docker exec -it mydlspace python /projects/dl-docker/census_keras.py
```

"HOW CAN IT BE THIS EASY ?"

Docker Compose

Defining and running multi-container Docker applications

What is Docker Compose?

1

A tool for defining and running multi-container Docker applications

2

With Compose, you use a YAML file to configure your application's services.

3

Compose works in all environments: production, staging, development, testing, as well as CI workflows.

4

With a single command, you create and start all the services from your configuration

BUT

- Binding to different ports on the host
- Setting environment variables differently
- Specifying a restart policy
- Adding extra services

Docker Compose is a 3 Steps Process

Define your app's environment with a Dockerfile

Define the services that make up your app in Docker Compose file

Run the CLI:

$ docker-compose up

dockerfile

```
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py
```

docker-compose.yml

```yaml
web:
  build: .
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis
redis:
  image: redis
```

docker-compose up

```
$ docker-compose up
Pulling image redis...
Building web...
Starting composetest_redis_1...
Starting composetest_web_1...
redis_1 | [8] 02 Jan 18:43:35.576 # Server started, Redis version 2.8.3
web_1   |  * Running on http://0.0.0.0:5000/
```

docker compose cli

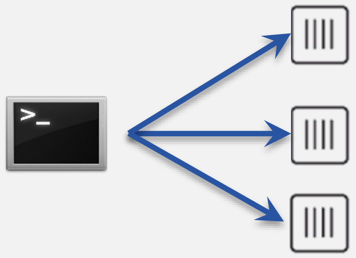
commands

```
build

logs

run

scale

up
```

Docker Compose: Multi Container Applications

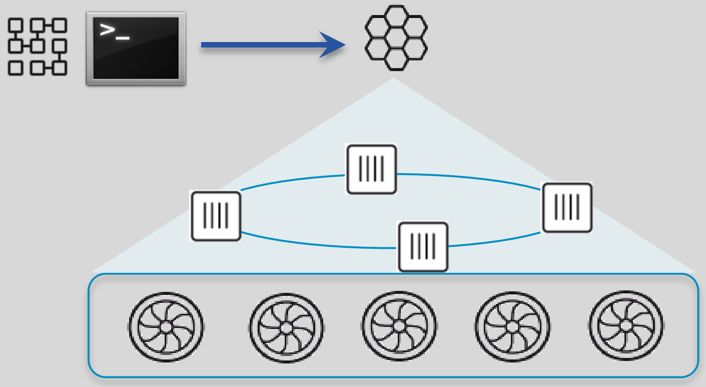
Without Compose

- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



With Compose

- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane



Multiple container application in Docker

```
$ docker pull mysql

$ docker pull wordpress

$ docker run -d --name=db -e MYSQL_ROOT_PASSWORD=root mysql

$ docker run --name=wp -p 8000:80 --link db:db \
     -e WORDPRESS_DB_HOST=db \
     -e WORDPRESS_DB_PASSWORD=root wordpress
```

Docker Compose - YAML

```
$ docker pull mysql

$ docker pull wordpress

$ docker run -d --name=db
        -e MYSQL_ROOT_PASSWORD=root mysql

$ docker run --name=wp  \
        -p 8000:80 \
        --link db:db \
        -e WORDPRESS_DB_HOST=db \
        -e WORDPRESS_DB_PASSWORD=root  \
        wordpress
```

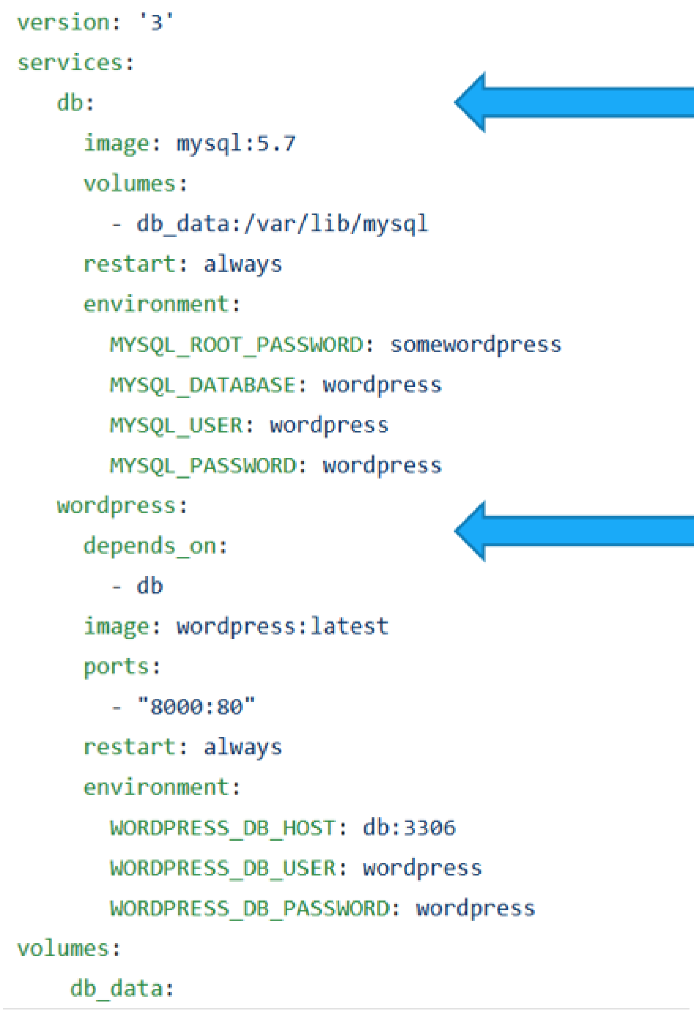
```
version: '2'
services:
  db:
   image: mysql
   environment:
    MYSQL_ROOT_PASSWORD: root
  wp:
   depends_on:
    - db
   image: wordpress
   ports:
    - "8000:80"
   environment:
    WORDPRESS_DB_HOST: db
    WORDPRESS_DB_PASSWORD: root
```

docker

Docker Compose - YAML

```
$ docker-compose up

$ docker-compose ps

$ docker-compose stop
```

```
version: '2'
services:
  db:
   image: mysql
   environment:
    MYSQL_ROOT_PASSWORD: root
  wp:
   depends_on:
    - db
   image: wordpress
   ports:
    - "8000:80"
   environment:
    WORDPRESS_DB_HOST: db
    WORDPRESS_DB_PASSWORD: root
```



```yaml
version: '3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Backend Service

Specify Volumes/Network

Environmental variables

Frontend Service

Specify Volumes/Network

Environmental variables