An Introduction To Docker

Ali Nobari
Software Developer (Coop)

What we will cover

Docker Overview

Creating a new application is as easy as...

1. Importing the project and its dependencies
2. Creating the Dockerfile
3. Running the application through Docker

Docker Overview

Using an existing Docker application is as easy as...

1. Importing the Docker application from a repo
2. Running the application through Docker

Docker Overview

Improvements from a Virtual Machine:

- Extremely **<u>less</u>** resource intensive than a VM
- **<u>Higher performance</u>** for applications running in Docker
  - Docker estimates as high as **<u>twice</u>** the speed of a VM
- VM could take **<u>several minutes</u>** to create/launch locally, whereas Docker only needs **<u>a few seconds</u>**
- **<u>Compatible</u>** over various systems

Let’s first understand the basics

The **image** is the application that you want to execute. It includes everything: the code, a runtime, libraries, and config files

A **container** is an executed instance of the image: what the image becomes in memory when executed, much like a process

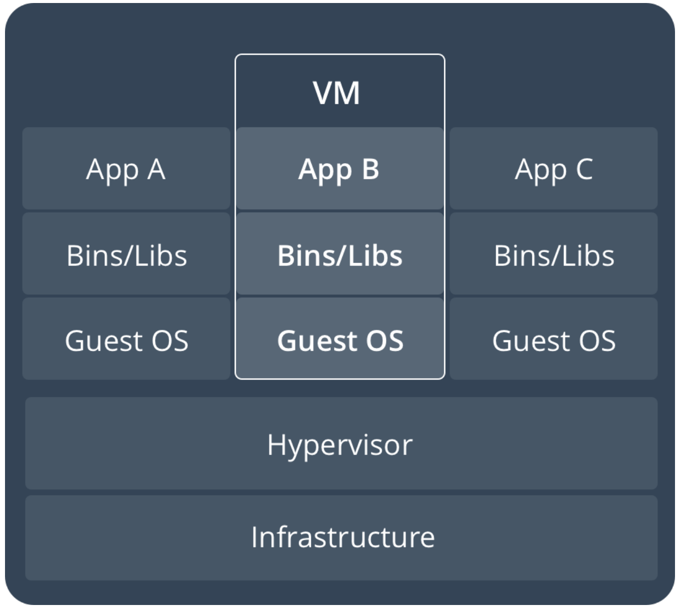
Let’s look at it this way

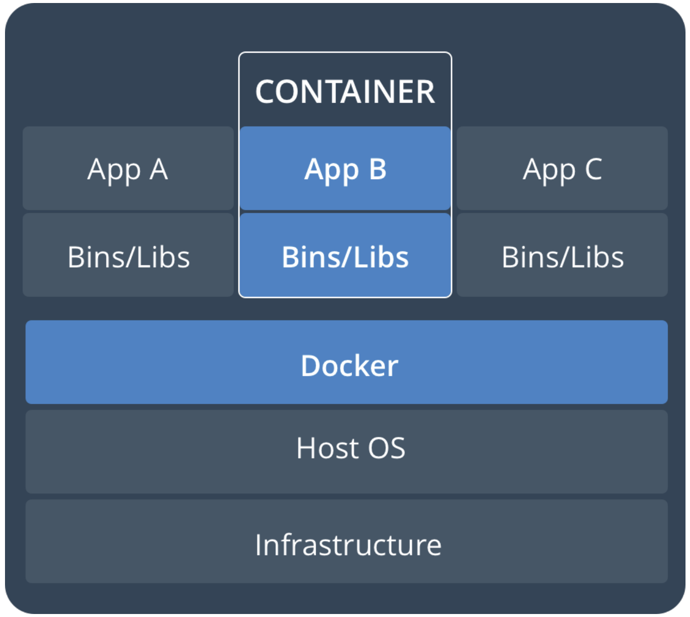
The **image**: A CD

The **container**: A CD Player

A container is a lightweight VM?

Layers! A game changing concept!





- Runs a memory-intensive **guest** OS
- Gives virtual access to **host**
- Has more resources than needed

- A **container** runs natively
- Sharing the Kernel
- Only taking memory it needs

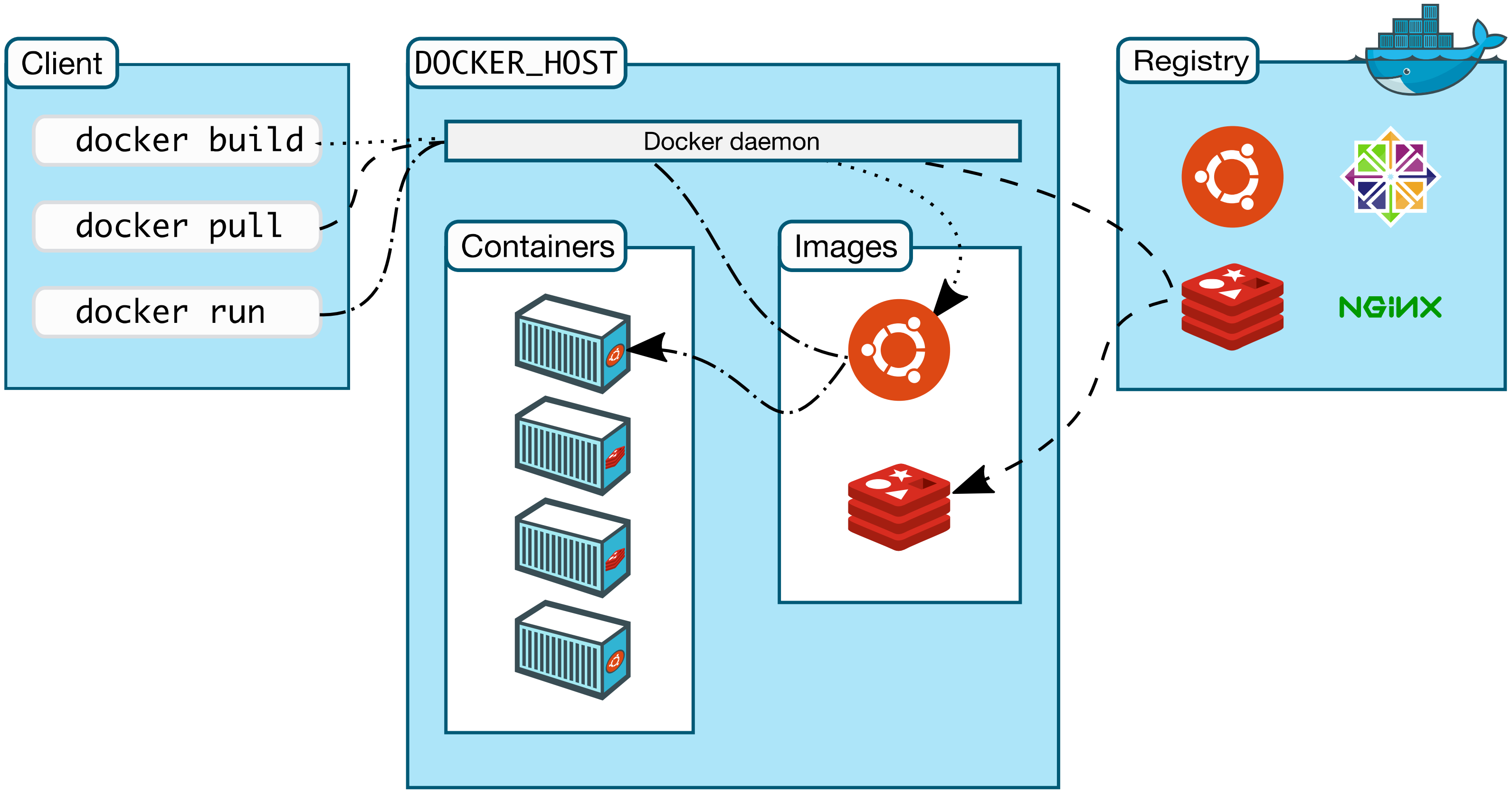
What can Docker do?

- Fast and consistent delivery of applications
  - Write code/create projects locally
  - Share your work with others using Docker
  - Use Docker to push to a test environment
  - Seamless integration between test and development environments

What can Docker do?

- Portable and easy to scale
  - Can run on a mixture of environments, such as a personal machine, physical/virtual machines, or cloud solutions

Docker Architecture Overview



Docker Architecture Overview

- The **client** talks to the **daemon**
  - They communicate either through REST API, UNIX sockets, or a network interface
- The **client** and the **daemon** can run on the same system
  - The **daemon** can also be on a remote system
- The **daemon** uses the **Registry** for the storing of Images

The Docker Client

- The main way Docker users will interact with Docker
  - Using the **docker** command over the terminal
- An example is using the command **docker run** sending it onto **dockerd**(the daemon), which carries it out
- The **docker** command uses the Docker API for communication with the Daemon
- The Docker client is able to communicate with more one daemon

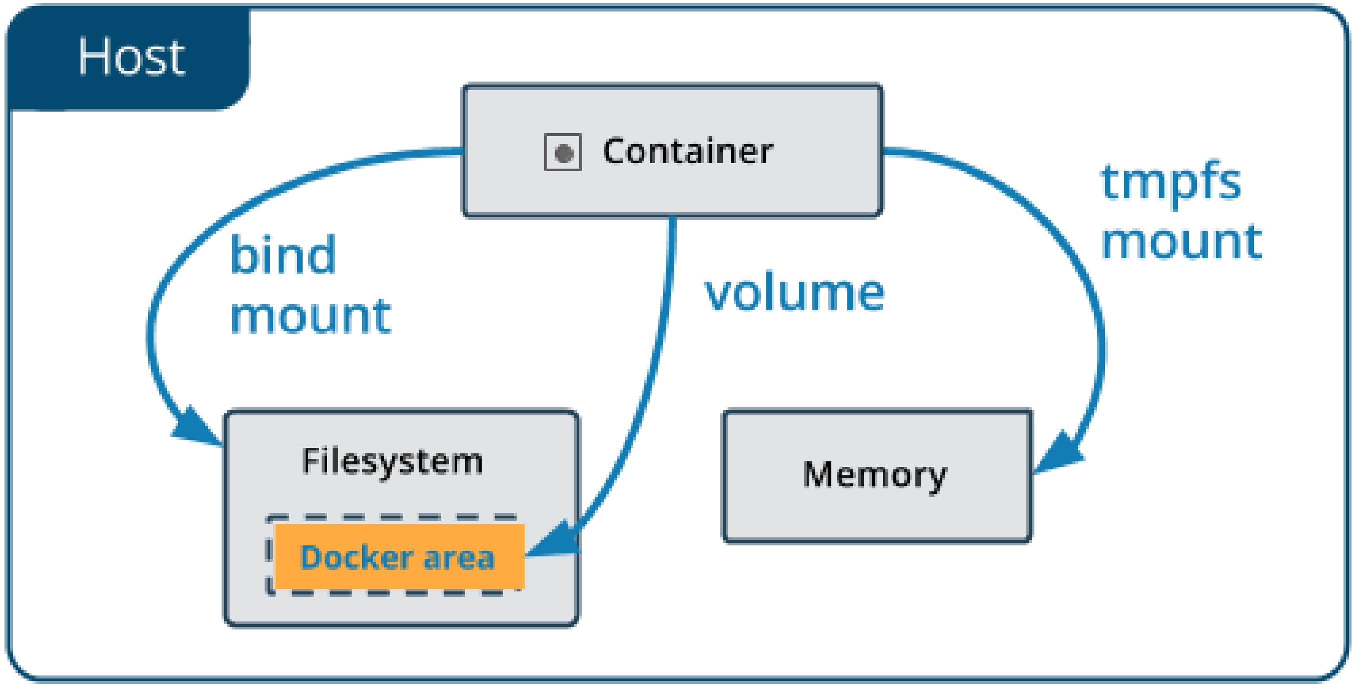
The Docker Daemon

- The Docker daemon (**dockerd**) listens for Docker API requests
- The daemon manages Docker objects including:
  - Images, containers, networks, and volumes
- The daemon does the heavy work of building, running and distributing containers
- The daemon can also communicate with other daemons to manage Docker services

Docker Registries

- The storage for Docker images
- This includes the Docker Hub and Docker Cloud
  - Cloud services for Docker
  - Public resgistries that anyone can access
- Private Repositories can also be set up
- Docker Store is another registry
  - Allows to buy and sell Docker images
  - Or distribute them for free

Volumes


Host
Container
bind mount
volume
tmpfs mount
Filesystem
Docker area
Memory

Volumes

- The preferred mechanism for persisting data
  - Either generated by or used by a Docker container
- Volume advantages over bind mounts
  - **Easier** to back up and migrate
  - Manage volumes using Docker CLI (terminal)
  - Work on **multi OS** containers
  - Volume drivers allow for **encryption** of the content, or other added functionality
  - Safely shared between multiple containers
  - Stored on remote hosts or cloud providers

Install Docker CE (Ubuntu)

```
# Update the system
$ sudo apt update

# Intall packages for using the Docker repository
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common

#Add Docker GPG key
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

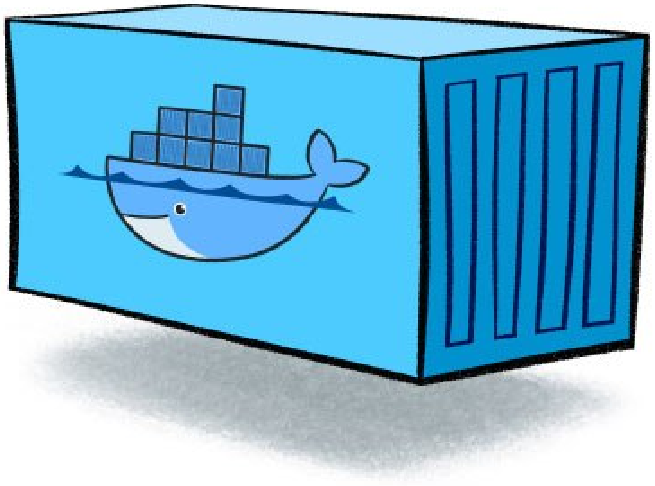
Install Docker CE (Ubuntu)

```
# Verify the installation
$ sudo apt-key fingerprint 0EBFCD88
 - $ pub   4096R/0EBFCD88 2017-02-22
 - $       Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
 - $ uid                  Docker Release (CE deb) <docker@docker.com>
 - $ sub   4096R/F273FCD8 2017-02-22

# Set up a  stable repository (this is specific for X86_64/amd64)
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"n
```

Install Docker CE (Ubuntu)

```
# Install Docker CE onto the machine
$ sudo apt update
$ sudo apt install docker-ce


# Test to see if Docker and the repositories have installed correctly
$ sudo docker run hello-world
```

Containers



Containers

- Defined through a **Dockerfile**
  - Series of instructions for building an image
  - Defines the access to resources outside the container
  - Specifies exactly what needs to be copied in to build the image
  - **Dockerfile** will behave exactly the same wherever it runs

The Dockerfile

```
$ mkdir docker-example
$ cd docker-example
$ nano Dockerfile
```

```
FROM ubuntu:latest
RUN apt update -y && apt install -y
   python-pip python-dev build-essential
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

The Dockerfile

We first have to pick our base

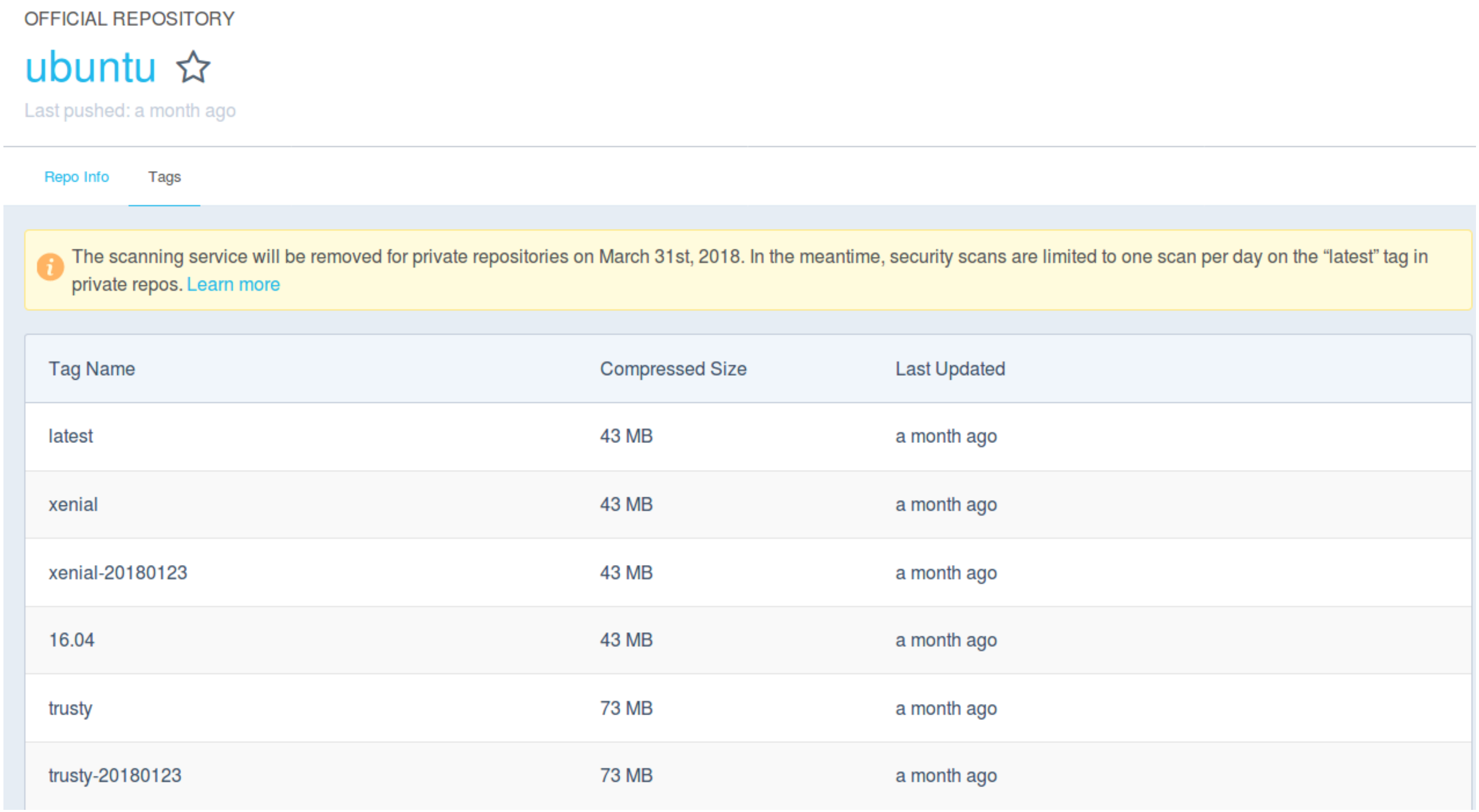
From the Ubuntu image:

Ubuntu tag	Compressed Size
/latest	43 MB

From the Python image:

Python tag	Compressed Size
/2.7-alpine3.6	30 MB

DockerHub



OFFICIAL REPOSITORY

ubuntu ☆

Last pushed: a month ago

Repo Info Tags

The scanning service will be removed for private repositories on March 31st, 2018. In the meantime, security scans are limited to one scan per day on the "latest" tag in private repos. Learn more

Tag Name	Compressed Size	Last Updated
latest	43 MB	a month ago
xenial	43 MB	a month ago
xenial-20180123	43 MB	a month ago
16.04	43 MB	a month ago
trusty	73 MB	a month ago
trusty-20180123	73 MB	a month ago

An alternative from the Ubuntu image

```
FROM ubuntu:latest

RUN apt update -y && apt install -y
   python-pip python-dev build-
   essential

WORKDIR /app

ADD . /app

RUN pip install -r requirements.txt

Expose 80

ENTRYPOINT [“python”]

CMD [“application.py”]
```

An alternative from the Ubuntu image

```
FROM python:2.7-alpine3.6
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

Using a custom base container

```
FROM python:2501231234/docker_app1
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

Setting a working directory

```
FROM ubuntu:latest
RUN apt update -y && apt install -y
   python-pip python-dev build-essential
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

Copy the current directory to the container

```
FROM ubuntu:latest
RUN apt update -y && apt install -y
   python-pip python-dev build-essential
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

Defining the RUN statements

```
FROM ubuntu:latest
RUN apt update -y && apt install -y
    python-pip python-dev build-
    essential
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

A better approach to RUN and ADD

Bad space usage:

```
ADD http://site.come/largefile.tar.xz /app/site
RUN tar -xvf /app/site/largefile.tar.xz -C /app/site
RUN make -C /app/site/ all
```

Wise space usage:

```
RUN mkdir -p /app/site/ \
   && curl -SL http://site.come/largefile.tar.xz  /app/site \
   | tar -xvf /app/site/ &&
   /app/site/largefile.tar.xz -C /app/site \
   && make -C /app/site/ all
```

Requirements File

- A Requirements file defines additional dependencies needed by Docker
- It installs the needed dependencies for the application
- Used to simplify and organize the Dockerfile

Entrypoint/CMD

```
FROM ubuntu:latest
RUN apt update -y && apt install -y
    python-pip python-dev build-essential
WORKDIR /app
ADD . /app
RUN pip install -r requirements.txt
Expose 80
ENTRYPOINT [“python”]
CMD [“application.py”]
```

Services

- Simply put, they are containers in production
- Container specifications:
  - The port to use
  - The parameters to define
  - The resources to pass in
  - The volumes to mount
- This can be accomplished easily with a YAML file

docker-compose.yml

```yaml
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

docker-compose.yml

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

docker-compose.yml

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: “0.1”
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

docker-compose.yml

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

Run Docker Service

```
# Initialize the swarm
$ docker swarm init

# Deploy the service using a name
$ docker stack deploy -c docker-compose.yml swarm_name
```

Our single application is now running 5 container instances of our deployed image on one host

Docker Swarm

- Group of machines running Docker and joined into a Cluster
- Normal Docker commands are used, but they apply to a Cluster
- The machines can be either **physical** or **virtual**
  - This can include running multiple Vms (such as Virtualbox)

Demo Time

EXAMPLES

- Python
- Java
- ASP.NET

Python Example

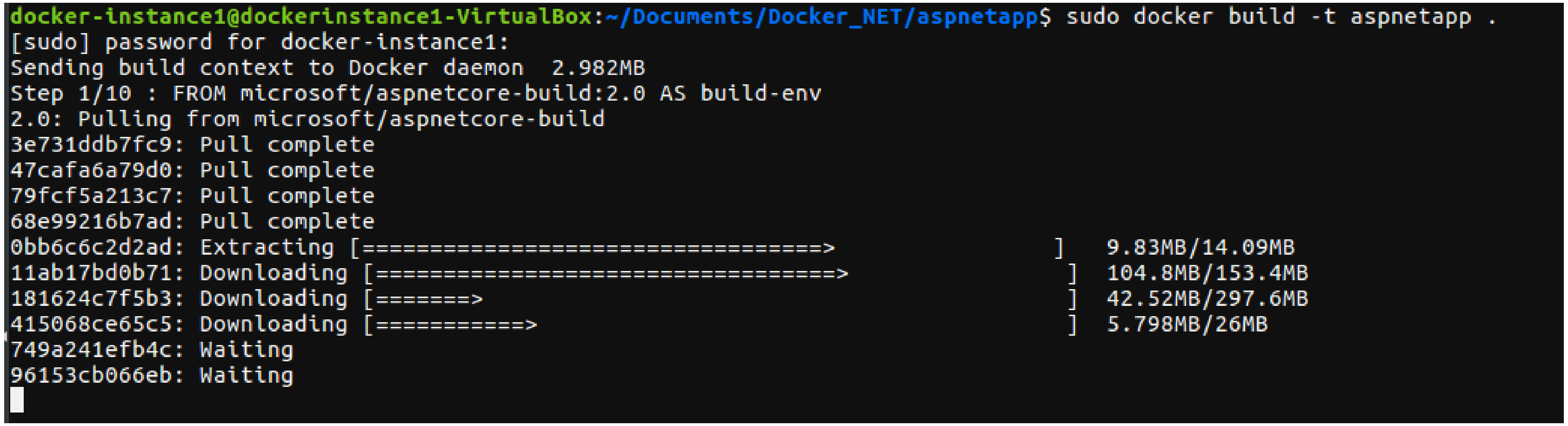
- Create directory for the Application
- Create a Python application
- Specify a Dockerfile for the application
- Import requirements if needed
- Run commands to build and run the application

Python Example

```
# Create the necessary folders and files
$ mkdir docker-python-example
$ cd docker-python-example
$ nano Dockerfile
$ nano requirements.txt
$ nano app.py
```

Python Example

```
# Build the Docker application

$ docker build -t pythonhello ./
```



```
docker-instance1@dockerinstance1-VirtualBox:~/Documents/Docker_NET/aspnetapp$ sudo docker build -t aspnetapp .
[sudo] password for docker-instance1:
Sending build context to Docker daemon  2.982MB
Step 1/10 : FROM microsoft/aspnetcore-build:2.0 AS build-env
2.0: Pulling from microsoft/aspnetcore-build
3e731ddb7fc9: Pull complete
47cafa6a79d0: Pull complete
79fcf5a213c7: Pull complete
68e99216b7ad: Pull complete
0bb6c6c2d2ad: Extracting [===================================>               ]   9.83MB/14.09MB
11ab17bd0b71: Downloading [==================================>               ]  104.8MB/153.4MB
181624c7f5b3: Downloading [=======>                                           ]  42.52MB/297.6MB
415068ce65c5: Downloading [===========>                                       ]  5.798MB/26MB
749a241efb4c: Waiting
96153cb066eb: Waiting
```

Python Example

```
# Run the application detached in the background
$ docker run -d -p 4000:80 pythonhello
$ curl http://localhost:4000

# View the container running in the background
$ docker container ls

# Stop the container
$ docker container stop $ID
```

Java Example (Dockerize a Java app)

- Create directory for the Application
- Import the Java application
- Specify a Dockerfile for the application
- Import requirements if needed
- Run commands to build and run the application

Java Example (Dockerize a Java app)

```
# Build the image of the Dockerfile
$ docker build -f Dockerfile -t java-app:test .

# View the Java app and its Tag
$ sudo docker image ls

# Build the Java application
$ docker run --rm -v $PWD:/app -w /app java-app:test
javac Main.java
```

Java Example (Dockerize a Java app)

# Build the Java application

```
$ docker run --rm -v $PWD:/app -w /app java-app:test
javac HelloWorld.java
```

# Run the Java application

```
$ docker run --rm -v $PWD:/app -w /app java-app:test
java HelloWorld
```

Flag explanation

--rm : automatically clean up the container and remove the file system when the container exits

-v : specifies a volume

-w: reference to the working directory

ASP.NET Application

- Create directory for the Application
- Create the ASP.NET application
- Specify a Dockerfile for the application
- Import requirements if needed
- Run commands to build and run the application

ASP.NET Application

```
# Build the image of the Dockerfile
$ docker build -t aspnetapp:latest .

# Run and view the ASP.NET application
$ docker run -d -it --rm -p 8000:80 aspnetapp
$ curl http://localhost:8000

# View the container running in the background
$ docker container ls

# Stop the container
$ docker container stop $ID
```

What is the takeaway?

- Docker can be a great improvement from a VM
  - Less space
  - Higher performance
- Docker allows application instances to build and run within seconds
- Run numerous instances **simultaneously** and in **parallel**
- Only need to specify the Dockerfile **once**
- Expect the same result even on different OS and hardware

Questions?