

# Docker

# Agenda

## Section 1:

- What is Docker
- Basic Docker Commands
- Dockerfiles

## Section 2:

- Anatomy of a Docker image
- Docker volumes

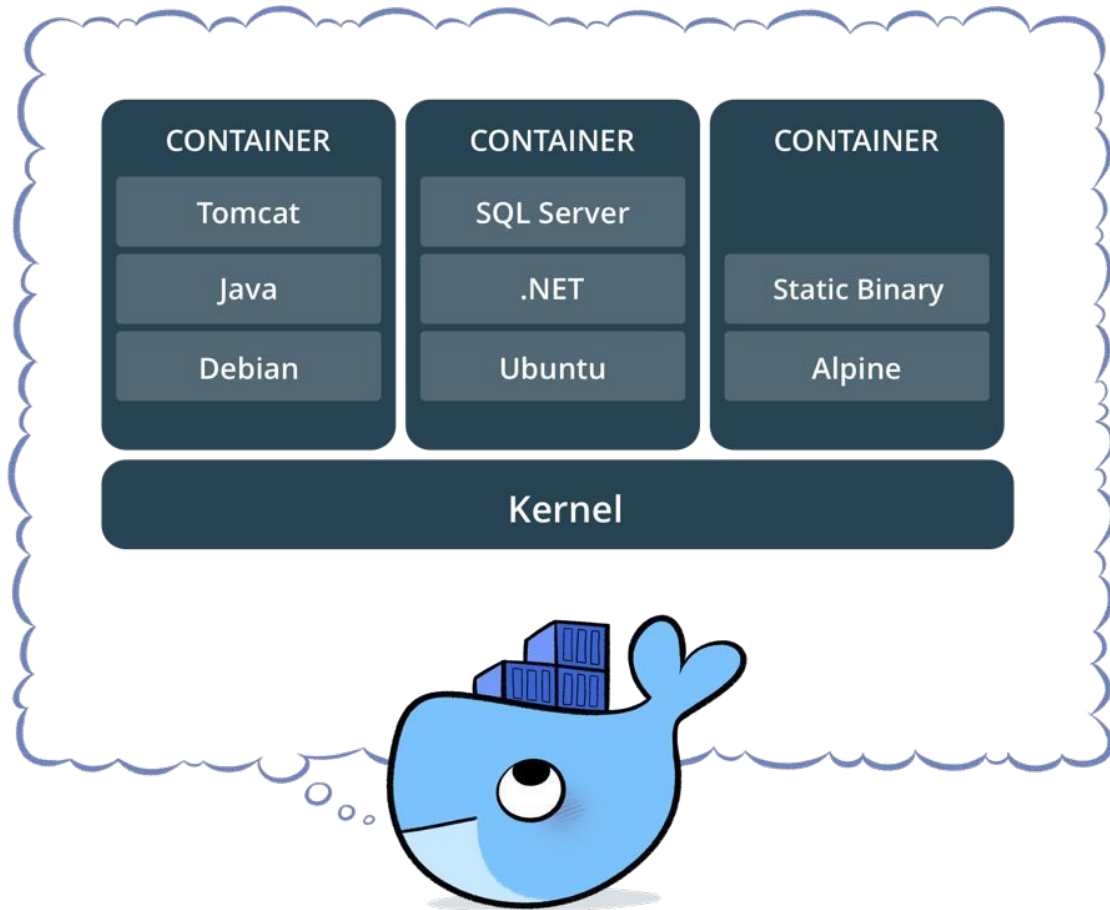
## Section 3:

- Networking

## Section 4:

- Docker compose / stacks

# What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Containers native to Windows Server 2016

# The Role of Images and Containers



Docker Image

Example: Ubuntu with Node.js and  
Application Code



Docker Container

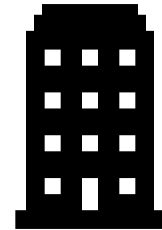
Created by using an image. Runs your  
application.

# Docker containers are NOT VMs

- Easy connection to make
- Fundamentally different architectures
- Fundamentally different benefits

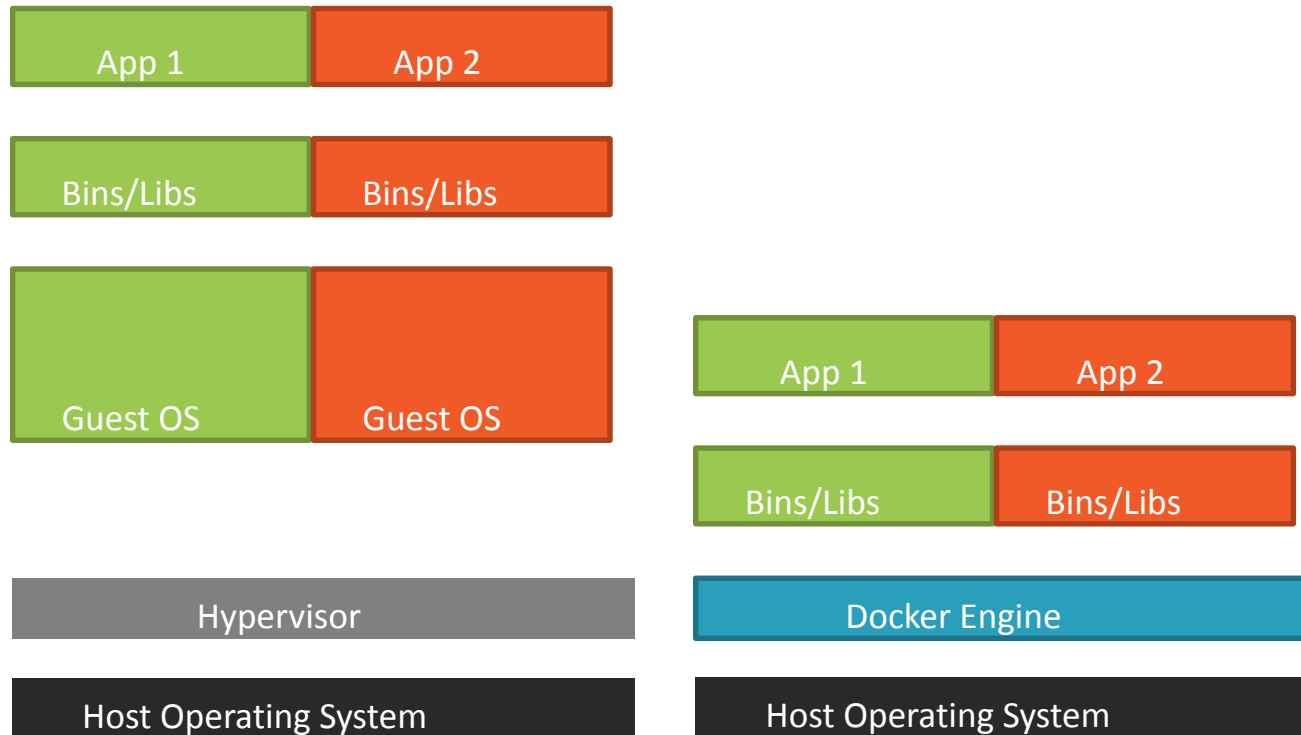


Maquina Virtual



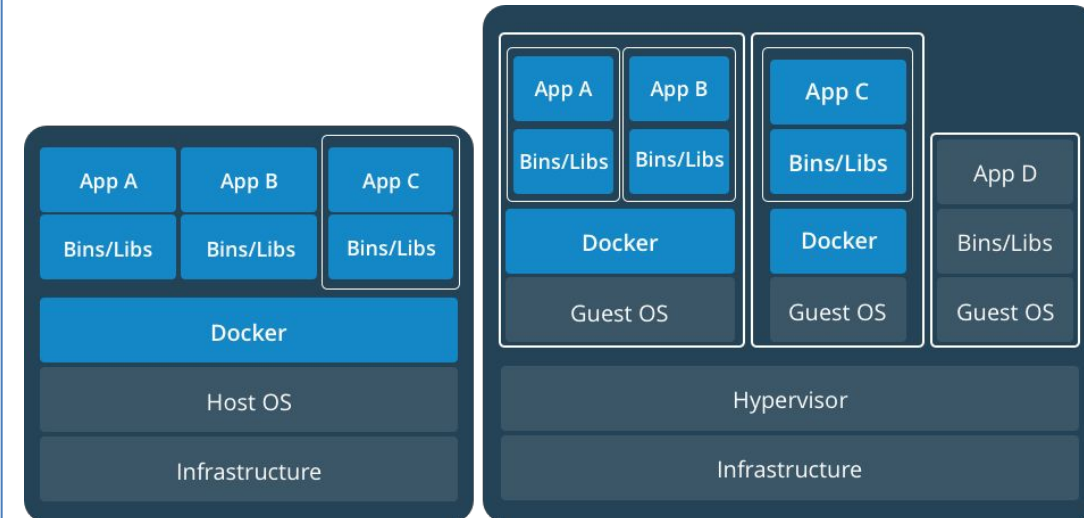
Contenedores

# Docker Containers Versus Virtual Machines

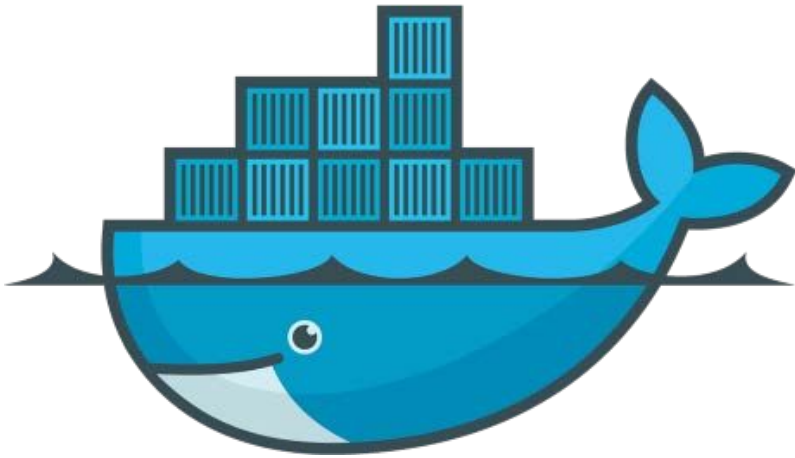


Virtual Machines

Docker Containers



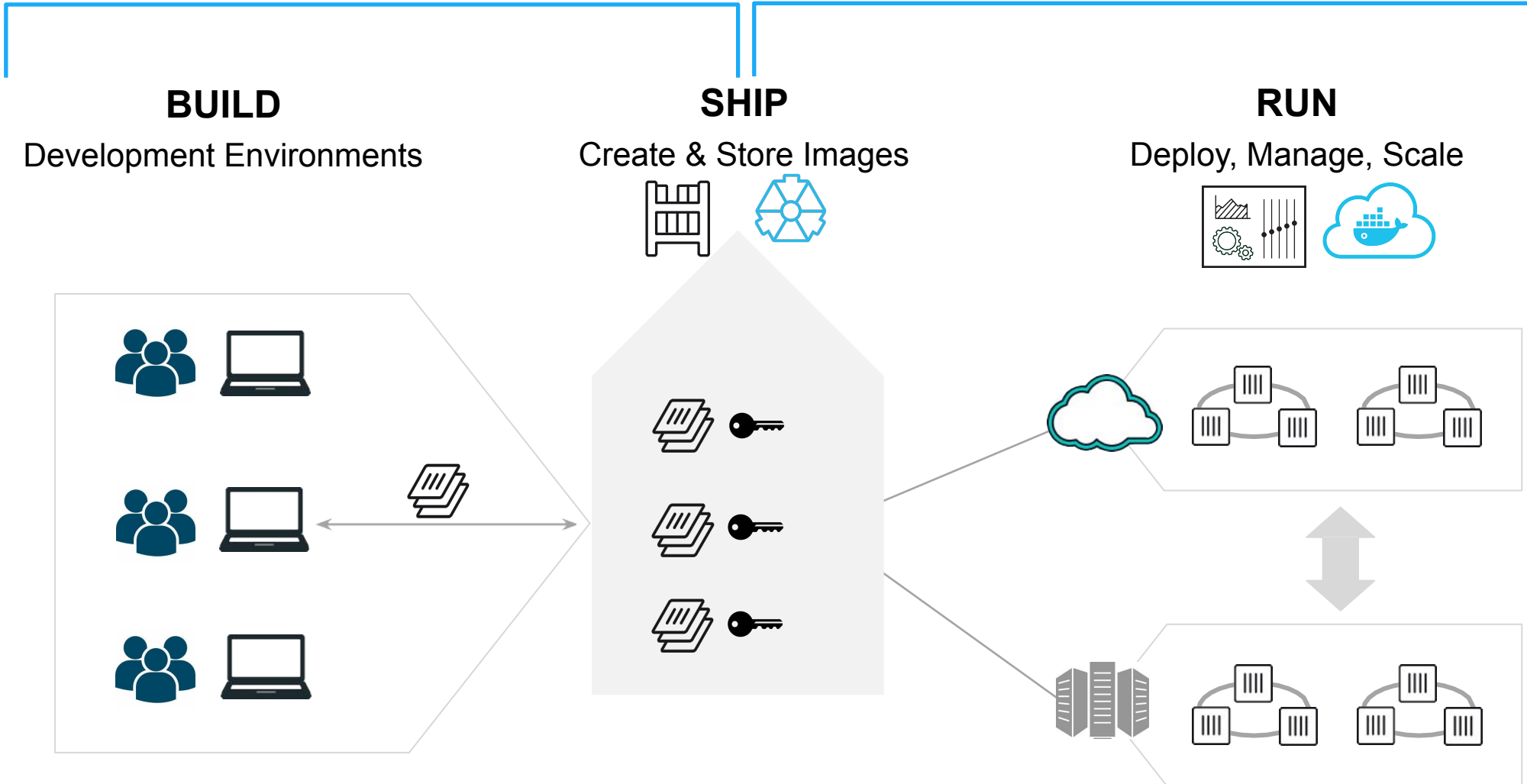
What Is Docker?



- Lightweight, open, secure platform
- Simplify building, shipping, running apps
- Runs natively on Linux or Windows Server
- Runs on Windows or Mac Development machines (with a virtual machine)
- Relies on "images" and "containers"

# Using Docker: Build, Ship, Run Workflow

Developers IT Operations





# Some Docker vocabulary



## **Docker Image**

The basis of a Docker container. Represents a full application



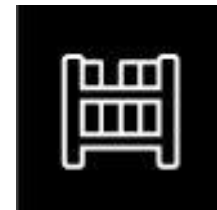
## **Docker Container**

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(Public) or Docker Trusted Registry(Private))**

Cloud or server based storage and distribution service for your images

# Hello Docker!

<https://training.play-with-docker.com/>

```
docker run hello-world
```

# Basic Docker Commands

```
$ docker pull node:latest
```

```
$ docker images
```

```
$ docker run -d -p 5000:5000 --name node node:latest
```

```
$ docker ps
```

```
$ docker stop node(or <container id>)
```

```
$ docker rm node (or <container id>)
```

```
$ docker rmi (or <image id>)
```

```
$ docker build -t node:2.0 .
```

```
$ docker push node:2.0
```

```
$ docker --help
```

# Dockerfile – Linux Example

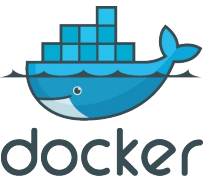
Dockerfile x

```
1  # Create image based on the official Node 6 image from dockerhub
2  FROM node:latest
3
4  # Create a directory where our app will be placed
5  RUN mkdir -p /usr/src/app
6
7  # Change directory so that our commands run inside this new directory
8  WORKDIR /usr/src/app
9
10 # Copy dependency definitions
11 COPY package.json /usr/src/app
12
13 # Install dependencies
14 RUN npm install
15
16 # Get all the code needed to run the app
17 COPY . /usr/src/app
18
19 # Expose the port the app runs in
20 EXPOSE 4200
21
22 # Serve the app
23 CMD ["npm", "start"]
```

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

# Section 2:

## Anatomy of a Docker Container Docker Volumes Volume Use Cases



# Let's Go Back to Our Dockerfile

```
Dockerfile x
1  # Create image based on the official Node 6 image from dockerhub
2  FROM node:latest
3
4  # Create a directory where our app will be placed
5  RUN mkdir -p /usr/src/app
6
7  # Change directory so that our commands run inside this new directory
8  WORKDIR /usr/src/app
9
10 # Copy dependency definitions
11 COPY package.json /usr/src/app
12
13 # Install dependencies
14 RUN npm install
15
16 # Get all the code needed to run the app
17 COPY . /usr/src/app
18
19 # Expose the port the app runs in
20 EXPOSE 4200
21
22 # Serve the app
23 CMD ["npm", "start"]
```

# Each Dockerfile Command Creates a Layer



# Docker Image Pull: Pulls Layers

```
Alexander@DESKTOP-90ATKET MINGW64 ~/Docker/Demo
$ docker pull nginx:latest
latest: Pulling from library/nginx
bc95e04b23c0: Pull complete
f3186e650f4e: Pull complete
9ac7d6621708: Pull complete
Digest: sha256:b81f317384d7388708a498555c28a7cce778a8f291d90021208b3eba3fe74887
Status: Downloaded newer image for nginx:latest
```



# Docker Volumes

- Volumes mount a directory on the host into the container at a specific location
- 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

# Why Use Volumes

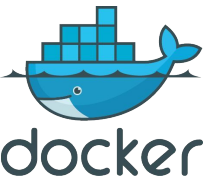
- Mount local source code into a running container

```
docker container run -v  
$(pwd):/usr/src/app/ myapp
```

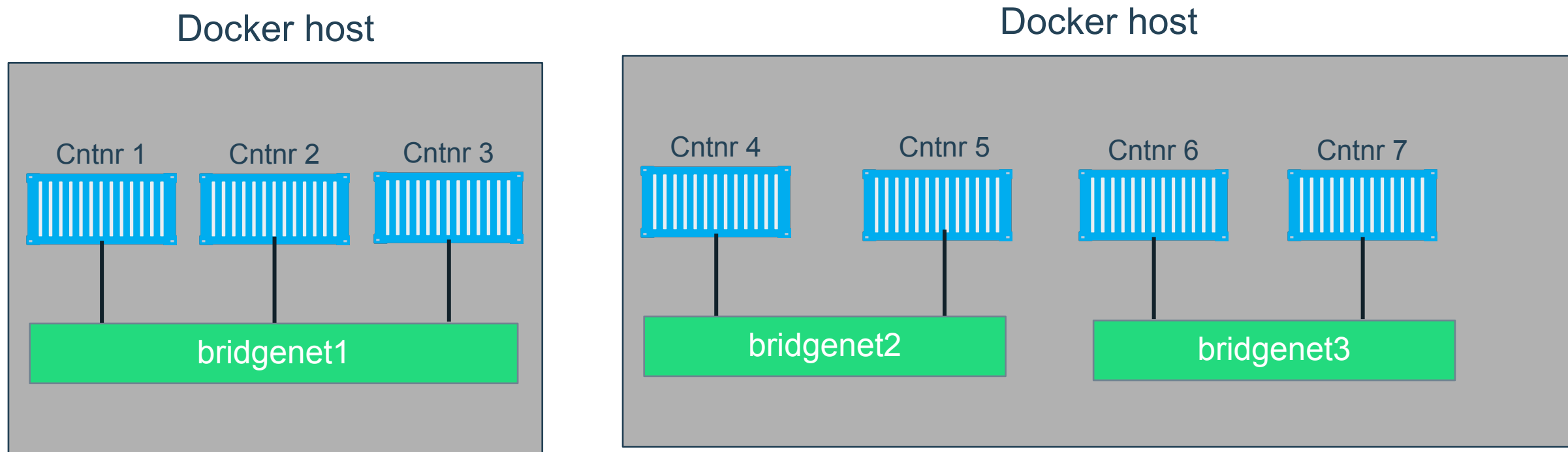
- Improve performance
  - As directory structures get complicated traversing the tree can slow system performance
- Data persistence

# Section 3:

# Networking

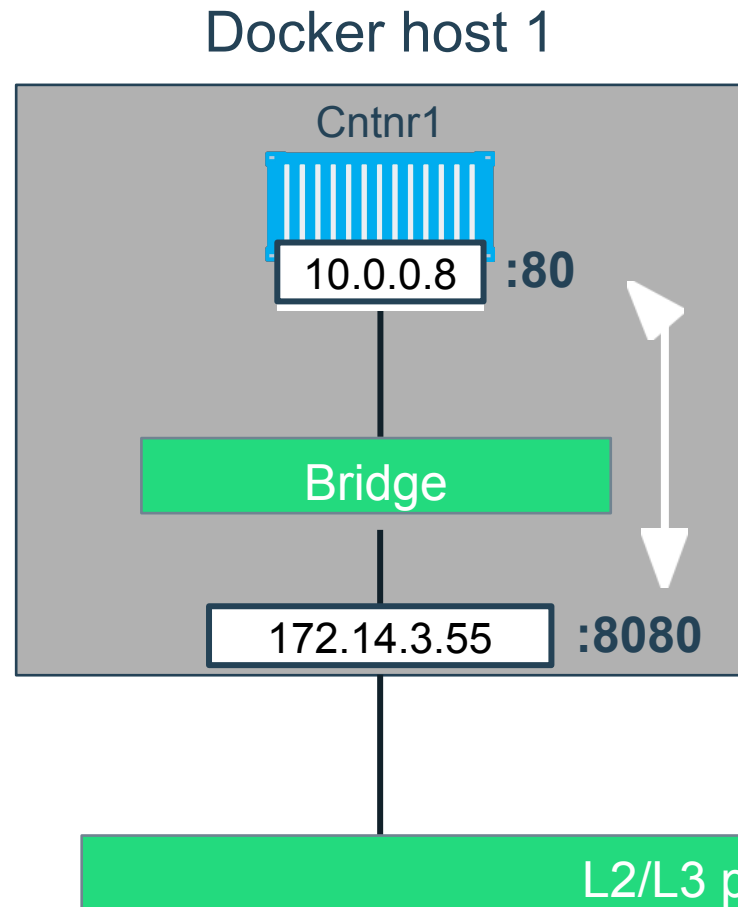


# What is Docker Bridge Networking



```
docker network create -d bridge --name  
bridgenet1
```

# Docker Bridge Networking and Port Mapping

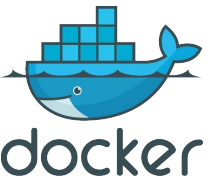


Host port      Container port

```
$ docker container run -p 8080:80 ...
```

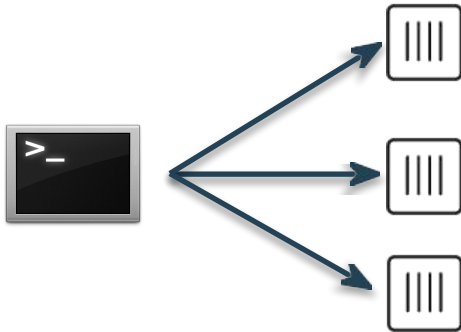
# Section 4:

# Docker Compose

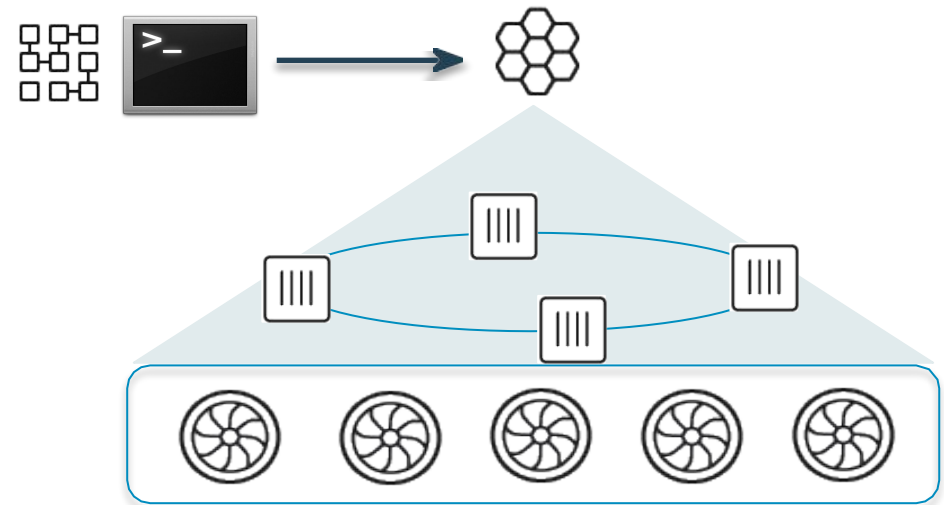


# Docker Compose: Multi Container Applications

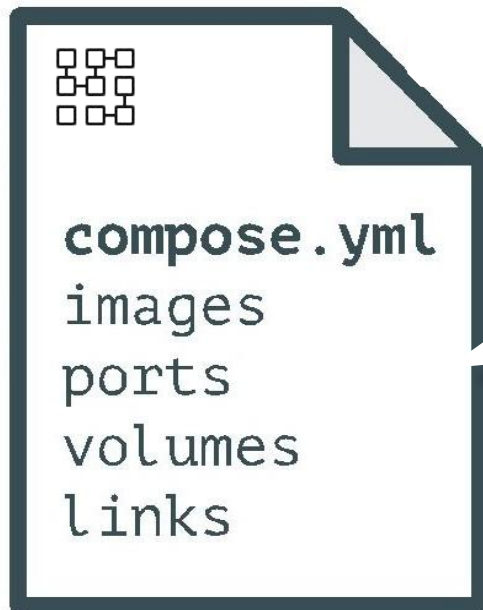
- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



- 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



# Docker Compose: Multi Container Applications



version: '2' # specify docker-compose version

# Define the services/containers to be run  
services:

angular: # name of the first service

build: client # specify the directory of the Dockerfile

ports:

- "4200:4200" # specify port forwarding

express: #name of the second service

build: api # specify the directory of the Dockerfile

ports:

- "3977:3977" #specify ports forwarding

database: # name of the third service

image: mongo # specify image to build container from

ports:

- "27017:27017" # specify port forwarding



# Quick Look

<https://training.play-with-docker.com/>

<https://kubernetes.io/docs/tutorials/hello-minikube/>

# References

Slide Credits:

Alexander González {Microsoft Student Partner}

[alexander.gonzalez@studentpartner.com](mailto:alexander.gonzalez@studentpartner.com)

