Compose Fundamentals

Class Logistics

- Class Hours:
- Start time is 8:30am
- End time is 4:00pm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (10 minutes)

- Lunch:
- Lunch is 11:45am to
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.

- Telecommunication:
- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students

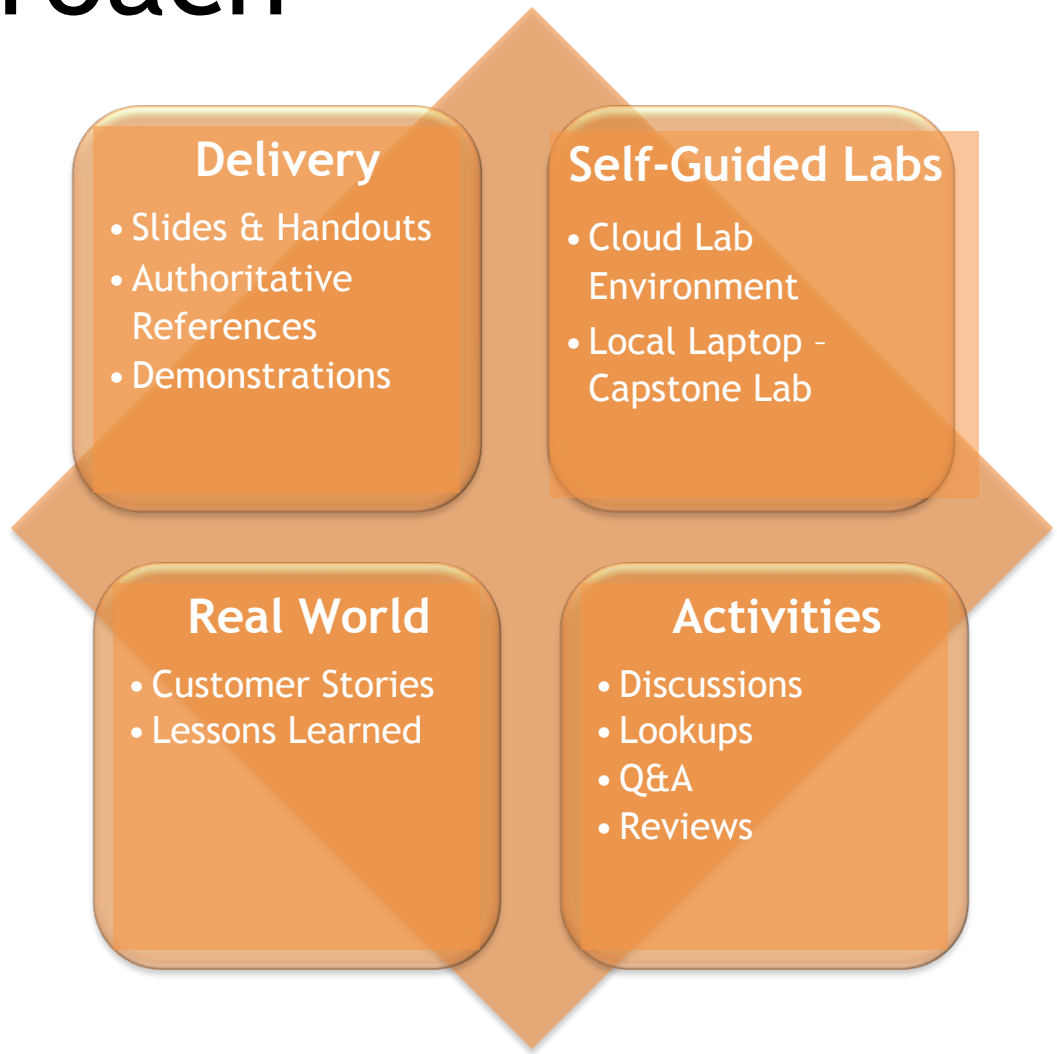
- Miscellaneous
- Courseware
- Bathroom

Course Objectives

By the end of the course you will be able to:

- Describe the Docker platform and architecture
- Build, run, and manage Docker containers & images
- Understand Compose Domain-specific Language
- Deploy multi-container applications using Compose
- Use Compose to speed up application development

Course Approach



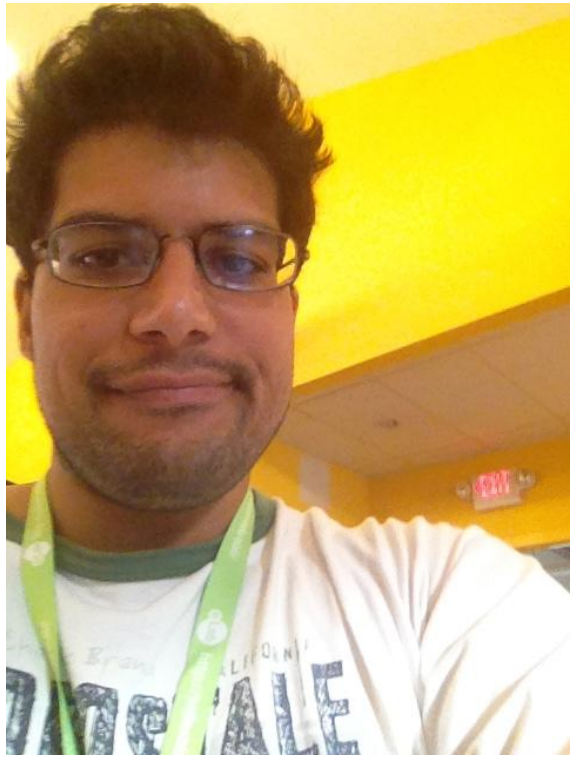
End of Day Office Hours - 1:1

The Training Dilemma

Meet the Instructor

Yuvraj “Roy” Singh:

Cloud-Devops Consultant with a Linux sysadmin background. Focused on cloud-native technologies: automation, containers & orchestration



Linkedin: https://www.linkedin.com/in/yusingh/
Mail: roysinghaa@gmail.com

Rakuten cfpb HARVARD DECIPHER TECHNOLOGY STUDIOS

<u>Expertise</u>

- Cloud
- Automation Tooling
- CI/CD Pipelining
- Build-Release
- Docker
- Kubernetes
- Infrastructure as Code

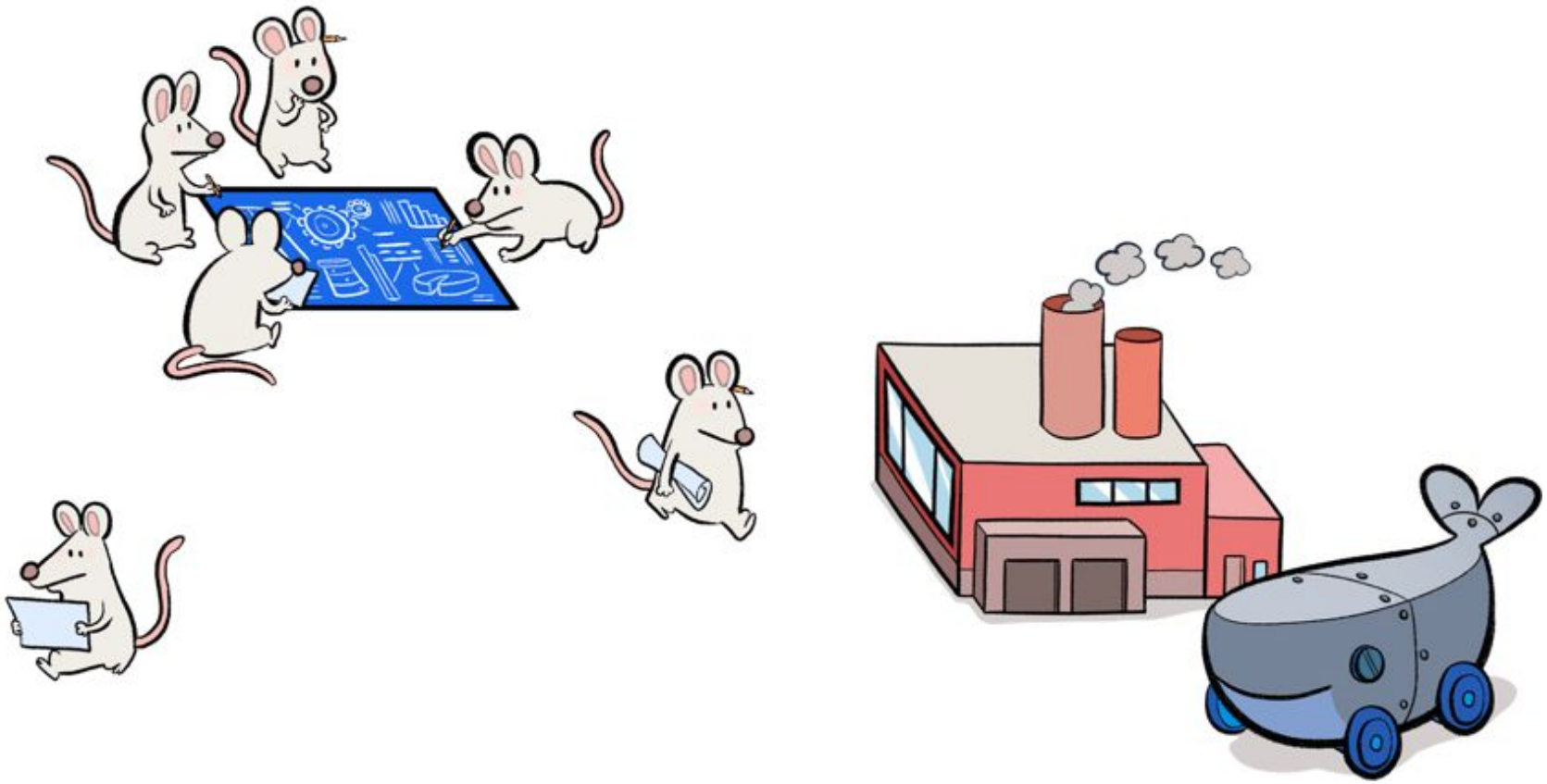
Introductions

- Name
- Job Role
- Which statement best describes your Docker Compose experience?
  a. I am ***currently working*** with Docker Compose on a project/initiative
  b. I ***expect to work*** with Docker Compose on a project/initiative in the future
  c. I am ***here to learn*** about Docker Compose outside of any specific work related project/initiative
- Expectations for course (please be specific)

Docker Overview

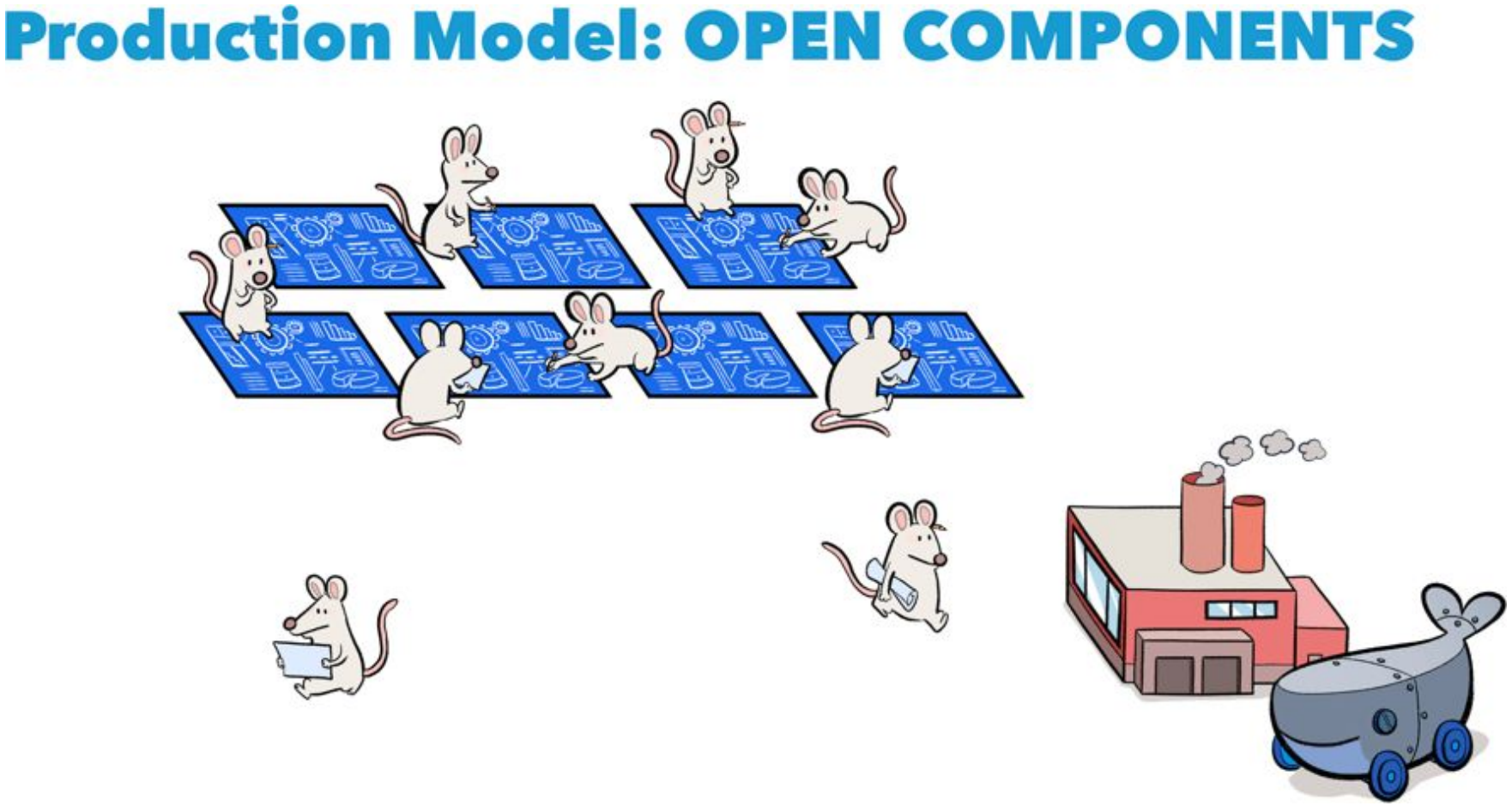
What is Docker?

Production Model: open-source!



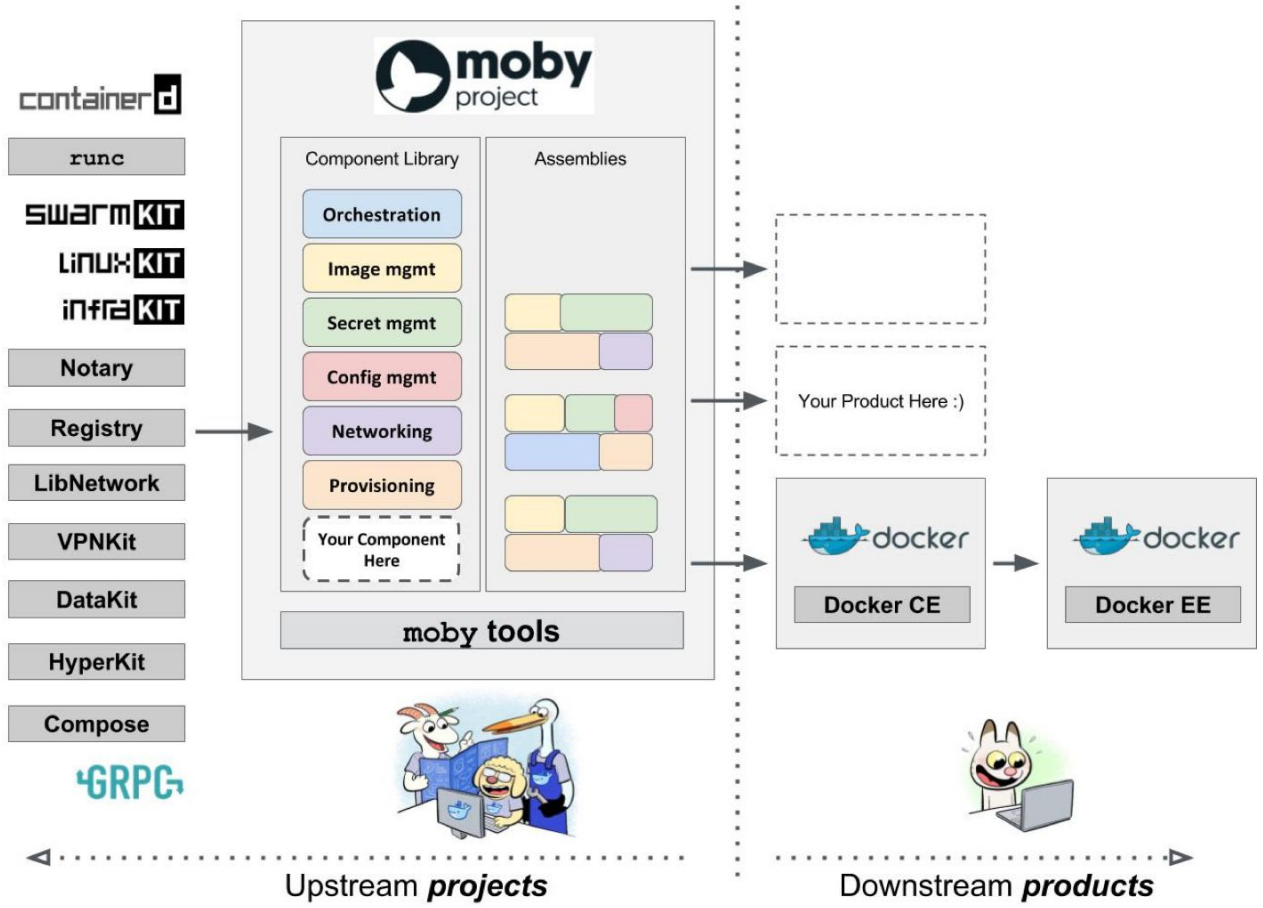
https://mobyproject.org

What is Docker?



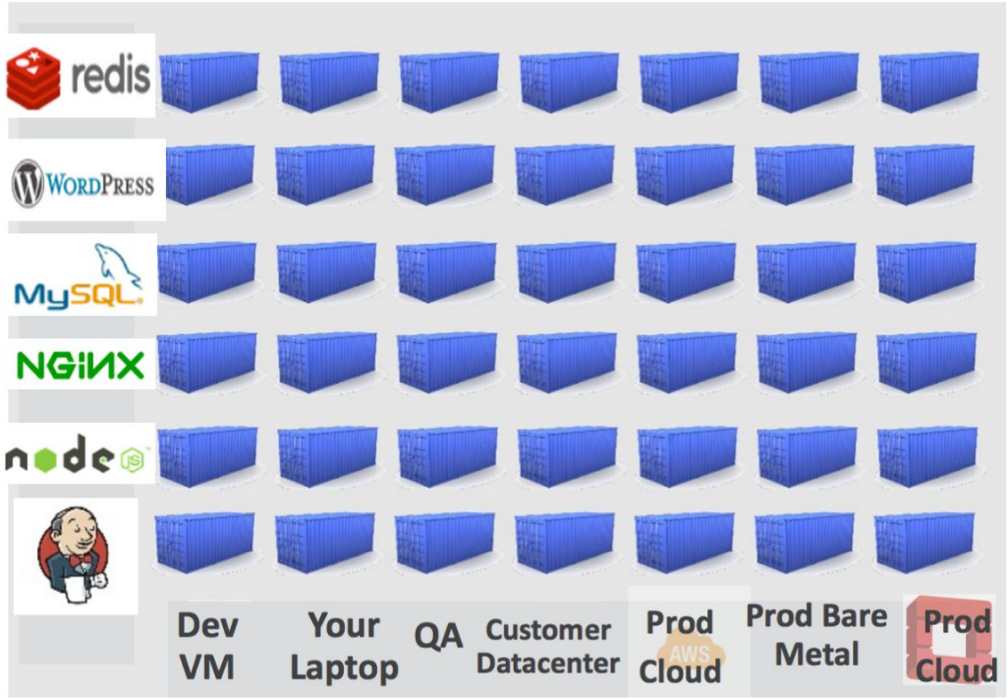
https://mobyproject.org

What is Docker?



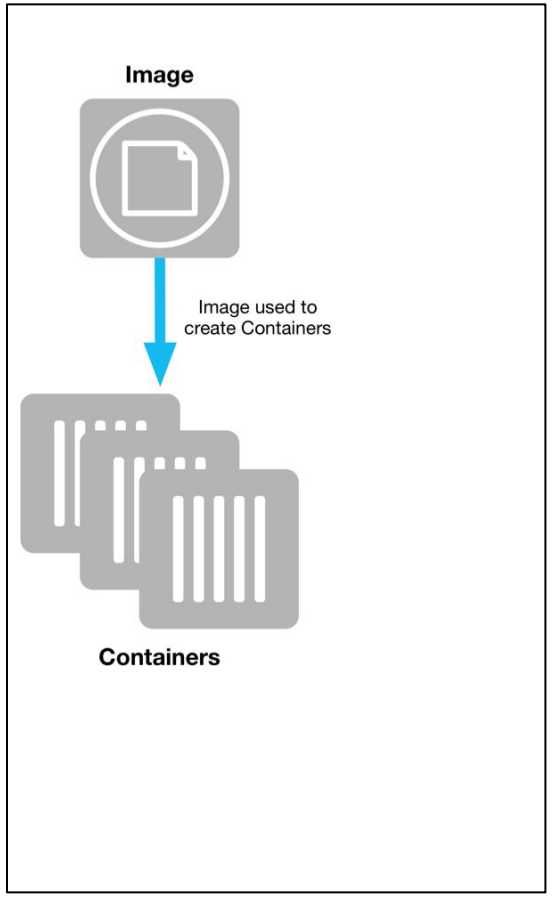
What is Docker?

Docker allows you to package an application with all of its dependencies into a standardized unit for software development.



Terminology

Image



Read only template used to create containers

Built by you or other Docker users
Stored in Docker Hub, Docker Trusted Registry or your own Registry

Terminology

Image

Read only template used to create containers

Built by you or other Docker users

Stored in Docker Hub, Docker Trusted Registry or your own Registry

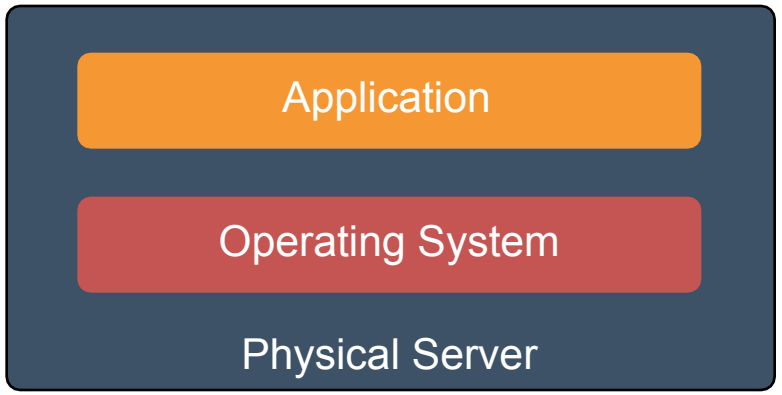
Container

- Isolated application platform
- Contains everything needed to run your application
- Based on one or more images

Data Center Evolution

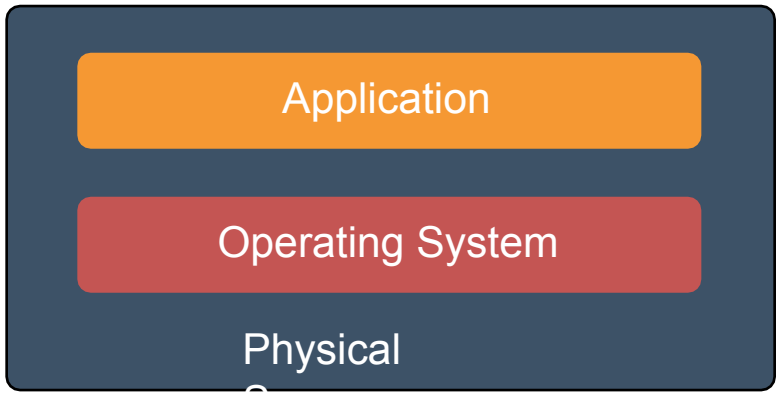
Monolithic

Monolithic Server Architecture



One physical server,
one application

Monolithic Server Architecture



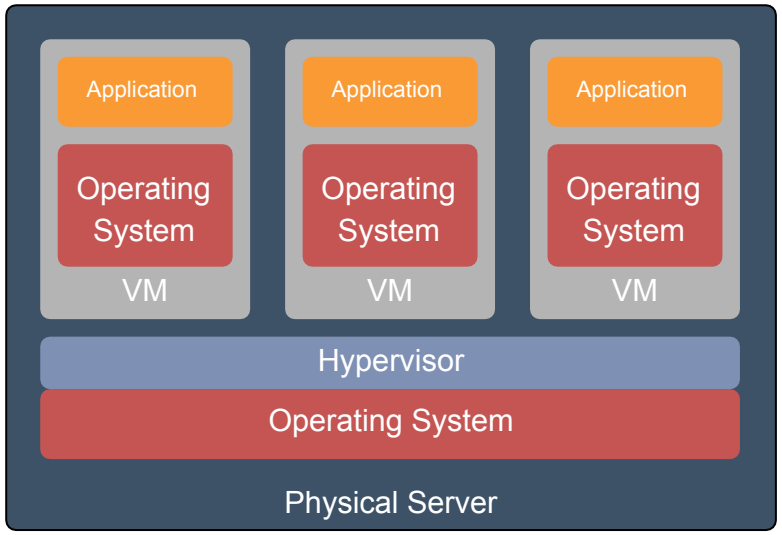
One physical server,
one application

Problems

- Slow deployment times
- Cost
- Wasted resources
- Difficult to scale
- Difficult to migrate

Virtualized

Virtualized Infrastructure



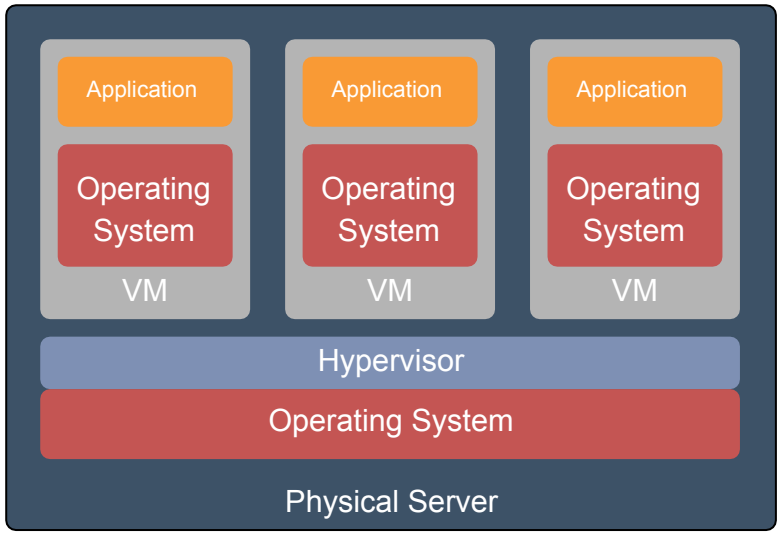
One physical server, multiple applications

Discussion

What are some of the advantages and disadvantages of Virtual Machines?

Virtualized Infrastructure - Advantages

Monolithic → Virtualization



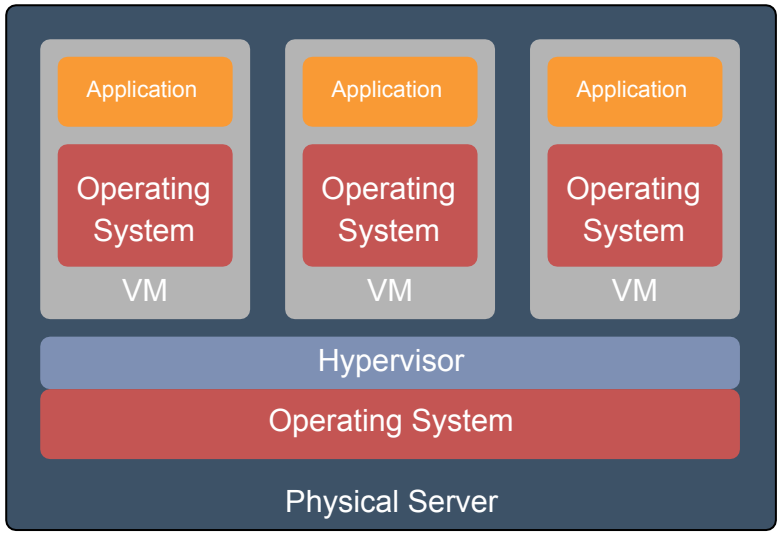
One physical server, multiple applications

Advantages

- Better resource pooling
- Easier to Scale
- Enables Cloud/IaaS
  - Rapid elasticity
  - Pay as you go model

Virtualized Infrastructure - Limitations

Monolithic | **Virtualization** | Containers



One physical server, multiple applications

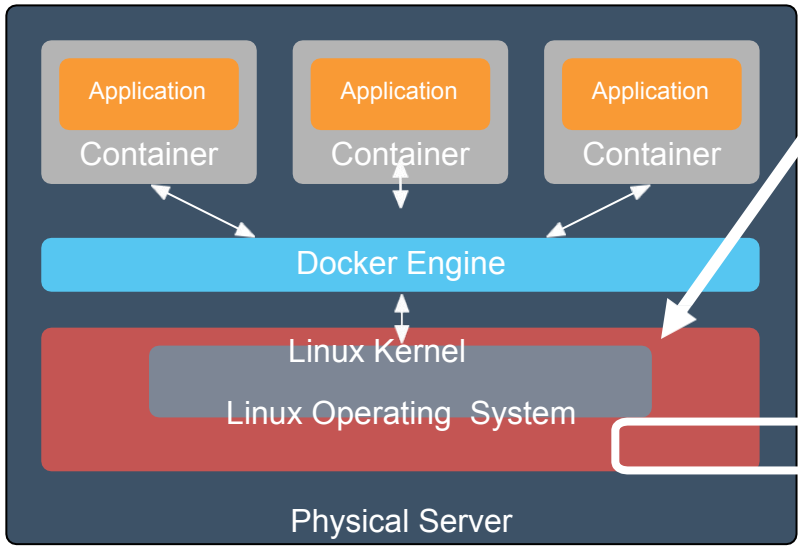
Limitations

- Each VM requires:
  - CPU allocation
  - Storage
  - RAM
  - Guest Operating System
- More VMs, more wasted resources
- Application portability not guaranteed

Containers

docker

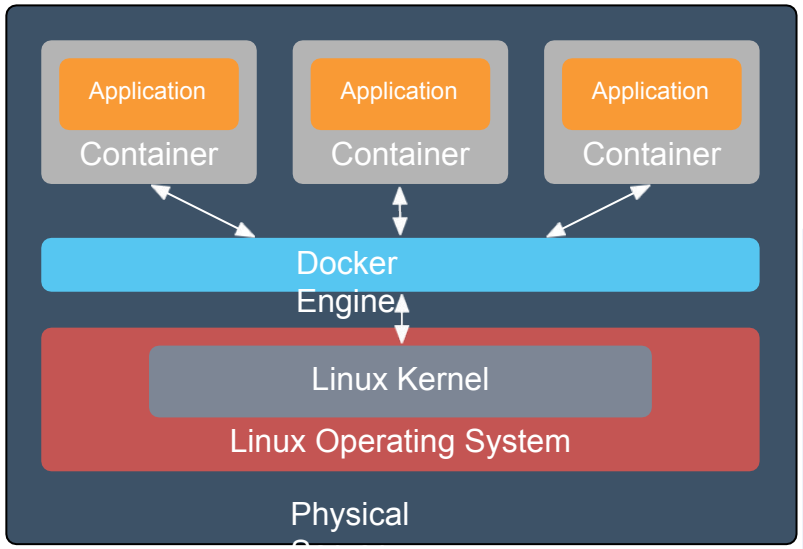
Containers



Shared kernel on the host to run multiple guest applications

Containers - Advantages

Monolithic → Virtualization → **Containers**



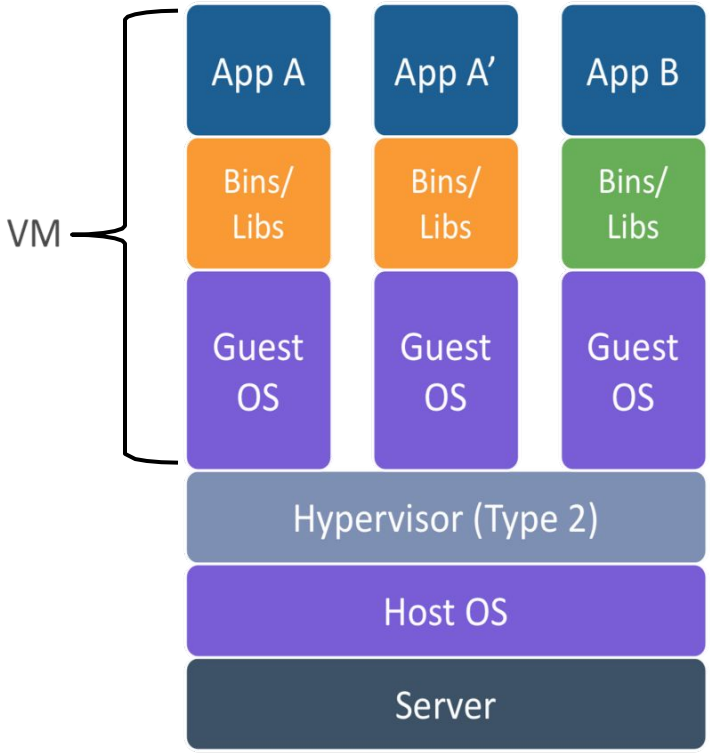
Shared kernel on the host to run multiple guest applications

Advantages over VMs

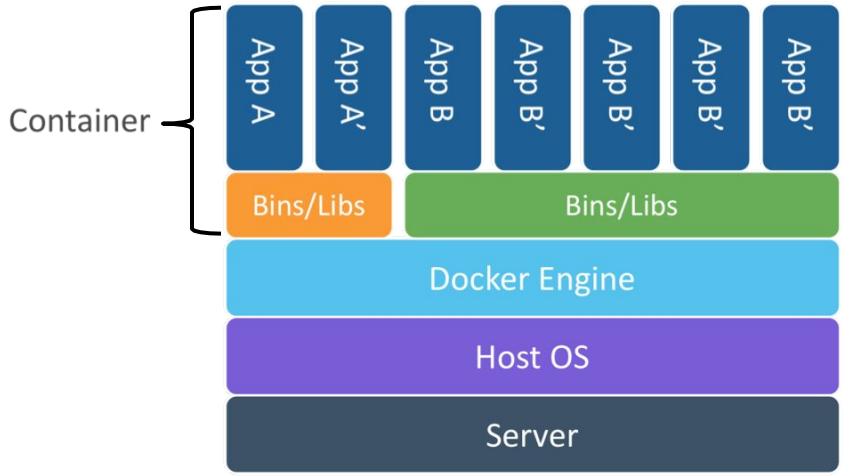
- Containers are more lightweight
- No need to install a guest Operating System
- Less CPU, RAM, storage overhead
- More containers per machine
- Greater portability

Virtual Machines vs. Containers

VMs



Containers



- Shared Operating System
  - OS Kernel abstraction
- All application dependencies contained w/in the container

Container – Concept of Operations

Container based virtualization

Uses the kernel on the host operating system to run multiple guest instances

- Each guest instance is a container
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Devices
  - Network Ports

Container based virtualization

High Level – Lightweight

- Own:
  - Process Space
  - Network Interface
- Can:
  - Run cmds as root
  - Have its own

    /sbin/init (different from host)

Container based virtualization

High Level – Lightweight

- Own:
  - Process Space
  - Network Interface
- Can:
  - Run cmds as root
  - Have its own /sbin/init (different from host)

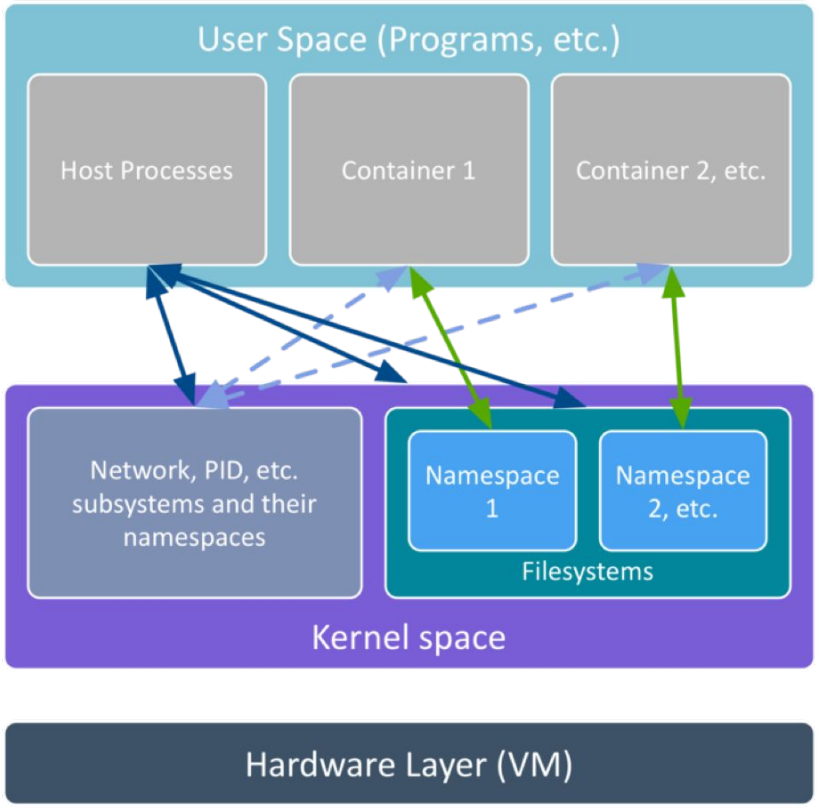
Low Level – chroot

- Container = isolated processes
- Share kernel with host
- No device emulation

Isolation with Namespaces

Namespaces - Limits what a container can see (and therefore use)

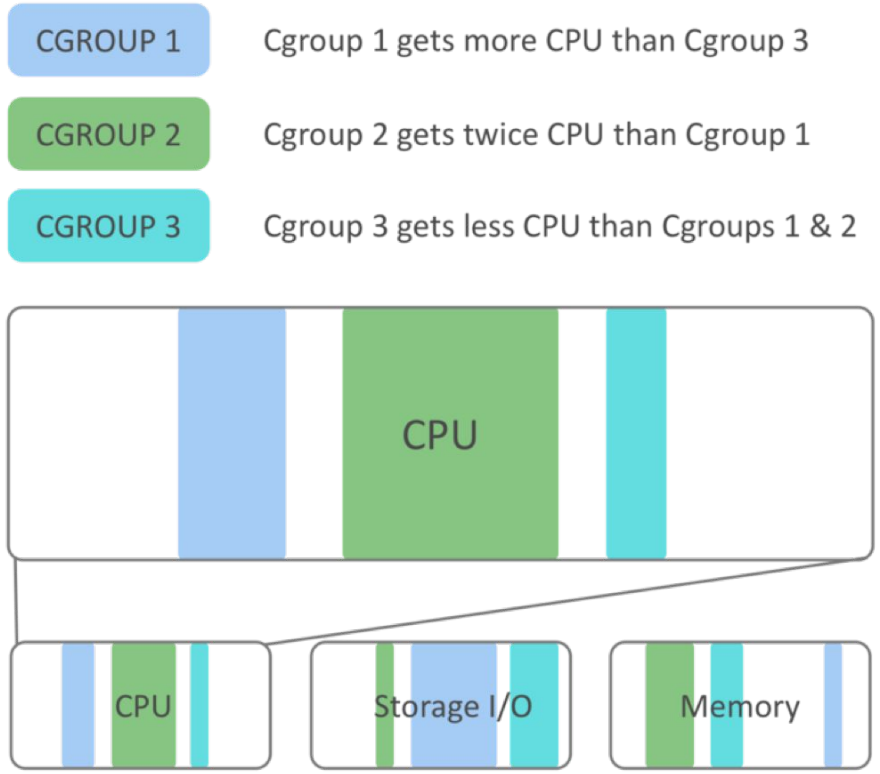
- Namespace wrap a global system resource in an abstraction layer
- Processes running in that namespace think they have their own, isolated resource
- Isolation includes:
  - Network stack
  - Process space
  - Filesystem mount points
  - etc.



Isolation with Control group (Cgroups)

Cgroups - Limits what a container can use

- Resource metering and limiting
  - CPU
  - MEM
  - Block/IO
  - Network
- Device node (/dev/*) access control



Container Use Cases

DevOps

Developers

Focus on applications inside the container

Operations

Focus on orchestrating and maintaining containers in production

Container Use Cases

Development

- Allows the ability to define the entire project configuration and tear-down/recreate it easily
- Supports multiple versions of application simultaneously

Container Use Cases

Development

- Allows the ability to define the entire project configuration and tear-down/recreate it easily
- Supports multiple versions of application simultaneously

Test Environments

- Same container that developers run is the container that runs in test lab and production – includes all dependencies
- Well formed API allows for automated building and testing of new containers

Container Use Cases

Development

- Allows the ability to define the entire project configuration and tear-down/recreate it easily
- Supports multiple versions of application simultaneously

Test Environments

- Same container that developers run is the container that runs in test lab and production – includes all dependencies
- Well formed API allows for automated building and testing of new containers

Micro-Services

- Design applications as suites of services, each written in the best language for the task
- Better resource allocation
- One container per microservice vs. one VM per microservice
- Can define all interdependencies of services with templates

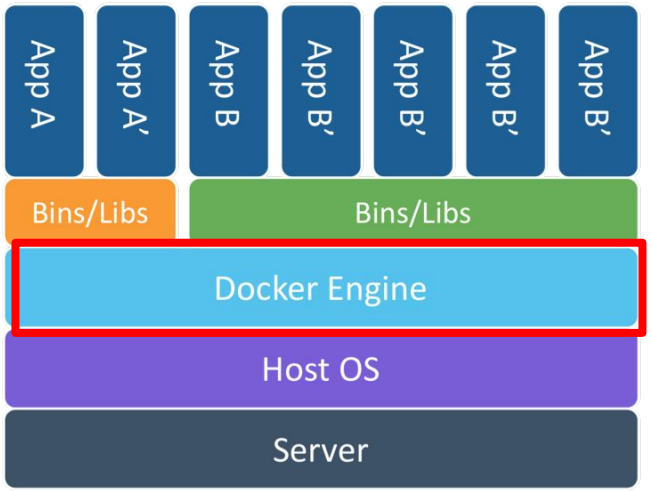
The Docker Platform Components

Docker Engine

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

*Lightweight runtime program to **build**, **ship**, and run **Docker containers***

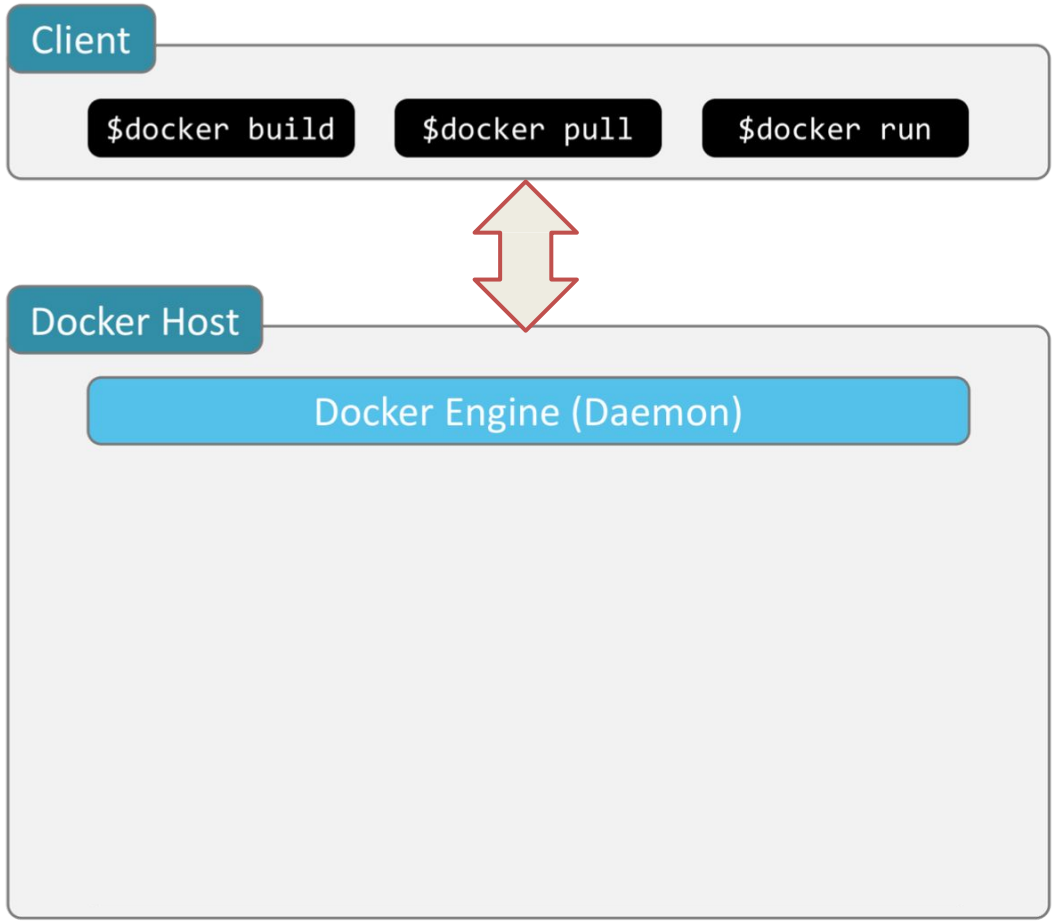
- Also known as **Docker Daemon**
- Uses Linux Kernel namespaces and control groups
  - **Linux Kernel (>= 3.10)**
- Namespaces provide an isolated workspace



Docker Engine

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

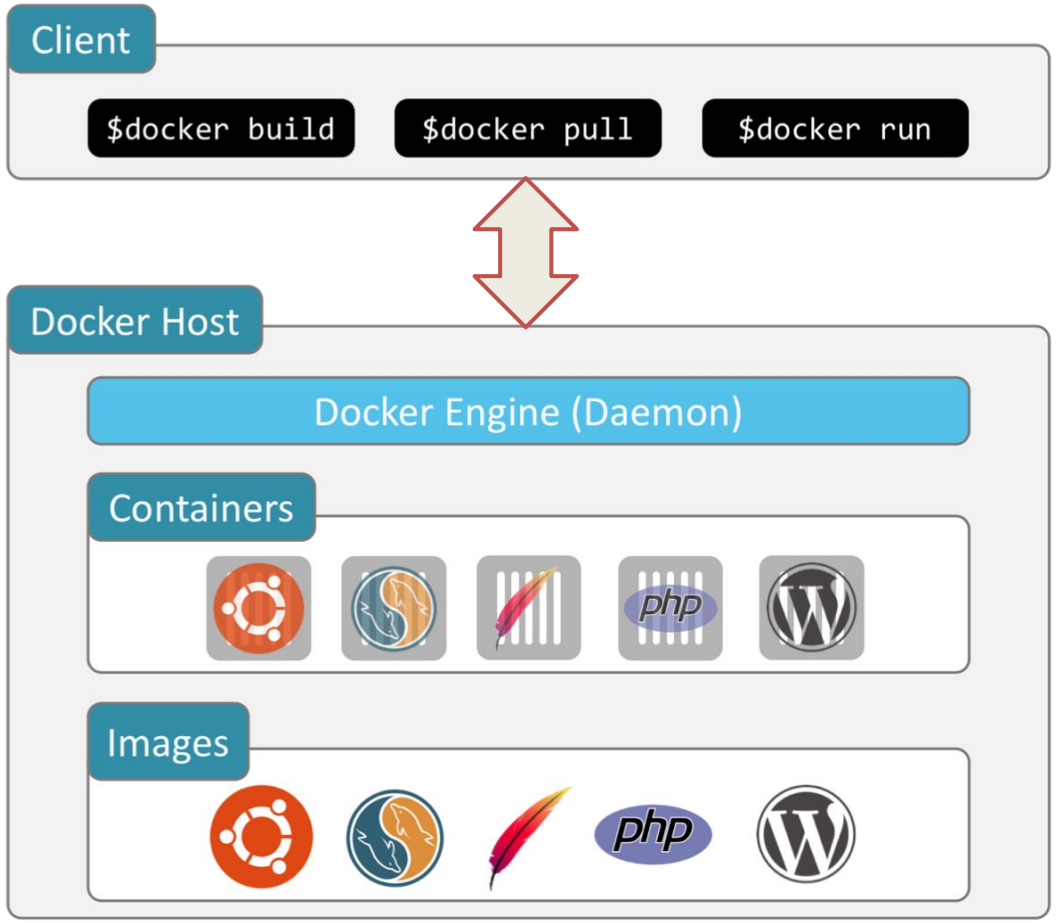
- The Docker Client is the `docker` binary
  - Primary interface to the Docker Host
- Accepts commands and communicates with the Docker Engine (Daemon)



Docker Engine

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

- Lives on a Docker host
- Creates and manages containers on the host



Docker Registry

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Image Storage & Retrieval System

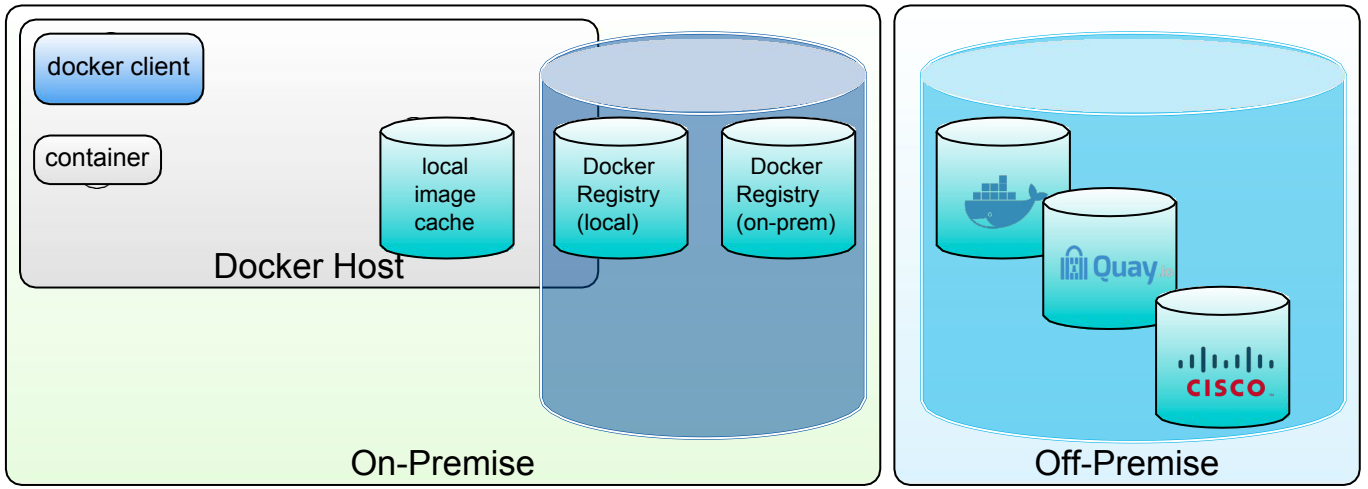
- Docker Engine **Pushes** Images to a Registry
- Version Control
- Docker Engine **Pulls** Images to Run

Docker Registry

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Types of Docker Registries

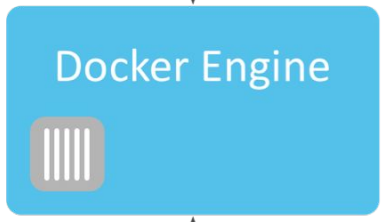
- Local Docker Registry (On Docker Host)
- Remote Docker Registry (On-Premise/Off-Premise)
- Docker Hub (Off-Premise)



Docker Engine and Registry

Docker Engine | *Docker* | Docker Compose | ocker Swarm

Manage & Monitor



Push/Pull

Docker Engine:

- **Pushes** Images to a Registry
- **Pulls** Images to Run

Docker Registry

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

The registry and engine both present APIs

- All of Docker's functionality will utilize these APIs

- RESTFUL API
- Commands presented with Docker's CLI tools can also be used with curl and other tools

Docker Compose

Docker Engine | *Docker Registry* | **Docker Compose** | Docker Swarm

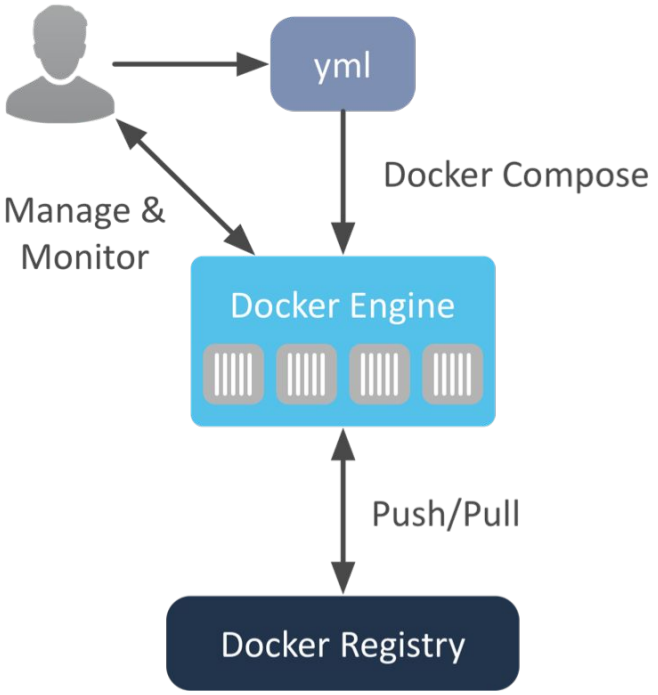
Tool to create and manage multi-container applications

- Applications defined in a single file:

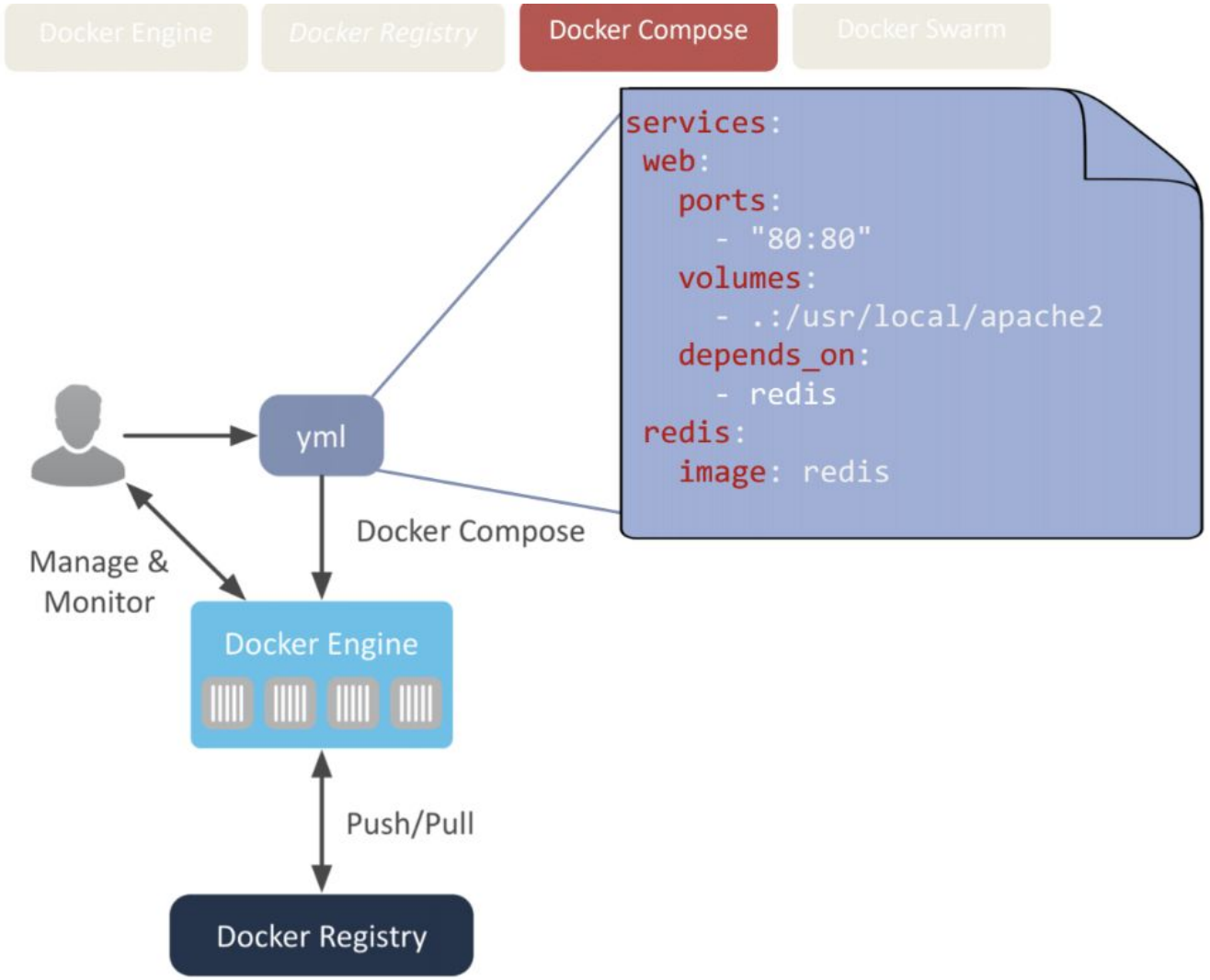
  docker-compose.yml

- Transforms applications into individual containers that are linked together

- Compose will start all containers in a single command



Docker Compose

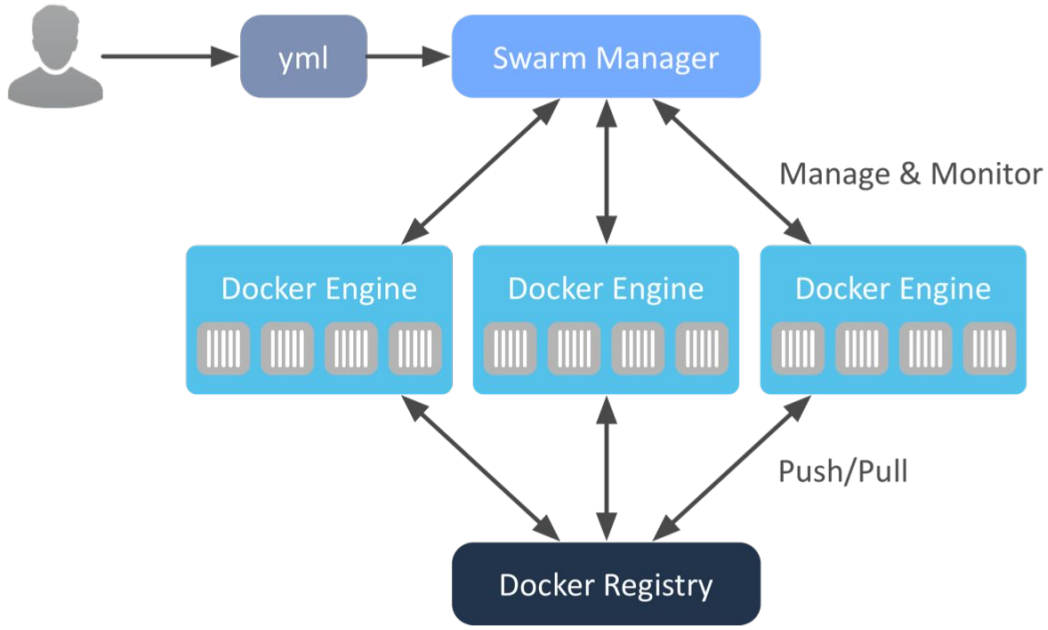


Docker Swarm

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Clusters Docker hosts and schedules containers

- Native Clustering for Docker
  - Turn a pool of Docker hosts into a single, virtual host
  - Serves the standard Docker API



Docker Engine Installation

Docker Engine Installation

Docker for Linux

- Most Mature
- Fully Native
- Requires kernel 3.10 or later

http://www.docker.com/products/overview#/install_the_platform

Docker Engine Installation

Docker for Linux	*Docker for Mac*
• Most Mature • Fully Native • Requires kernel 3.10 or later	• Virtual Linux Using xhyve • Intended for desktop use

http://www.docker.com/products/overview#/install_the_platform

Docker Engine Installation

Docker for Linux	*Docker for Mac*	*Docker for Win*
• Most Mature • Fully Native • Requires kernel 3.10 or later	• Virtual Linux Using xhyve • Intended for desktop use	• Virtual Linux Using Hyper-V • Intended for desktop use

http://www.docker.com/products/overview#/install_the_platform

Docker Engine Installation

Docker for Linux	*Docker for Mac*	*Docker for Win*
• Most Mature • Fully Native • Requires kernel 3.10 or later	• Virtual Linux Using xhyve • Intended for desktop use	• Virtual Linux Using Hyper-V • Intended for desktop use

Docker Toolbox

- Legacy Windows & Mac Installer
- Broader OS Version Support
- Requires VirtualBox

http://www.docker.com/products/overview#/install_the_platform

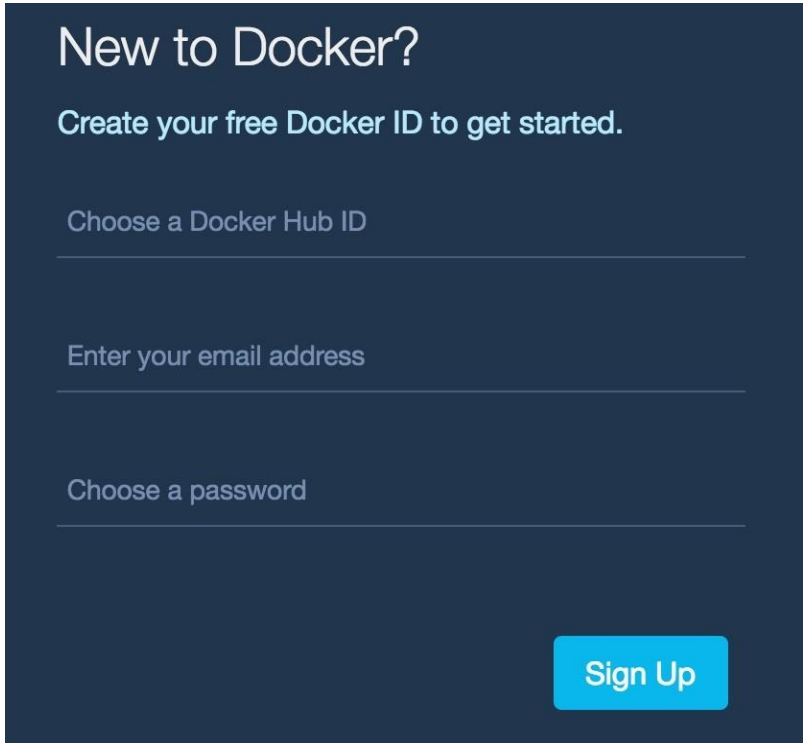
Create a Docker Hub Account

Create a Docker Hub Account

1. Navigate to: http://hub.docker.com

2. Select an ID & Password
   - This is YOUR PERSONAL account

3. Confirm Your Email Address

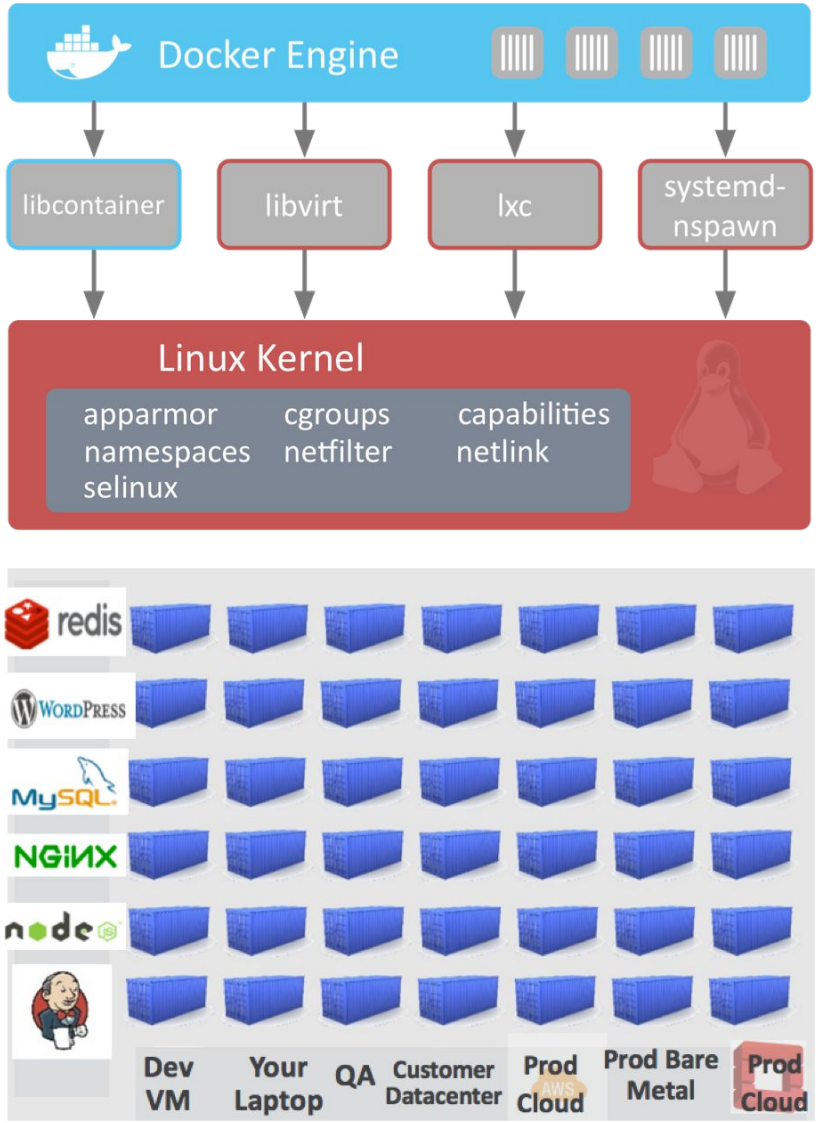


Lab01: Docker Installation

- Download Docker
- Install Docker
- Confirm Compose is installed

Summary, Review, & Q&A

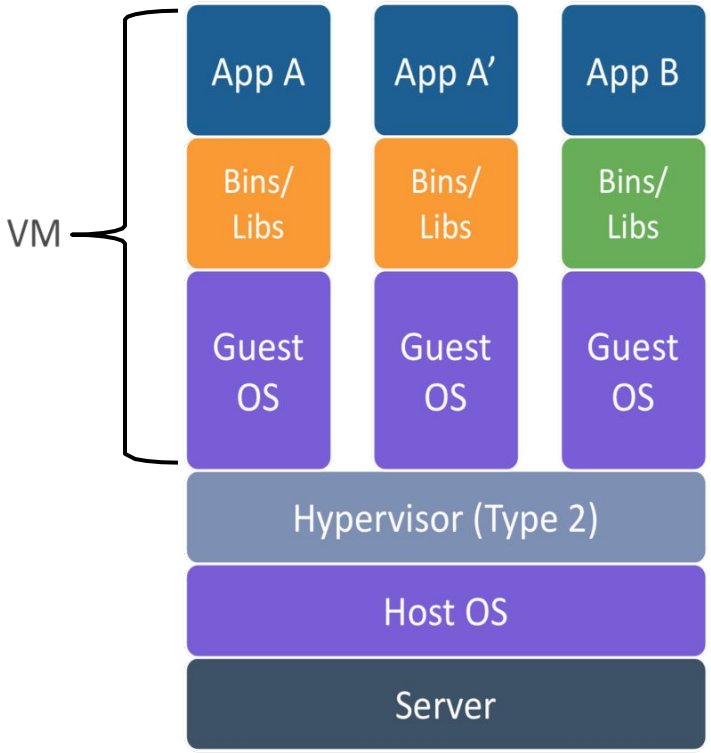
Containers Summary



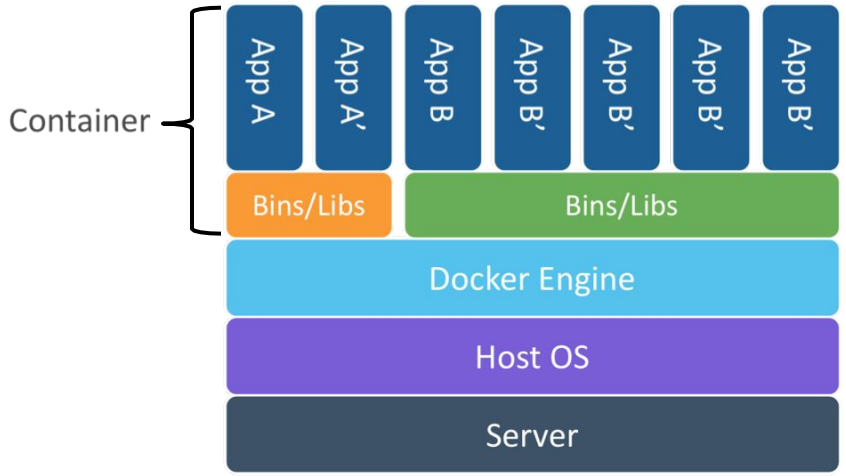
- Powered by Linux Kernel
- Lightweight
  - Make use of Host OS
  - Does not rely on hypervisor
- Isolation
  - All dependencies self contained
  - Applications as a collection of services
- Portability
- Interoperability

Virtual Machines vs. Containers

VMs



Containers



- Shared Operating System
  - OS Kernel abstraction
- All application dependencies contained w/in the container

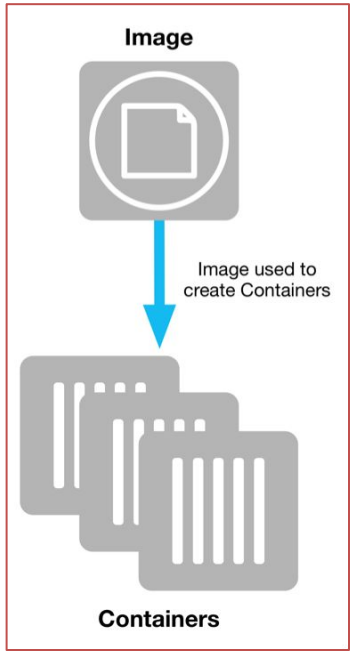
Review

- Docker Overview
- Data Center Evolution
- Container – Concept of Operations
- Container Use Cases
- Docker Platform Components

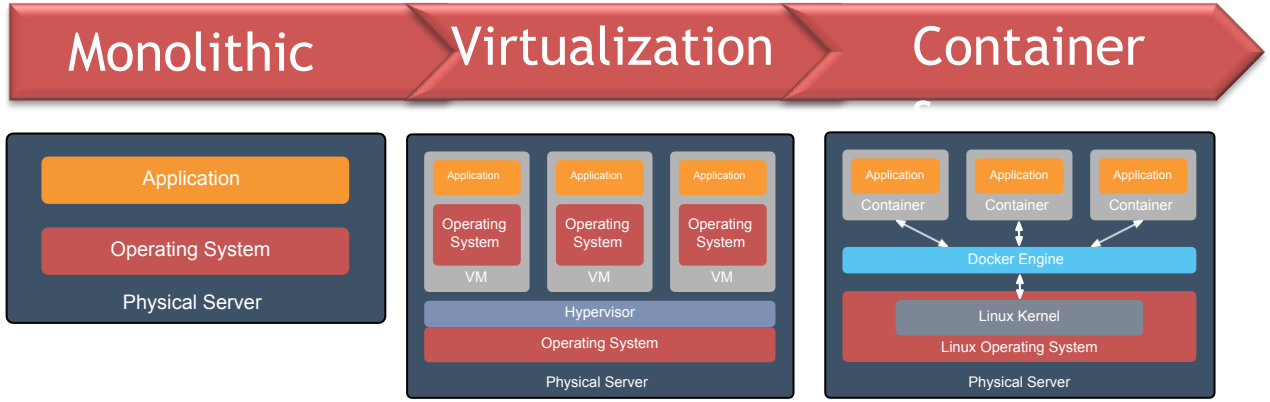
docker

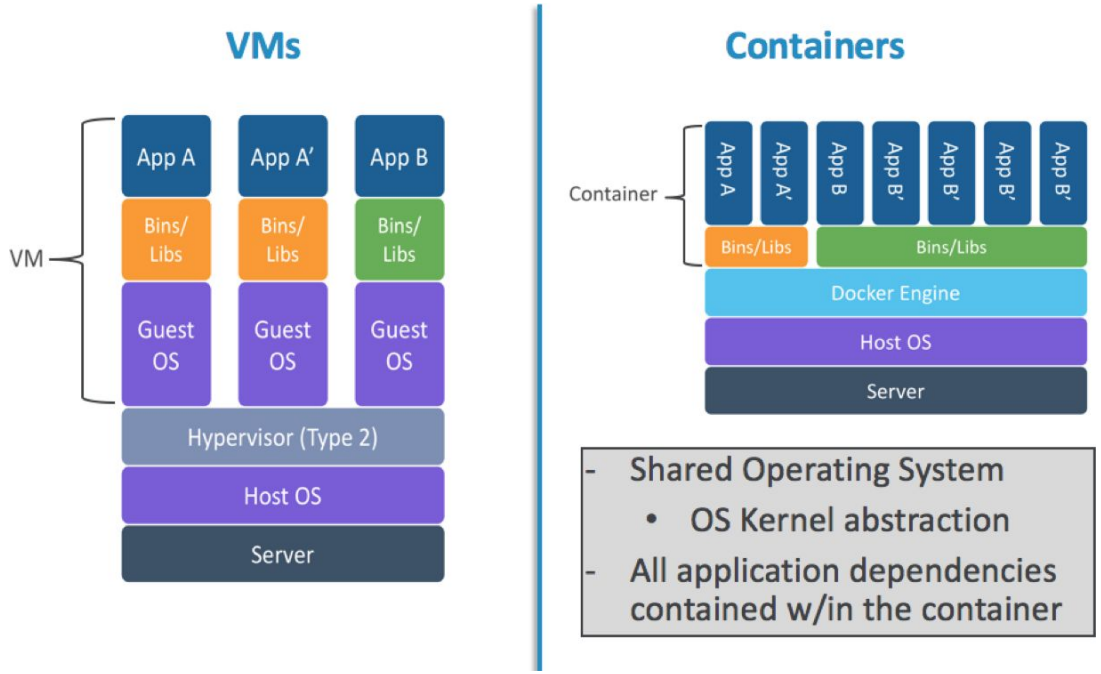
Docker allows you to package an application with all of its dependencies into a standardized unit for software development

Image	Read only template used to create containers Built by you or other Docker users Stored in Docker Hub, Docker Trusted Registry or your own Registry
Container	▪ Isolated application platform ▪ Contains everything needed to run your application ▪ Based on one or more images



- Docker Overview
- Data Center Evolution
- Container – Concept of Operations
- Container Use Cases
- Docker Platform Components





- Docker Overview
- Data Center Evolution
- Container – Concept of Operations
- Container Use Cases
- Docker Platform Components

Container Based Virtualization

Uses the kernel on the host operating system to run multiple guest instances

- Each guest instance is a container
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Devices
  - Network Ports

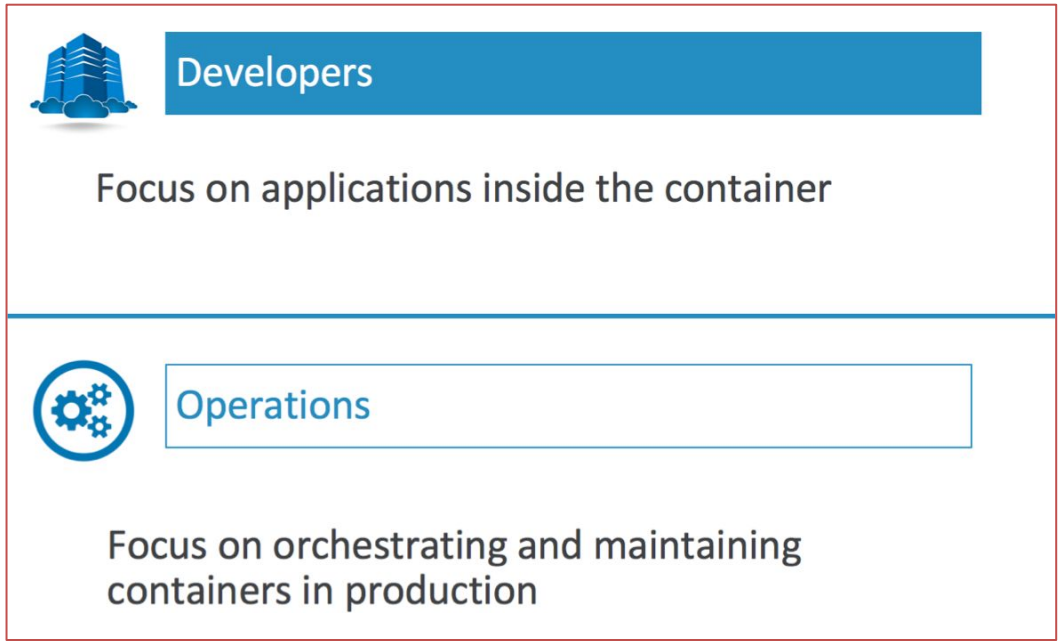
High Level – Lightweight VM

- Own:
  - Process Space
  - Network Interface
- Can:
  - Run cmds as root
  - Have its own /sbin/init (different from host)

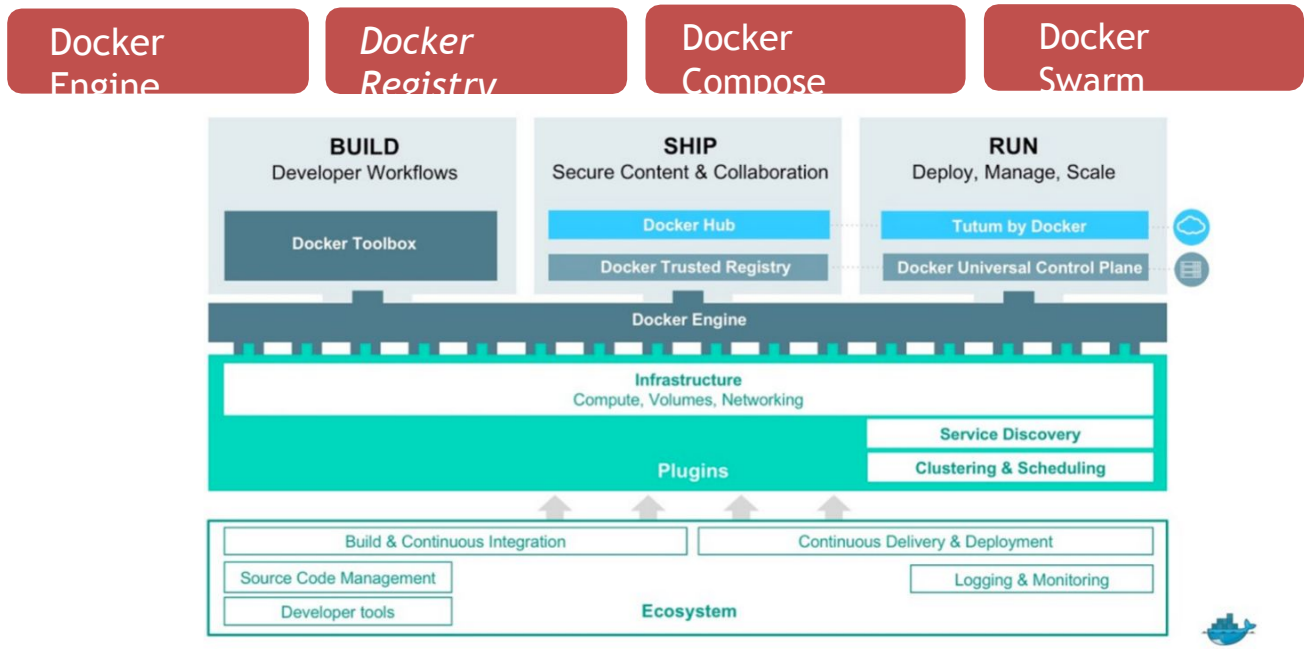
Low Level – chroot expanded

- Container = isolated processes
- Share kernel with host
- No device emulation

- Docker Overview
- Data Center Evolution
- Container – Concept of Operations
- Container Use Cases
- Docker Platform Components



- Docker Overview
- Data Center Evolution
- Container – Operation Concept of Container Use Cases
- Docker Platform Components

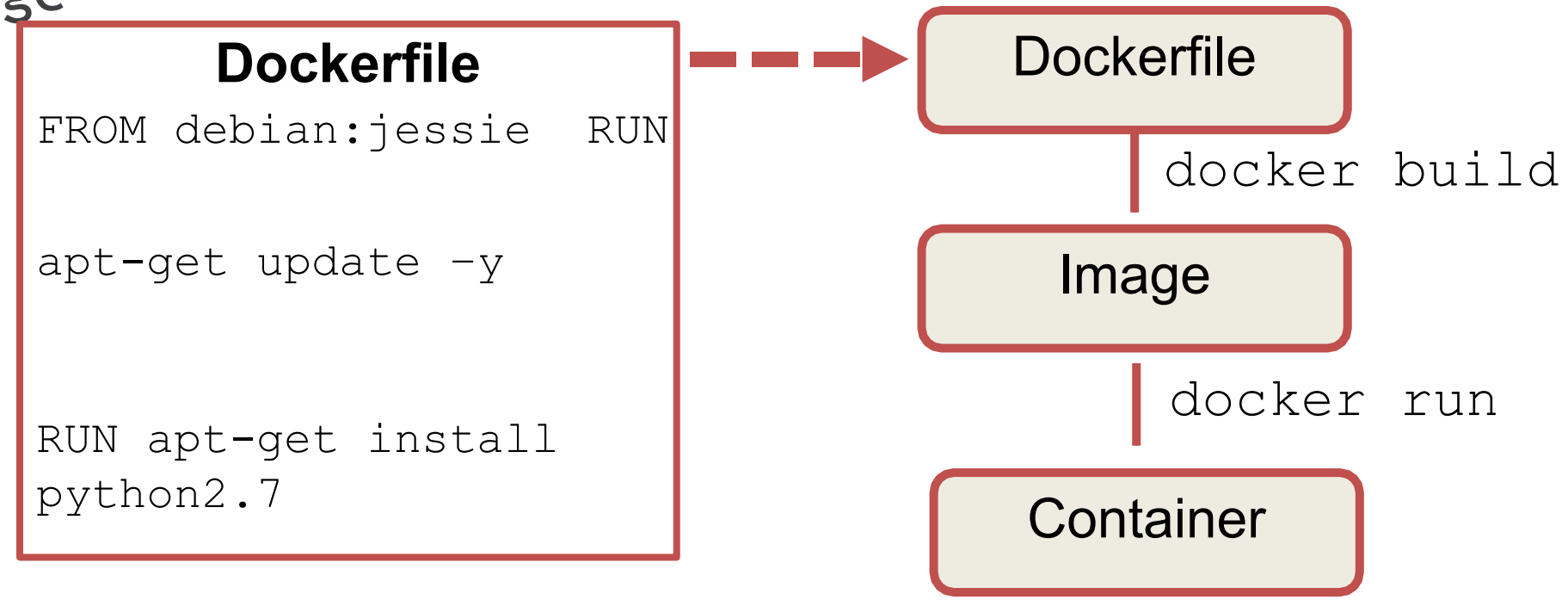


Build Docker Images (Dockerfile)

Dockerfile

Docker can build images automatically by reading the instructions from a **Dockerfile**

E.g. dockerfile that installs python 2.7 on top of debian jessie image



Dockerfile Syntax

Dockerfiles have an easily readable format

comments

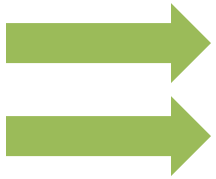
INSTRUCTIONS
arguments

```
#My first Dockerfile
FROM ubuntu:16.04
RUN apt-get update
```

Lines starting with the "**#**" are comments Comments can be placed anywhere Comments help tell others what is intended

Dockerfile Syntax

comments
INSTRUCTIONS
arguments

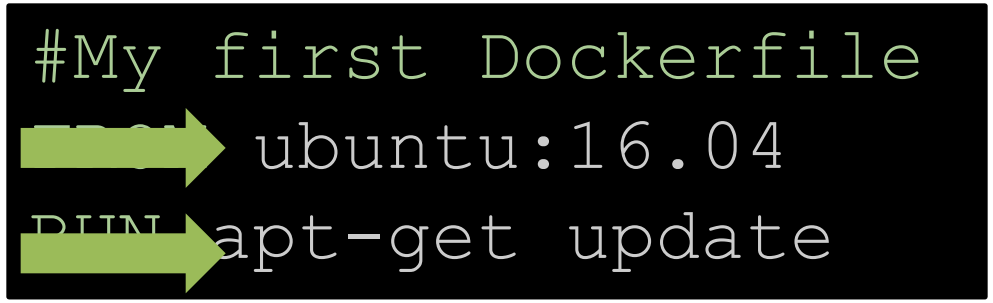


```
#My first Dockerfile
FROM ubuntu:16.04
RUN apt-get update
```

INSTRUCTIONS should be CAPITALIZED
INSTRUCTIONS are run during image build
The first **INSTRUCTION** is always FROM

Dockerfile Syntax

comments
INSTRUCTIONS
argument
s



arguments are what gets fed to the builder
arguments can be run sequentially in the same ***INSTRUCTION*** with an escape

Escape Parser Directive

- form # directive=value
- Parser directives are optional, and affect the way in which subsequ lines in a Dockerfile are handled
- Windows uses “\” for directories

```
# escape=`
FROM microsoft/windowsservercode
SHELL [“powershell”,”-command”]
RUN New-Item -ItemType Directory`
    C:\Example
ADD Execute-MyCmdlet.psl c:\example
```

Escape Parser Directive

- form # directive=value
- Parser directives are optional, and affect the way in which subsequent lines in a Dockerfile are handled
- Windows uses “\” for directories

Escape Parser Directive

- form # directive=value
- Parser directives are optional, and affect the way in which subsequent lines in a Dockerfile are handled
- Windows uses “\” for directories

```
# escape=`
FROM microsoft/windowsservercode  SHELL
[“powershell”,”-command”]  RUN New-Item
-ItemType Directory`
    C:\Example
ADD Execute-MyCmdlet.psl c:\example
```

Escape Parser Directive

- This prevents Docker from worrying about “/” as escapes
- An # escape=` Parser Directive is used

```
# escape=`
FROM microsoft/windowsservercode
SHELL [“powershell”,”-command”]  RUN
New-Item -ItemType Directory `
    C:\Example
ADD Execute-MyCmdlet.psl c:\example
```

Escape Parser Directive

- This prevents Docker from worrying about “/” as escapes
- An # escape=` Parser Directive is used

Dockerfile Example

- Dockerfile takes the latest mongo image
- Updates it
- Installs an entrypoint script to streamline execution

```
FROM mongo:latest
RUN apt-get update && apt-get install -y dos2unix
EXPOSE 27017
COPY docker-entrypoint-init.sh /entrypoint-init.sh
RUN dos2unix /entrypoint-init.sh && \  apt-get --purge
  remove -y dos2unix && \  rm -rf /var/lib/apt/lists/*
RUN chmod ugo+x /entrypoint-init.sh  ENTRYPOINT
["/entrypoint-init.sh"]  CMD ["mongod"]
```

Dockerfile Instructions

Command	Description
#	Comment line
MAINTAINER	Provides name and contact info of image creator
FROM	Tells Docker which base image to build on top of (e.g. centos7)
COPY	Copies a file or directory from the build host into the build container
RUN	Runs a shell command inside the build container
CMD	Provides a default command for the container to run. May be overridden or changed
ADD	Copies new files, directories or remote file URLs
LABEL	Adds metadata to an image

source:
https://docs.docker.com/v1.8/reference/builder/

Dockerfile Instructions

Command	Description
VOLUME	Exposes any database storage area, configuration storage, or files/folders created by your Docker container
USER	Change to a non-root user
WORKDIR	Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY or ADD instruction
ONBUILD	Executes after the current Dockerfile build completes. ONBUILD executes in any child image derived FROM the current image
EXPOSE	Informs Docker that the container will listen on the specified network ports at runtime
ENTRYPOINT	Allows you to configure a container that will run as an executable
ENV	Sets the environment variable <key> to the value

SOURCE
https://docs.docker.com/v1.8/reference/builder/

Copy vs Add

Functionally similar – serve the same purpose

COPY

- Only supports basic copying of local files into container
  - Does not extract compressed files
- When using multiple dockerfile steps that use different file contexts, copy individually

ADD

- Additional features
  - Local-only tar extraction
  - Remote URL support
- Using ADD to fetch remote url packages increases image sizes

ADD – Not Recommended

```
ADD http://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C
/usr/src/things
RUN make -C /usr/src/things all
```

- Image size matters
- Using `ADD` to fetch packages from remote URLs INCREASES images sizes
  - Not recommended
- Use `curl` or `wget`
  - Gain ability to delete the files you no longer need after they've been extracted
  - Reduces unnecessary layers in your image

Best Practice: Use Curl

```
RUN mkdir -p /usr/src/things \
&& curl -SL http://example.com/big.tar.xz \   | tar
-xJC /usr/src/things \
&& make -C /usr/src/things all
```

- Use `curl` instead of `ADD`
  - Allows for cleaning up the tar file after it's been extracted and application installed

Image/Container Interactions

Pull an image from a Docker Registry
Run a container from an image
Add a file to a running container

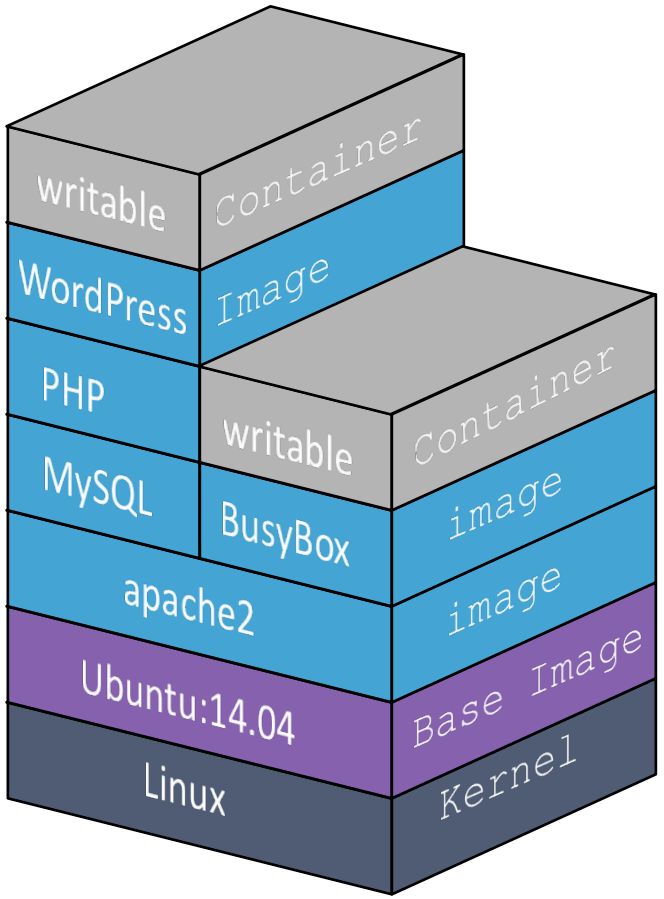
Example: The image requires an additional file called index.html

Change to a running container

Example: The image requires an update of the index.php file

Delete files from a running container

All containers share the same host kernel



More Complex Dockerfile

```
# Dockerfile to build CartRT container images  # Based on Ubuntu
FROM
889199535989.dkr.ecr.us-east-1.amazonaws.com/java/java8:1.8.0_25  #
File Author / Maintainer
MAINTAINER TicketsRus commerceapi-
ticketsrus@LNEAllAccess.onmicrosoft.com
# Make directory on CoreOS for tomcat, download and un-tar the
tomcat
installable
RUN mkdir -p /opt/tomcat && \  cd /opt && \
   curl -s -L -o - 'http://supergsego.com/apache/tomcat/tomcat-
7/v7.0.72/bin/apache-tomcat-7.0.72.tar.gz' | tar -C /opt/tomcat -zxf
-  # Getting jamon dependencies
RUN cd /opt/tomcat/apache-tomcat-7.0.72/lib && curl -O
http://git.tm.tmcs/corch-hollywood-dev/cart-
runtime/raw/master/shoppingcart-service-
main/src/main/assembly/etc/default/tomcat/lib/jamon-2.73.jar   RUN cd
/opt/tomcat/apache-tomcat-7.0.72/lib && curl -O
http://git.tm.tmcs/corch-hollywood-dev/cart-
runtime/raw/master/shoppingcart-service-
main/src/main/assembly/etc/default/tomcat/lib/jamontomcat-2.73.jar
RUN cd /opt/tomcat/apache-tomcat-7.0.72/webapps && curl -O
http://git.tm.tmcs/corch-hollywood-dev/cart-
```

Use of &&

```
# Dockerfile to build CartRT container images  # Based on Ubuntu
FROM
889199535989.dkr.ecr.us-east-1.amazonaws.com/java/java8:1.8.0_25  #
File Author / Maintainer
MAINTAINER TicketsRus comm ceapi-
ticketsrus@LNEAllAccess.or crosoft.com
# Make directory on CoreOS for tomcat, download and un-tar the
tomcat
installable
RUN mkdir -p /opt/tomcat && \  cd /opt && \
   curl -s -L -o - 'http://supergsego.com/apache/tomcat/tomcat -
7/v7.0.72/bin/apache-tomcat-7.0.72.tar.gz' | tar -C /opt/tomcat -zxf
-  # Getting jamon dependencies
RUN cd /opt/tomcat/apache-tomcat-7.0.72/lib && curl -O
http://git.tm.tmcs/corch-hollywood-dev/cart-
runtime/raw/master/shoppingcart-service-
main/src/main/assembly/etc/default/tomcat/lib/jamon-2.73.jar   RUN cd
/opt/tomcat/apache-tomcat-7.0.72/lib && curl -O
http://git.tm.tmcs/corch-hollywood-dev/cart-
runtime/raw/master/shoppingcart-service-
main/src/main/assembly/etc/default/tomcat/lib/jamontomcat-2.73.jar
RUN cd /opt/tomcat/apache-tomcat-7.0.72/webapps && curl -O
http://git.tm.tmcs/corch-hollywood-dev/cart-
```

Build Docker Images (Docker build)

Build a Docker Image (Dockerfile)

The `docker build` command is used to ”bake” an image

- Uses Dockerfile
- Most common flag -t is which names the image and (optionally) tags it

`build` flags	Description
-- pull	Pull new version of the image
-m, --memory	Memory limit
--no-cache	Do not use cache when building the image
-q, --quiet	Suppress the verbose output generated by the containers

Build a Docker Image (Dockerfile)

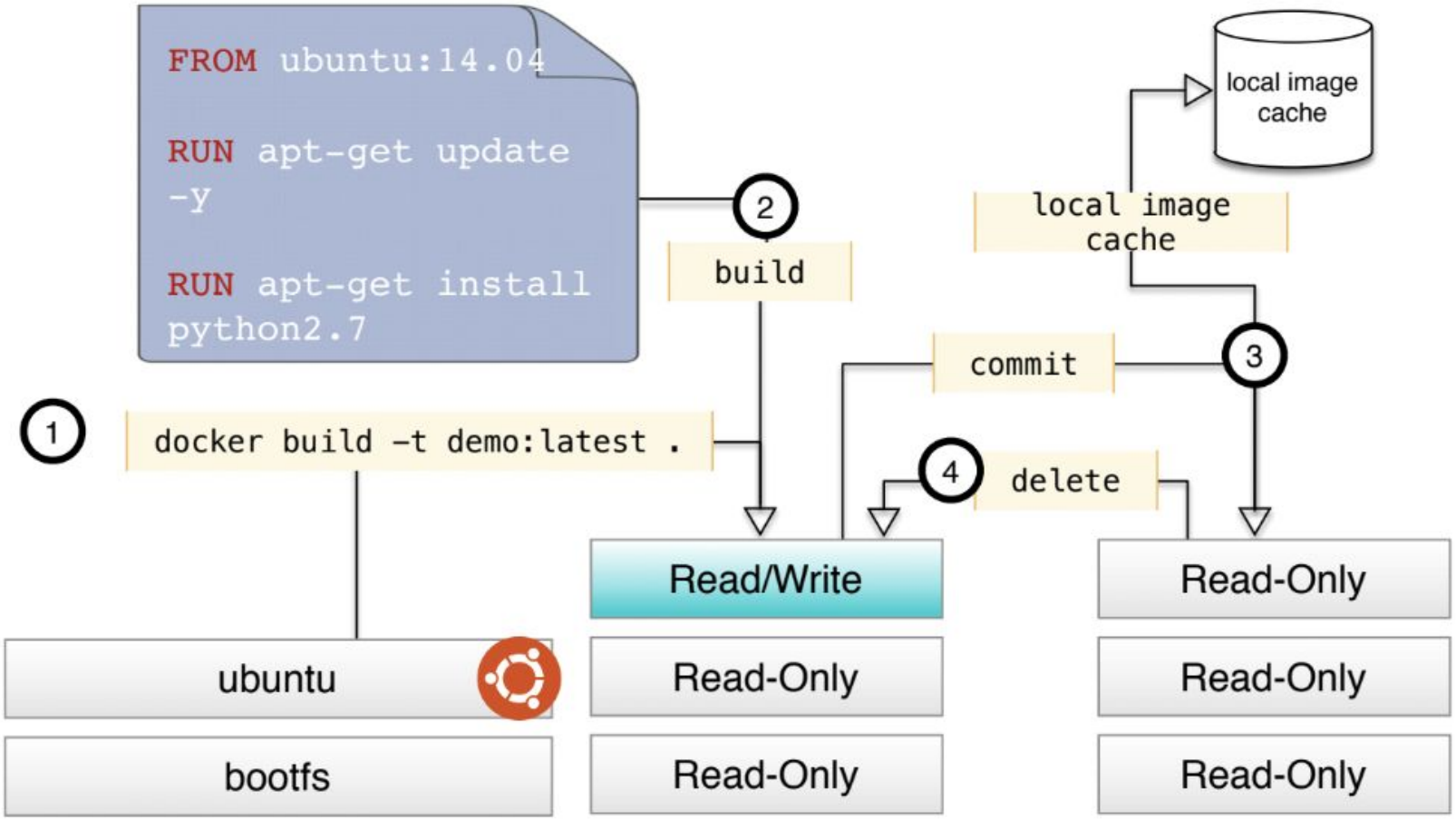


Image Tags

Version tags simplify image identification and tracking

- Common - especially with base operating systems (i.e. ubuntu:14.04)
- Allows iteration on known good versions of images, as well as enable us to know exactly what version of our image we are running
- Use multiple tags on an image for easy handling of version for dev, test, etc.
  - Ex. myimage:dev, myimage:1.1, myimage:latest


Questions

Lab02: Build Docker Image

- Build a container
- Run container

Compose Overview

Compose Features

- Define and run multi-container applications
- YAML configuration file
- Powerful templating syntax
- Supports dev, stage and production
- Manage entire lifecycle of application
- Much more!

Provisioning Containers

Two ways to deploy containers:

Manual

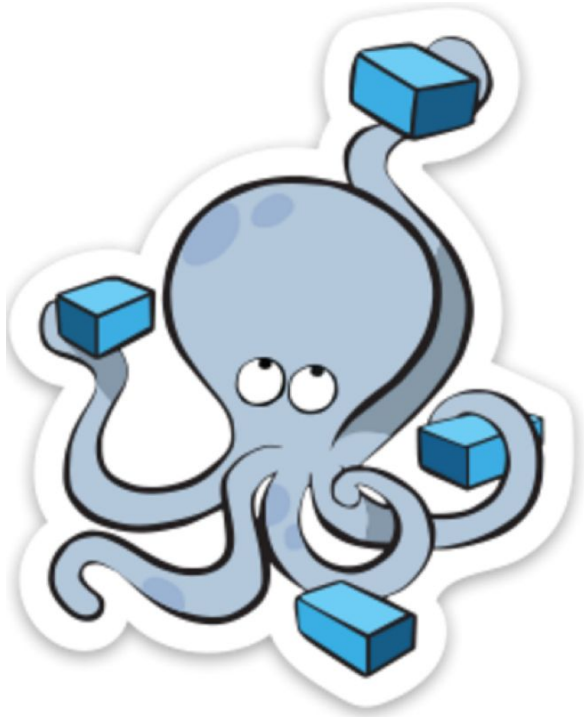
- Slow
- Doesn't Scale
- Error Prone
- Not what Docker was intended to do

Automated

- Fast
- Scales
- Repeatable
- Aligned with Docker's design intentions

Docker Compose

- Automates the building of applications (services, networks, and volumes) defined in a single file
- Each deployment is the same as the last
  - Compose files can be versioned along with the application source code



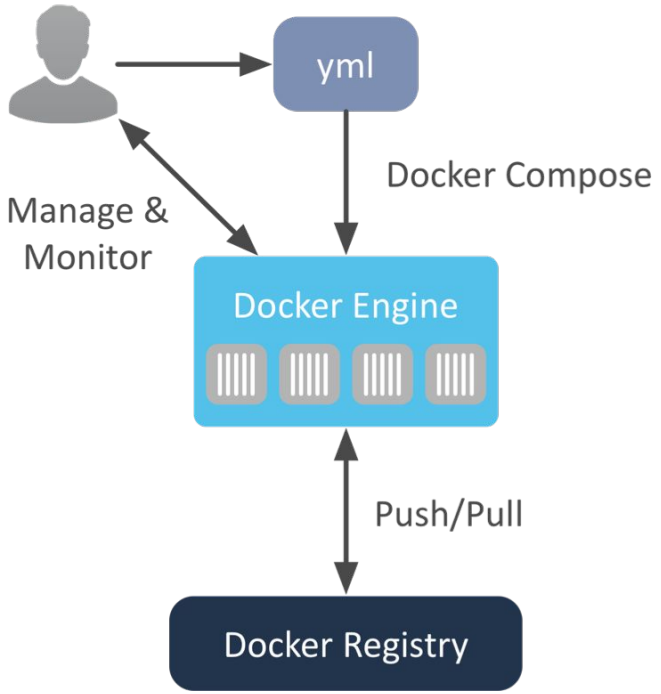
Docker Compose

Tool to create and manage multi-container applications

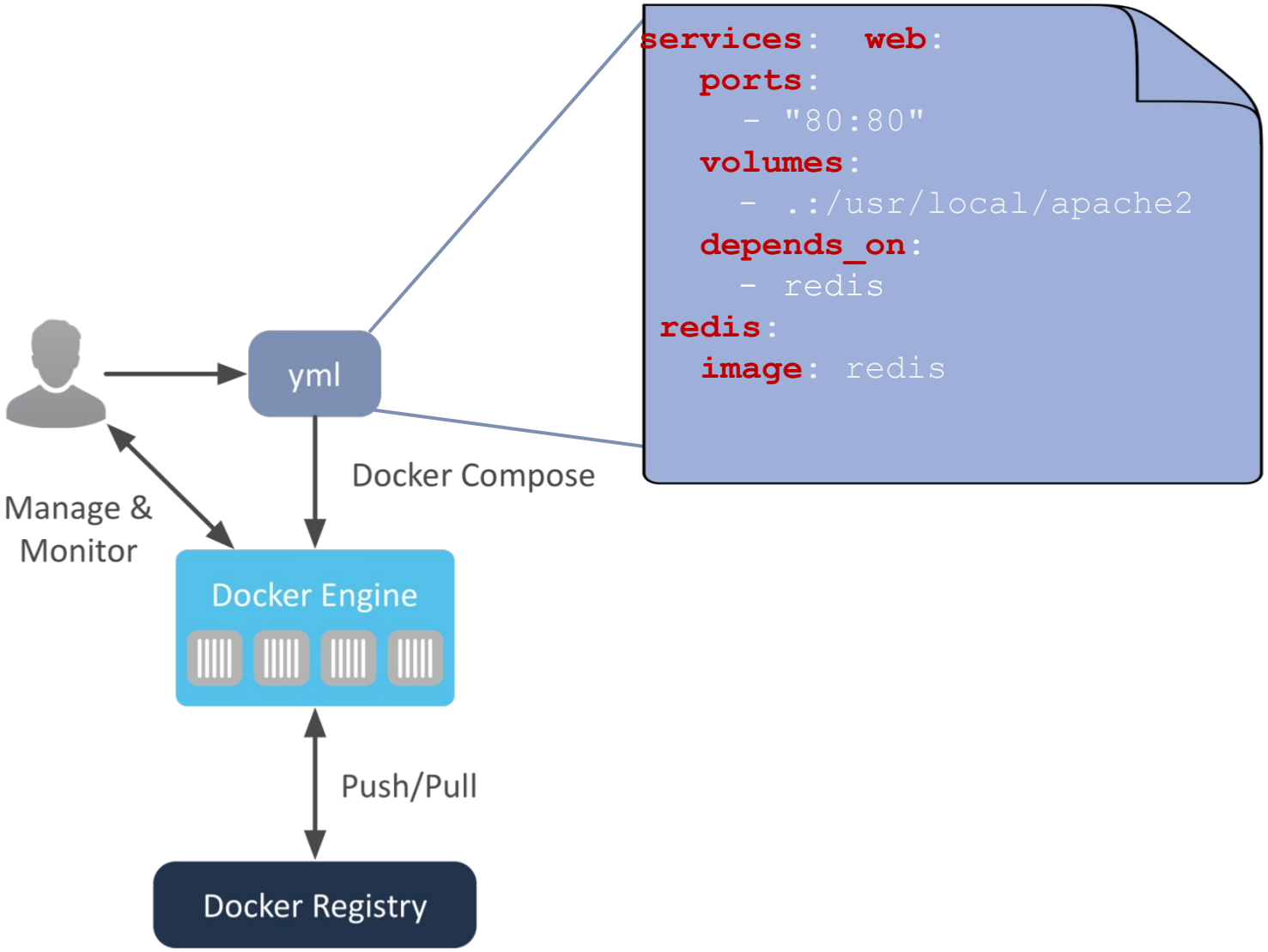
- Applications (services, networks, and volumes) defined in a single file:

  docker-compose.yml

- Transforms applications into individual containers that are linked together
- Compose will start all containers in a single command



Docker Compose



What is a Docker Composition

*A Docker Composition is a **YAML file** where all the top level keys are the names of a service and the values are the service definition.*

As with **`docker run`**, options specified in the Dockerfile are respected by default and do not need to be specified again in the **docker-compose.yml**

- **Default name and location for a compose file is:**

./docker-compose.yml

Example docker-compose.yml v2

```yaml
version: '2'

services:

  wordpress:
    image: wordpress
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_PASSWORD: example

  mysql:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: example
```

https://docs.docker.com/compose/compose-file/

Example docker-compose.yml v3

```yaml
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

https://docs.docker.com/compose/compose-file/

Docker compose CLI

```
$ docker-compose --help
Define and run multi-container applications with
Docker.  Usage:
  docker-compose [-f <arg>...] [options] [COMMAND]
  [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
  -p, --project-name  NAME    Specify an alternate project name
                              (default: directory name)
--verbose                     Show more output
-v, --version                 Print version and exit
-H, --host HOST               Daemon socket to connect to
```

Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
- Recreate only changed containers
- Variable substitution

- Default project name is the basename of the project directory
- Set a custom project name by using the `-p` command line option or the `COMPOSE_PROJECT_NAME` environment variable

Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
  - Finds any containers from previous runs
  - Copies the volumes from the old containers to the new containers
- Recreate only changed containers
- Variable substitution

Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
- Recreate only changed containers
  - Caches the configuration used to create a container
- Variable substitution

Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
- Recreate only changed containers
- Variable substitution

- Supports variables in the Compose file
- Can pass environment variables at runtime

Docker Compose Use Cases

Use Cases

- **Multiple development environments**
- Continuous Integration environments
- Shared Host environments
- Automated testing

Create multiple copies of a single environment

Example:

- Copy for
  - The stable branch
  - The current branch
  - Release candidate branch

Use Cases

- Multiple development environments
- **Continuous Integration environments**
- Shared Host environments
- Automated testing

On a shared build environment server

- Can set the project name prefix to a unique build number
- Keeps builds from interfering with each other

Use Cases

- Multiple development environments
- Continuous Integration environments
- **Shared Host environments**
- Automated testing

Allows projects that may use the same service names from interfering with each other

Use Cases

- Multiple development environments
- Continuous Integration environments
- Shared Host environments
- **Automated testing**

Run automated test scripts against a repeatable environment

Docker Compositions

Compose sections

- **version**
  - Specify file syntax version
- **services**
  - Name for container(s)
- **networks**
  - Create & configure networking for applications
- **volumes**
  - Mount a linked path on host into container

Compose versions

Compose version 1

- Docker engine versions up to 1.9.1+
- Compose up to version 1.6.x
- Bridge network, no overlay support
- Must “link” containers
- No named volume
- Only works with older versions of Swarm (not built-in)
- No “deploy” section

Compose version 1 file

```
vote:
    image: docker/example-voting-app-vote:latest
    environment:
      - "constraint:node==swarm-agent-753F9D8C000000"
    ports:
      - "5000:80"
    links:
      - redis

redis:
    image: redis:alpine
    environment:
      - "constraint:node==swarm-agent-753F9D8C000000"
    ports: ["6379"]

worker:
    image: docker/example-voting-app-worker:latest
    environment:
      - "constraint:node==swarm-agent-753F9D8C000000"
    links:
      - redis
      - db
```

Compose version 1 file

```
db:
    image: postgres:9.4
    ports: ["5432"]
    environment:
      - "constraint:node==swarm-agent-753F9D8C000000"

result:
    image: tmadams333/example-voting-app-result:latest
    ports:
      - "5001:80"
    environment:
      - "constraint:node==swarm-agent-753F9D8C000000"
    links:
      - db
```

Compose version 2

- Docker engine versions up to 1.10+
- Compose up to version 1.6+
- Overlay support
  - Containers communicate using networks
- Named volume support
- Meant for non-swarm deploys
- No “deploy” section (not applicable)

Compose version 2 file

```yaml
version: "2"

services:
  vote:
    image: docker/example-voting-app-vote:latest
    labels:
     - "com.example.description=Vote"
    ports:
      - "5000:80"
    networks:
      - front-tier
      - back-tier

  redis:
    image: redis:alpine
    ports: ["6379"]
    networks:
      - back-tier

  worker:
    image: docker/example-voting-app-worker:latest
    networks:
      - back-tier
```

Compose version 2 file

```yaml
  db:
    image: postgres:9.4
    ports: ["5432"]
    labels:
     - "com.example.description=Postgres Database"
    networks:
      - back-tier

  result:
    image: tmadams333/example-voting-app-result:latest
    ports:
      - "5001:80"
    networks:
      - front-tier
      - back-tier

networks:
  front-tier:
  back-tier:
```

Compose version 3

- Docker engine versions 1.13+
- Compose version 1.10+
- Overlay support
  - Containers communicate using networks
- Named volume support
- Designed for use with Swarm
- Deploy section for configuring Swarm options

Compose version 3 file

```yaml
version: "3"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
```

Compose version 3 file

```yaml
vote:
  image: dockersamples/examplevotingapp_vote:before
  ports:
    - 5000:80
  networks:
    - frontend
  depends_on:
    - redis
  deploy:
    replicas: 2
    update_config:
      parallelism: 2
    restart_policy:
      condition: on-failure
result:
  image: dockersamples/examplevotingapp_result:before
  ports:
    - 5001:80
  networks:
    - backend
  depends_on:
    - db
  deploy:
    replicas: 1
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure
```

Compose version 3 file

```yaml
result:
  image: dockersamples/examplevotingapp_result:before
  ports:
    - 5001:80
  networks:
    - backend
  depends_on:
    - db
  deploy:
    replicas: 1
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure
```

Compose version 3 file

```
  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]

networks:
  frontend:
  backend:

volumes:
  db-data:
```

Compose services

Compose Service Definition

This is top leve key, this is the Service Definition

Each defined service must specify exactly one **image**

Other keys are optional and analogous to their docker run command-line counter parts

```
wordpress:
  image: wordpress:latest
  ports:
    - "443:443"
  environment:
    - ADMIN_PASS=xxx
mysql:
  image:mysql  volumes:
    - mysql-data:/var/lib/mysql
  ports:
    - "3306:3306"
  links:
    - wordpress
```

<continued>

Compose Service Definition

ports: Publish a container's port(s) to the host

volumes: Bind mount a volume

environment: Set environment variables

devices: Add a host device to the container

dns: set custom DNS server

docker run --help for more information

```
mariadb:
  image: mariadb:latest
  ports:
    - "3306:3306"
  volumes:
    - wp-data:/var/wp-data
  environment:
    - DB_ADMIN_PASS=xxx
  devices:
    - "/dev/ttyUSB0:/dev/ttyUSB0"
  dns:
    - 8.8.8.8
    - 9.9.9.9
```

Linking multiple containers

Linking multiple containers

Service: Wordpress

links:
Add link to another container. Provides a secure channel via which Docker containers can communicate with one another.

In this example, the link provides secure communication between the Wordpress plug-in and Wordpress application.

```
wordpress:
  image: wordpress:latest
  ports:
    - "443:443"
  environment:
    - ADMIN_PASS=xxx
mysql:
  image: mysql
  volumes:
    - mysql-data:/var/lib/mysql
  ports:
    - "3306:3306"
  links:
    - wordpress
```

Simple Compose example

Simple Compose example

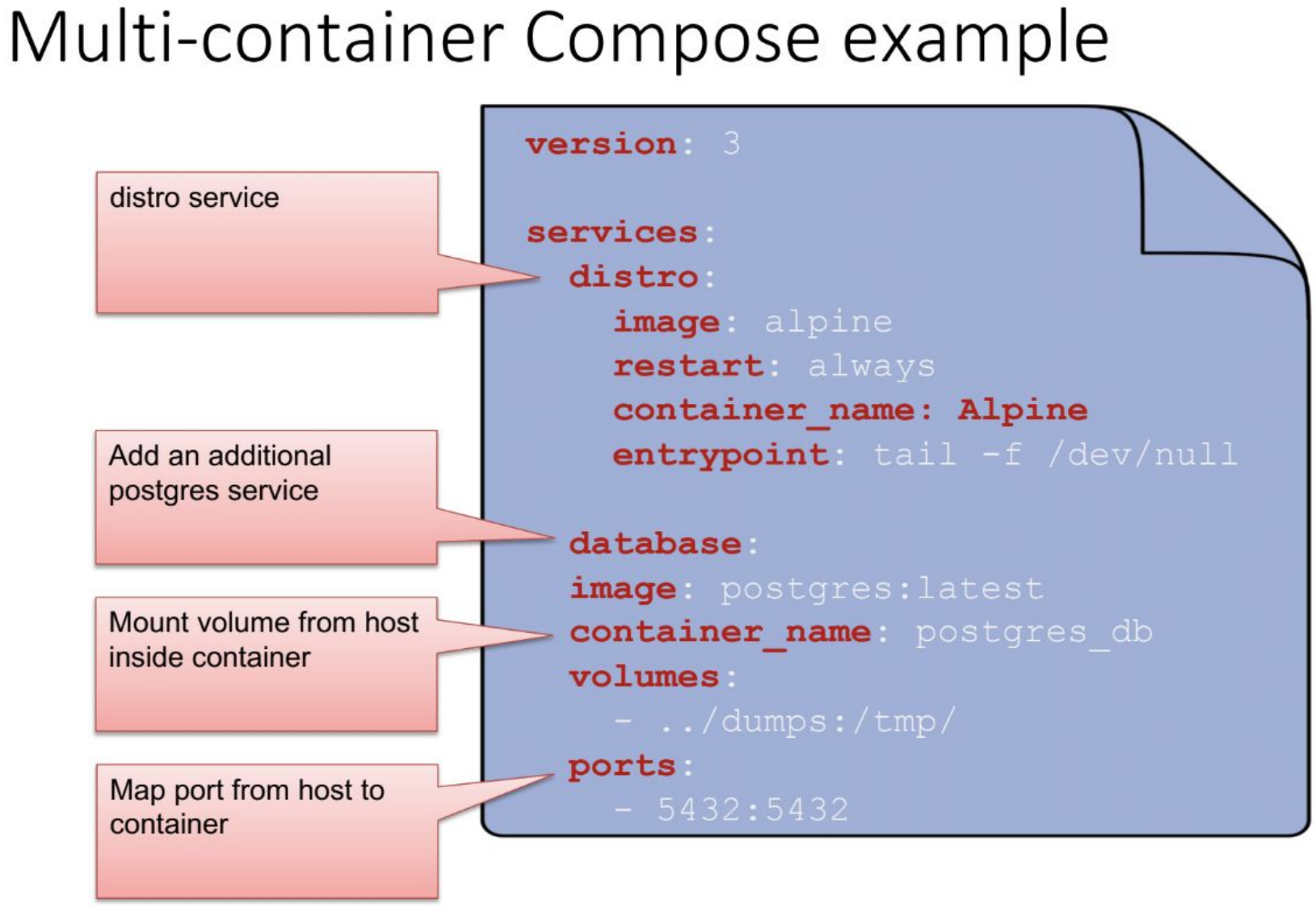
version of composition syntax

Each defined service must specify exactly one **image**

```
version: 3

services:
  distro:
    image: alpine
    restart: always
    container_name: Alpine
    entrypoint: tail -f /dev/null
```

Multi-container Compose example



Compose: build

build: Wordpress

environment:

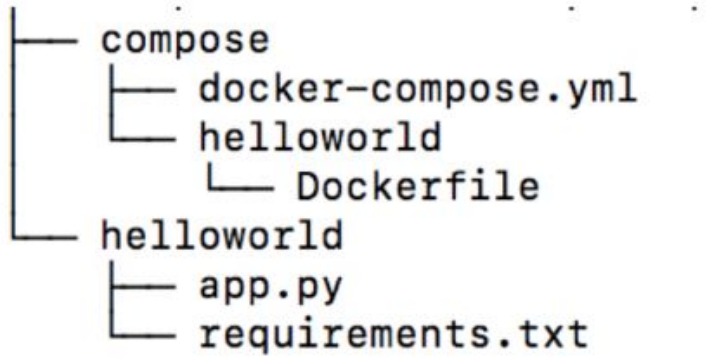
Define environment variables set in container's runtime environment

```
wordpress:
  build: ../wordpress
  ports:
    - "443:443"
  environment:
    - ADMIN_PASS=xxx
```

The build key is very useful for developers! It builds an image from source code every time compose is run. This speeds up development and avoids manually rebuilding images each time a change is made.

Compose: build

Build image from source files.



Compose file

```yaml
version: '2'
services:
  helloworld:
    build: ./helloworld
    image: helloworld:1.0
    ports:
    - "5000:5000"
    volumes:
    - ../helloworld:/code
```

Container Environment Variables

Container Environment Variables

Service: Wordpress

environment:

Define environment variables set in container's runtime environment

```
wordpress:
  image: wordpress:latest
  ports:
    - "443:443"
  environment:
    - ADMIN_PASS=xxx
```

To see what environment variables are available to a service, run:

```
# $ docker-compose run SERVICE env
```

Using the wordpress service as an example, the following slide displays the output of the command:

Show Available Environment Variables

```
# $ docker-compose run wordpress env

# HOSTNAME=f836bf313aed
# TERM=xterm
# CA_CERTIFICATES_JAVA_VERSION=20140324
#
  PATH=/usr/share/wordpress/bin:/usr/local/sbin:/usr/
  local/bin:/usr/sbin:/usr/bin:/sbin:/bin
# PWD=/
# JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
# LANG=C.UTF-8
# JAVA_VERSION=8u66
# SHLVL=0
# HOME=/root
# JAVA_DEBIAN_VERSION=8u66-b17-1~bpo8+1
```

Environment Variable Substitution

- You are able to pass variable values from the shell environment to docker-compose
- The environment variable POSTGRES_VERSION=9.4 might be set within the shell. You could then supply the following configuration:

```
db:
  image: "postgres:${POSTGRES_VERSION}"
```

Compose Deploy section

- Ignored by docker-compose commands
- Specify configuration related to deployment and services.
- Constrain services to specific nodes
- Deploy to Docker Swarm (docker stack deploy)

Compose Deploy example

```yaml
deploy:
  mode: replicated
  replicas: 5
  update_config:
    parallelism: 2
    delay: 10s
  restart_policy:
    condition: on-failure
```

Compose Deploy: mode

- Global
  - One container per node
  - Similar to DaemonSets in Kubernetes
- Replicated (Default)
  - Specified number of containers

Compose Deploy: replicas

- Number of containers that should be running at all times.
- Docker creates/deletes containers to keep at specified number

Compose Deploy: placement

- Specify which nodes containers should run on

```
placement:
  constraints:
    - node.role == manager
    - engine.labels.operatingsystem == ubuntu 16.04
```

Compose Deploy: update_config

- Configure how the service should be updated
  - Rolling updates
- parallelism
  - Containers to update at a time
- delay
  - Time to wait between updating group of containers
- failure_action
  - If update fails either “continue” or pause (Default)

Compose Deploy: restart_policy

- Configure restart of containers when they exit
  - condition: none, on-failure, any (Default any)
  - delay: How long to wait between restarts (Default 0)
  - max_attempts: How many times to restart before giving up (Default never)
  - window: How long to wait before deciding if restart was successful (Default immediately)

Compose Deploy: resource limits

- Set resource limits on containers
  - MEM
  - CPU

- Only works on containers running on Swarm.
- For individual containers use --compatibility

Compose Deploy: labels

- Two different options
  - Apply label to service
  - Apply label to containers

Service

```yaml
version: "3"
services:
  web:
    image: web
    deploy:
      labels:
        com.example.description: "This label will appear on the web service"
```

Containers

```yaml
version: "3"
services:
  web:
    image: web
    labels:
      com.example.description: "This label will appear on all containers for the web service"
```

Docker Compose CLI

Running in Detached Mode

Running **`docker-compose`** with **`-d runs the`** Docker containers in the background

```
# $ docker-compose up
# 02multiple_web_1 is up-to-date
# Attaching to web_1
# web_1 | => Configuring NGINX ...
# web_1 | => ... Starting Services
# Gracefully stopping... (press
 Ctrl+C  again to force)
# Stopping 02multiple_web_1 ... done
# $ docker-compose up -d
# Starting 02multiple_web_1
# $
```

Displaying logs for docker-compose

`docker-compose logs` displays log output from services

```
# $ docker-compose logs
# Attaching to web_1
# web_1 | => Configuring NGINX ...
# web_1 | => Using default NGINX
  configuration
# web_1 | => Configuring PHP-FPM ...
# web_1 | => Done!
# web_1 | => ... Starting Services
```

Scaling docker-compose containers

`docker-compose scale [SERVICE=NUM]`
sets the number of containers for a service

```
# $ docker-compose scale web=3
# Creating and starting 2 ... done
# Creating and starting 3 ... done
# $ docker-compose scale web=1
 # Stopping    02multiple_web_2        ...   done
 # Stopping    02multiple_web_3        ...   done
 # Removing    02multiple_web_3        ...   done
 # Removing    02multiple_web_2        ...   done
```

Running a Selected Single Service

docker-compose run, runs a one-time command against a service

Docker-compose run overrides the commands specified in the **docker-compose** service configuration

```
# $ hostname
# docker-workstation
# $ docker-compose run web bash
# [root@9ad9eec462fd /]# hostname
# 9ad9eec462fd
# [root@8f3f8001f8df /]# exit
# exit
# $ hostname
# docker-workstation
```

Stopping a Selected Single Service

`docker-compose stop SERVICE` shuts down the service gracefully

`docker-compose kill SERVICE` immediately kills the service

```
# $ docker-compose run --rm -d
web
# Starting web_1
# $ docker-compose kill web
# Killing web 1 ... done
# Name   Command  State   Ports
# web_1  /bin/run Exit    137
# $
```

Lab03: Simple Docker Compose

- Manually build Docker image
- Manually run Docker container
- Create Compose file to automate building & deployment of application.

Docker Container Volumes

Discussion

What are some examples where Container Data Persistence might be required?

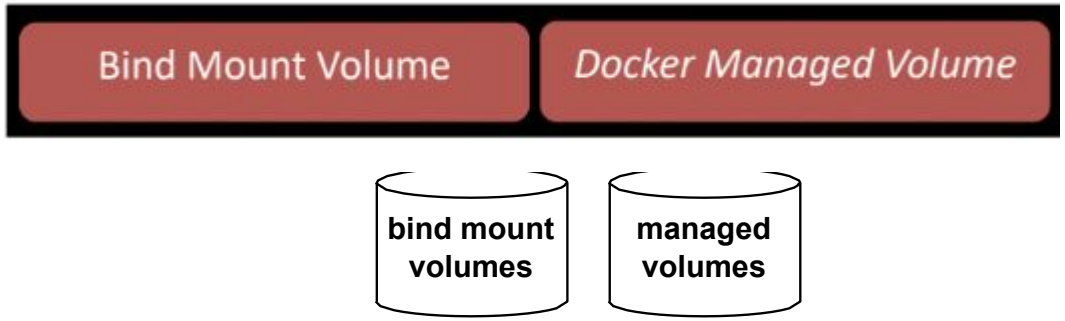
Discussion

What are some examples where Container Data Persistence might be required?

- Where Data Persistence is important:
  - Database Records (e.g. MySQL, MariaDB, Cassandra)
- Shared data from host to containers
  - build tools (e.g. Jenkins jobs)
  - automation tools (e.g. Ansible playbooks)
- Share data between containers
  - Container-A creates data, Container-B uses created data
- Access to Docker host's data
  - /var/log, /etc/hosts, /var/lib/, etc

Docker Volume Types

Volume Types



Provides Data:

- Accessibility by container and non-containers
- Persistence beyond a container's lifecycle
- Sharing between Containers
- Segmentation between Containers

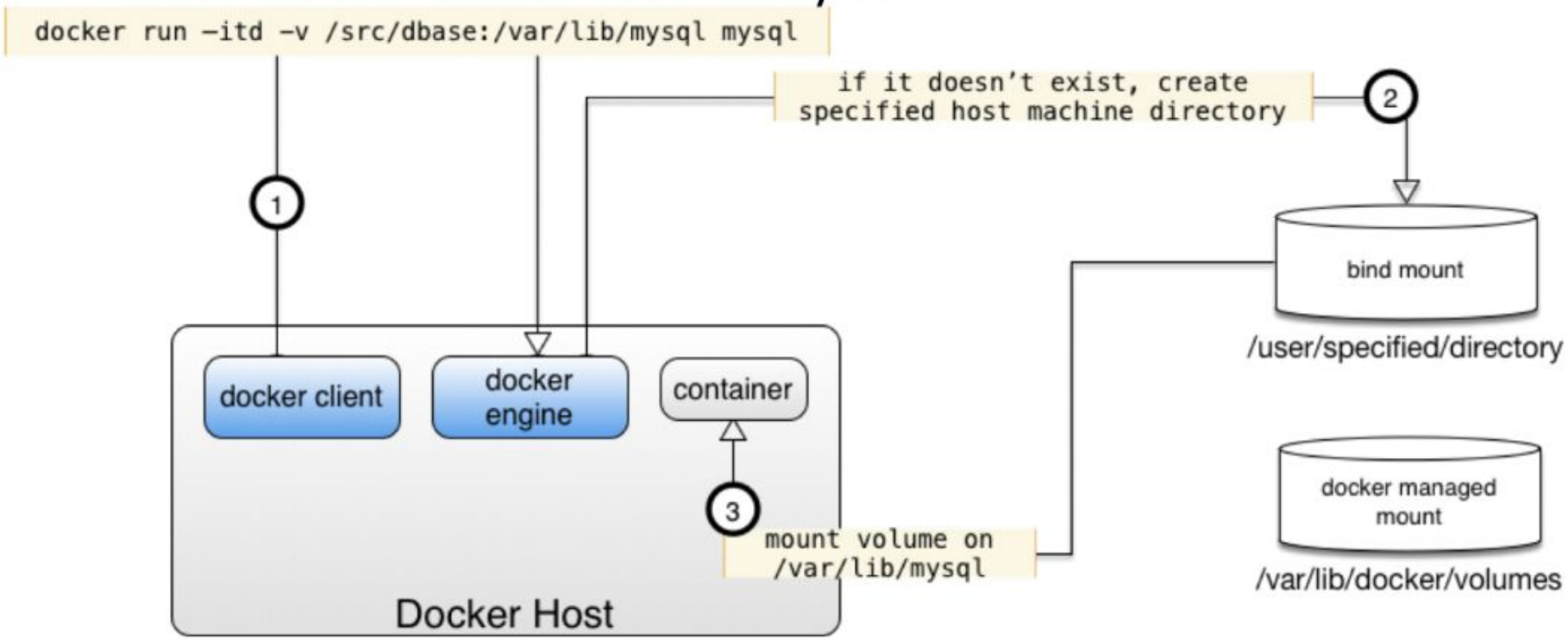
Bind-mount volumes

- User Defined
- Provides Data:
  - Accessible by containers and non-containers
  - Creates the directories, if required, in the volume path
  - Absolute paths must be used
  - Can not be automatically cleaned up by Docker-Engine
  - Managed by User

Bind-mount volumes

Bind-Mount Volume:

- user defined
- resides outside the Docker-managed space
- can be shared between many containers



Create a bind-mount volume

Create a bind-mount volume:

- Creates the /data, if it doesn't exist, mounts on container

```
$ docker run -itd -v /data:/data busybox top
```

Mounts are found in **`docker inspect`** output:

```
$ docker inspect $(docker ps -lq)  "Mounts": [
            {
                "Source": "/data",
                "Destination": "/data",  "Mode":
                "",
                  "RW": true
}
```

Compose: Create a bind-mount volume

Create a bind-mount volume:

- Creates the /data, if it doesn't exist, mounts on

```
$ docker-compose up -d
```

docker-compose file (volumes):

```
version: 2  services:
  mysql:
    image: mysql  container_name: mysql
    volumes:
      - /data:/data
}
```

Create multiple bind-mount volumes

Create a bind-mount volume:

- Creates the directory, if it doesn't exist, mounts on container

```
$ docker run -itd --name share \
 -v /peanut:/peanut -v /butter:/butter \
 -v /jelly:/jelly busybox top
```

Mounts are found in **`docker inspect`**

```
$ docker inspect $(docker ps -lq)
"Mounts": [
            {
                "Source": "/peanut",
                "Source": "/butter",
                "Source": "/jelly",
}
```

Compose: Create multiple bind-mount volumes

Create a bind-mount volume:

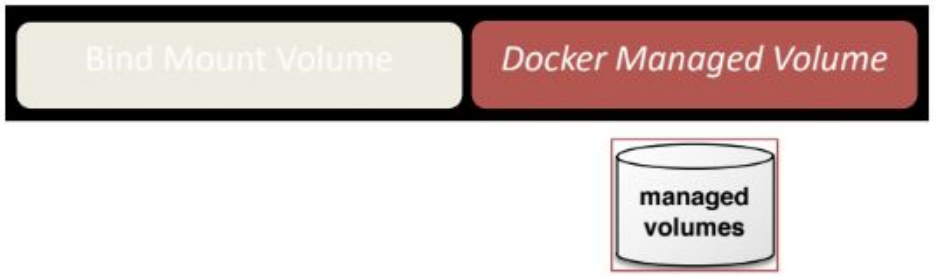
- Creates the /data, if it doesn't exist, mounts on container

```
$ docker-compose up -d
```

docker-compose file (volumes):

```
version: 2  services:
  mysql:
    image: mysql  container_name: mysql
    volumes:
      - /data:/data
      - ./config:/etc/mysql/my.cnf
}
```

Docker managed volumes



- Docker Defined Mount Point
- Provides Data:
  - Accessible by container and non-containers
  - Creates the volume in /var/lib/docker/volumes/
  - Can be automatically cleaned up by Docker-Engine
  - Can be queried for orphaned volumes
  - Managed by Docker

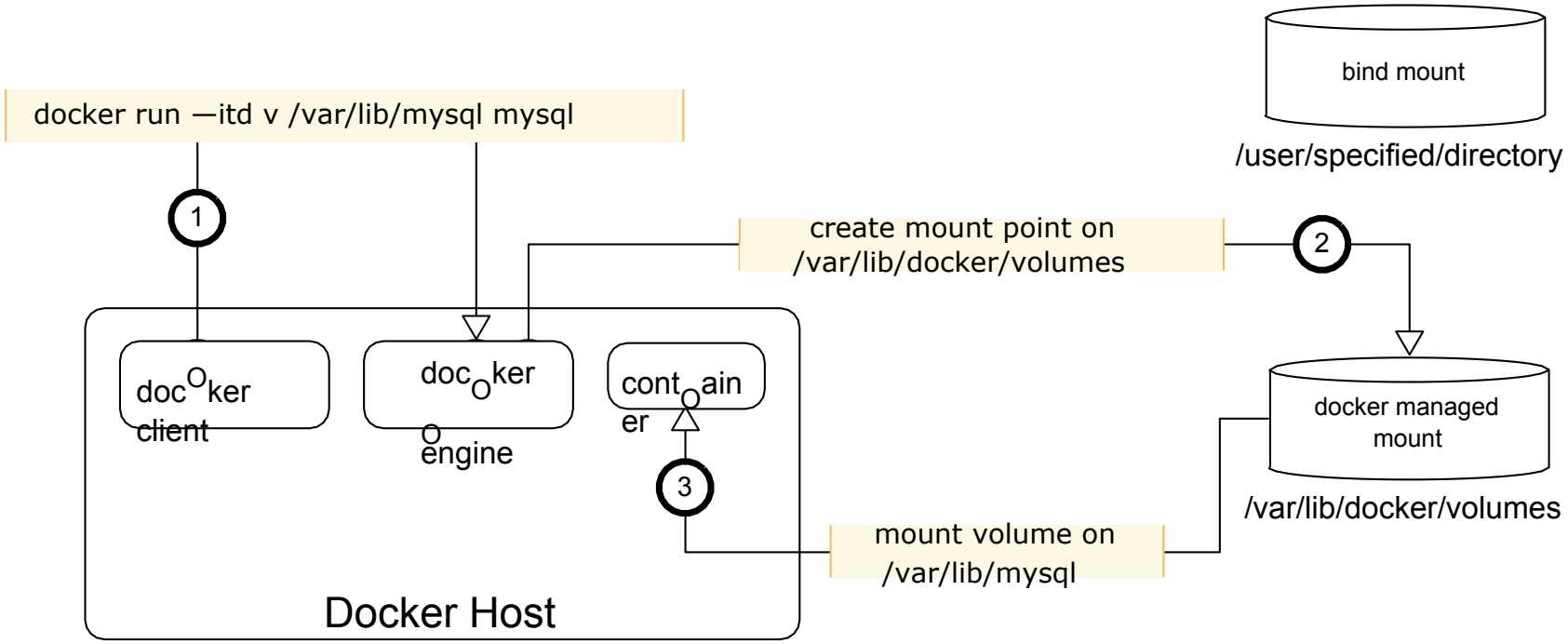
```
$ docker run -itd -v data:/data busybox top
```

```
$ docker run -itd -v /data busybox top
```

Managed Volumes

Managed Volume:

- defined by docker-engine
- resides inside the Docker-managed space
- can be shared between many containers



Create multiple named volumes

Create a named volume:

- Creates the directory, if it doesn't exist, mounts on container

```
$ docker run -itd --name share \
 -v peanut:/peanut -v butter:/butter \
 -v jelly:/jelly busybox top
```

Compose:Create multiple named volumes

Create a bind-mount volume:

- Creates the /data, if it doesn't exist, mounts on container

```
$ docker-compose up -d
```

docker-compose file (volumes):

```
version: 2  services:
  mysql:
    image: mysql  container_name: mysql
    volumes:
      - data:/data
      - mysql-config:/etc/mysql/my.cnf
}
```

Docker Volume Metadata

Volume Metadata

What is it – volume metadata describes what the volume is through the use of key value pairs

- **docker volume inspect <id>**
  - Queries the Docker Engine, a json formatted output is returned

```
$ docker volume inspect <volume>
```

```
        "Name":  "Driver":  "Mountpoint":  "Labels":
         "Scope":
```

Questions

Lab04: Docker Compose evolution

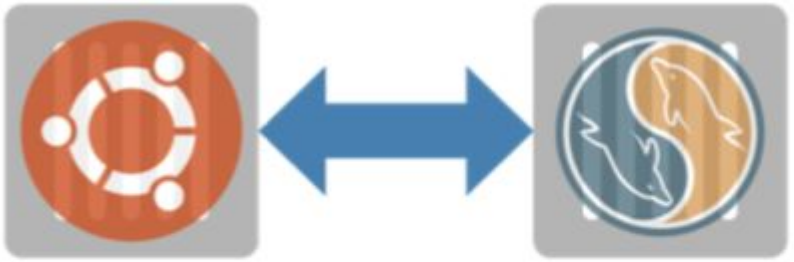
- Build simple compose file
- Add additional services to compose file
- Add links
- Add volumes

Docker Networking

Docker Networking Overview

Docker Engine provides network access for containers that need:

- Other Containers



Docker Networking Overview

Docker Engine provides network access for containers that need:

- Other Containers
- Networks

Docker Networking Overview

Docker Engine provides network access for containers that need:

- Other Containers
- Networks

Accomplished through:

1. IP Address Management (IPAM)
2. Service Port exposure
3. Manual Port Mapping
4. Dynamic Port Mapping

Docker Networking Architecture

Network Interfaces

Docker Container

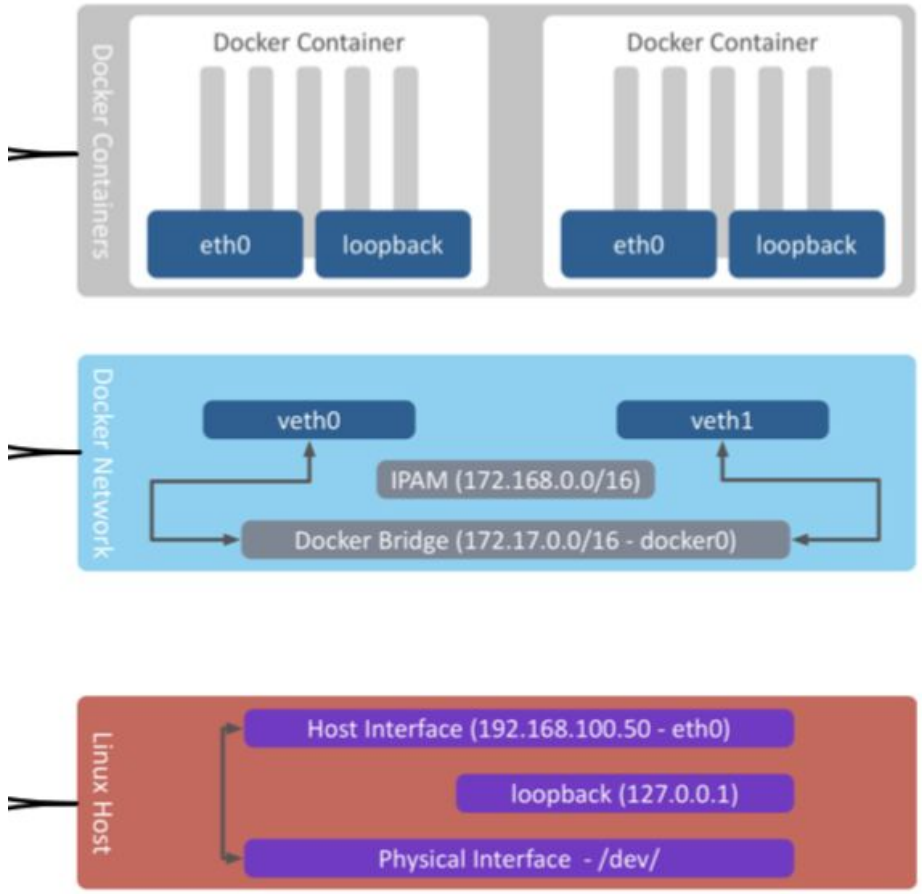
- eth0 Interface
- Loopback Interface

Docker Network

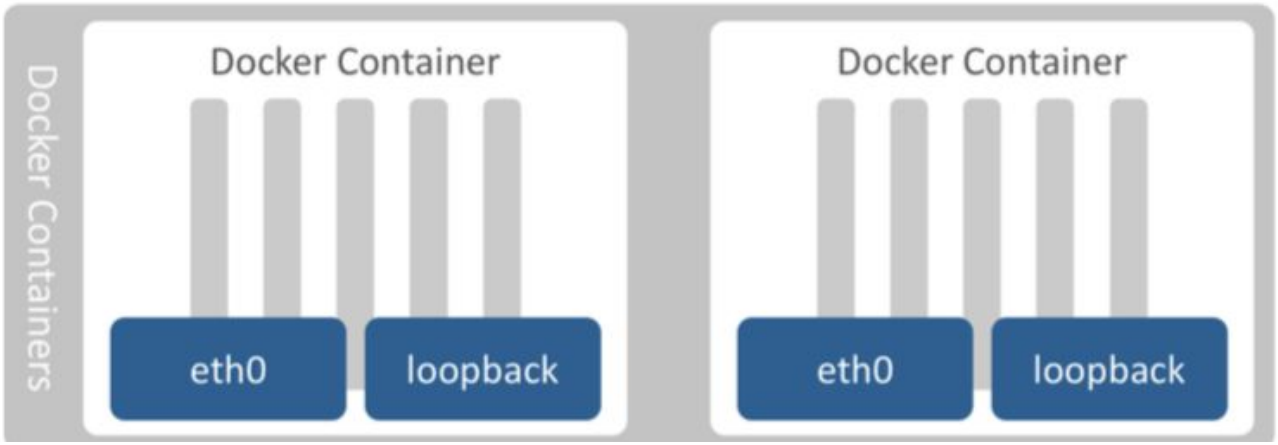
- Docker Bridge
- IP Address Management

Linux Host Network

- Host Interfaces
- Host Loopback Interface
- Physical Interfaces



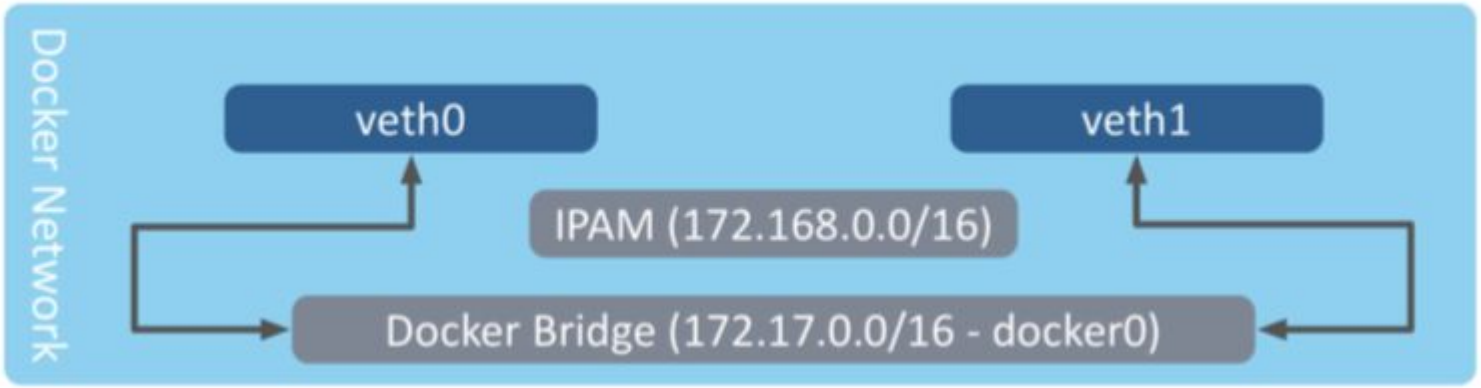
Docker Container



Docker Container

- **Loopback Interface** - used for internal communication
- **eth0 Interface** - used for external communication
- Container TCP Port Exposure

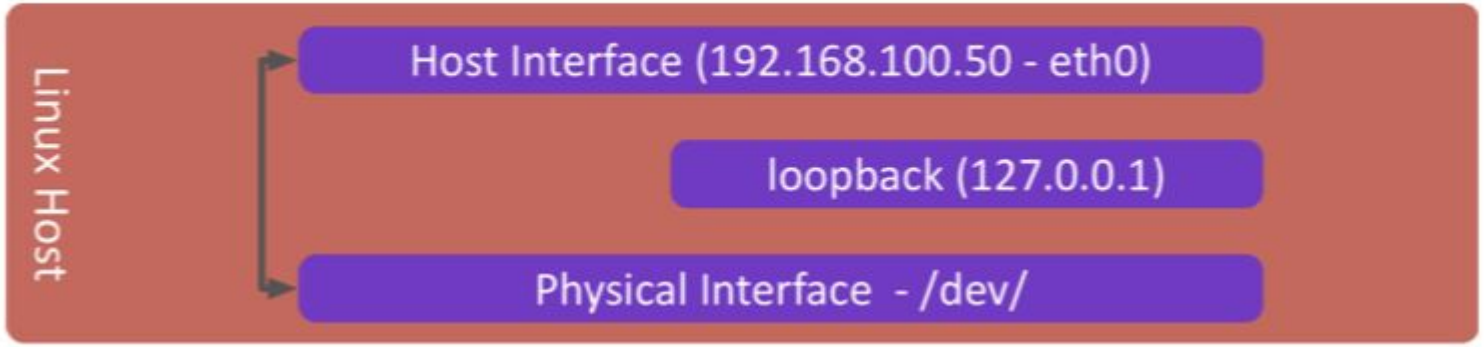
Docker Container



Docker Network

- **Docker Bridge** (e.g. docker0) - the bridge which manages traffic between Docker Engine and the Linux Host
- **veth Interfaces** - interfaces on the bridge to the containers
- **IP Address Management** - assigns IP addresses to container that require them
- Container to Host TCP Port Mapping

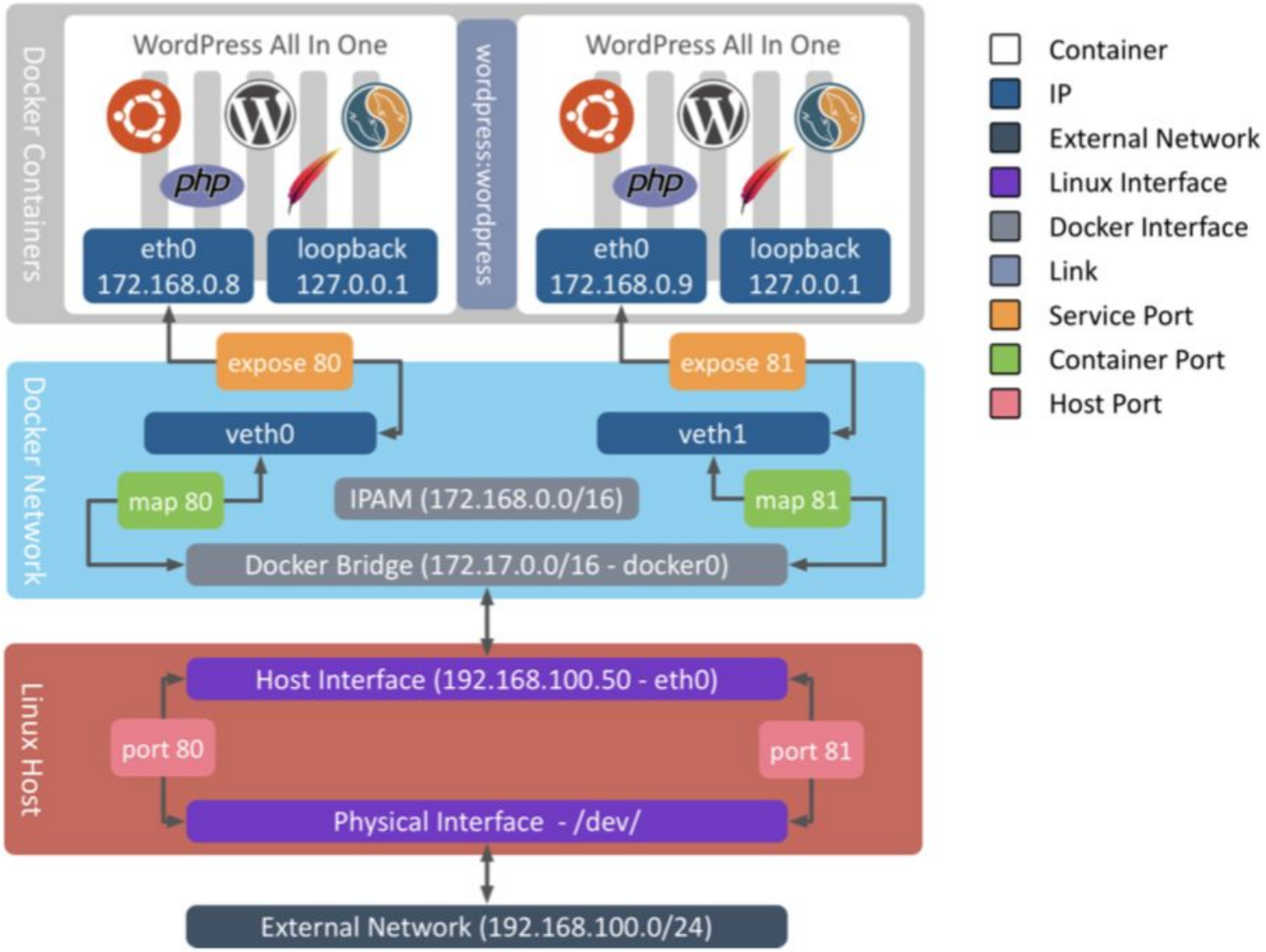
Docker Container



Linux Host Network

- **Host Loopback Interface** - host internal communication
- **Host Interfaces** - internal and external communication
- **Physical Interfaces** - sends/receives external communication to/from the outside world
- Linux Host TCP Ports

Docker Networking Architecture



Compose:Networks

Configure default network:

```
$ docker-compose up -d
```

docker-compose file (network):

```
version: 2  services:
  web:
    build: .  ports:
              - 8000:8000
  db:
    image: postgres

networks:  default:
    driver: custom-driver-1
```

Compose:Networks

Create your own networks:

```
version: 2  services:
  web:
    build: .  networks:
      front:
        ipv4_address: 172.16.238.10
  db:
    image: postgres  networks:
  front:
    driver:bridge  ipam:
      driver: default  config:
        - subnet: 172.16.238.0/24
```

Compose:Networks

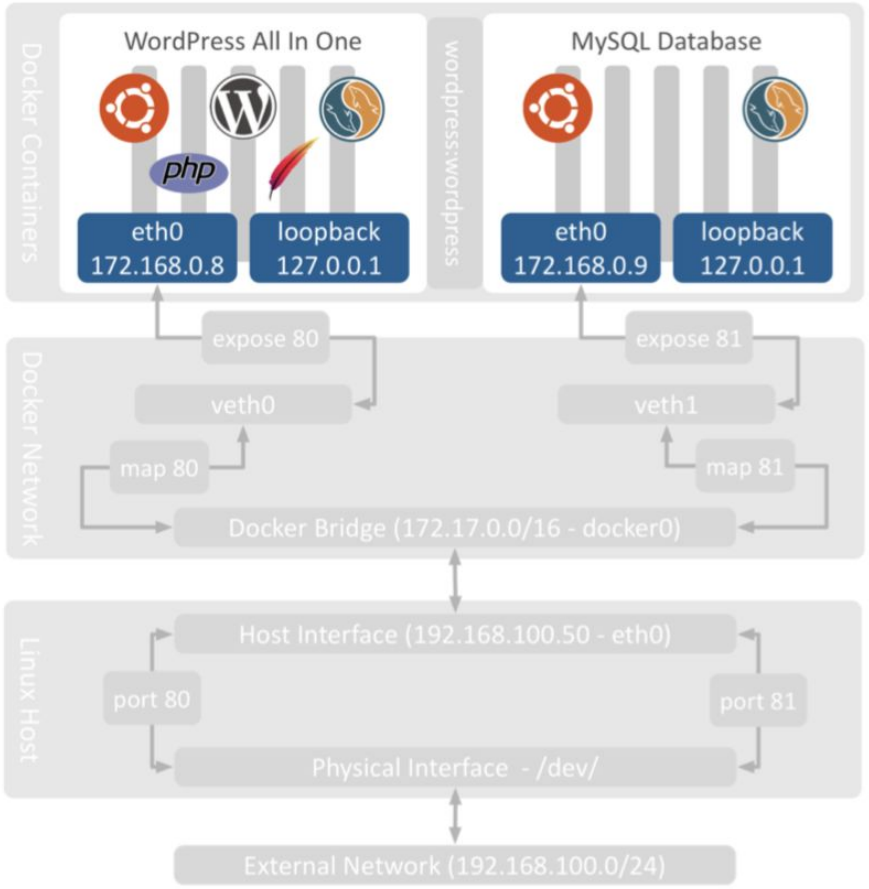
Use existing networks:

```
version: 2  networks:
  default:  external:
      name: network-already-created
```

Compose creates networks by default, if you specify an external network it does not create it, and instead connects the container to it.

Link Containers

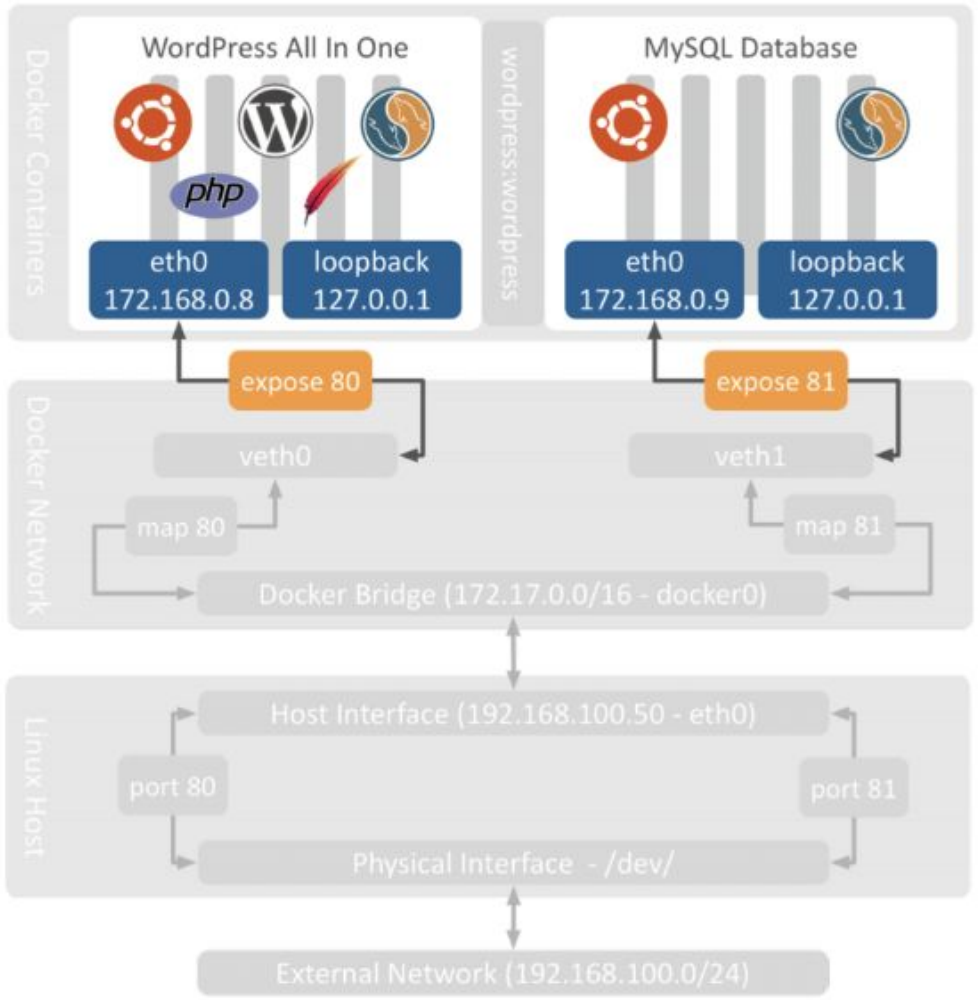
Link containers



Linking:

- Service Ports are known to all containers
- Ports are exposed from the container
- Host ports are mapped to container ports for accessibility
- Linking containers create internal routing rules
- Linking containers uses aliases to connect containers

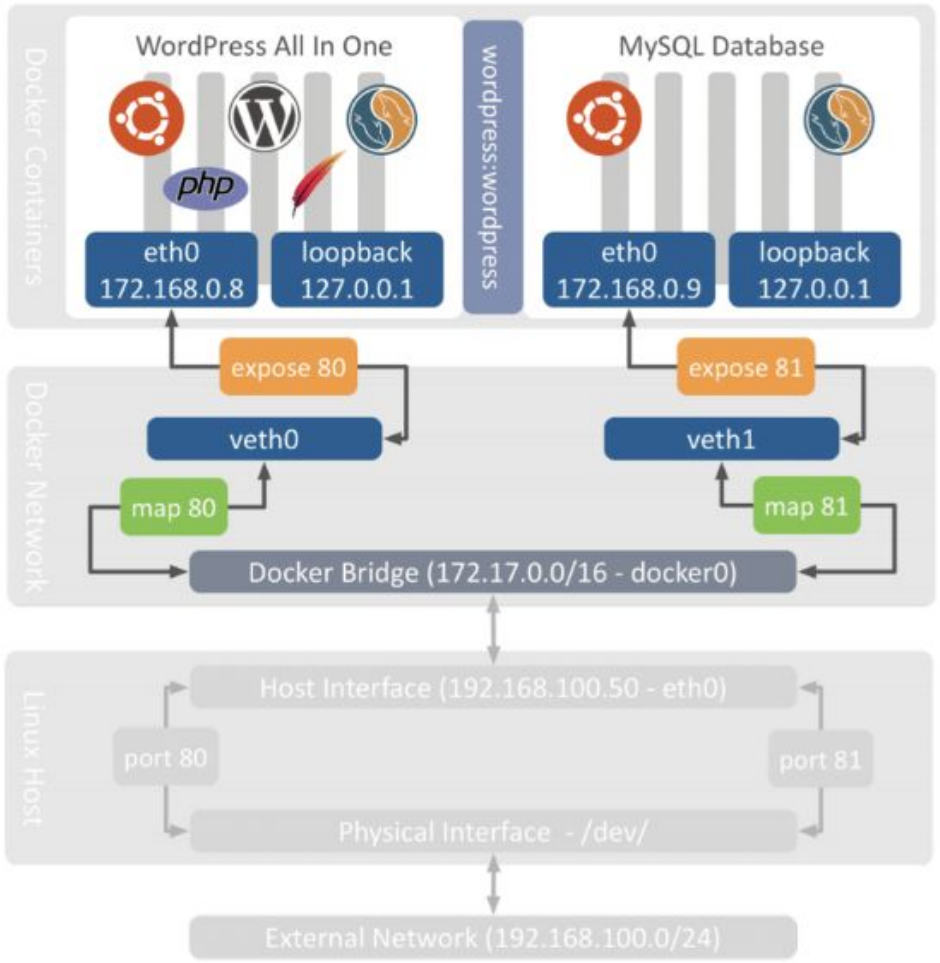
Link containers



Linking:

- Service Ports are known to all containers
- Ports are exposed from the container
- Host ports are mapped to container ports for accessibility
- Linking containers create internal routing rules
- Linking containers uses aliases to connect containers

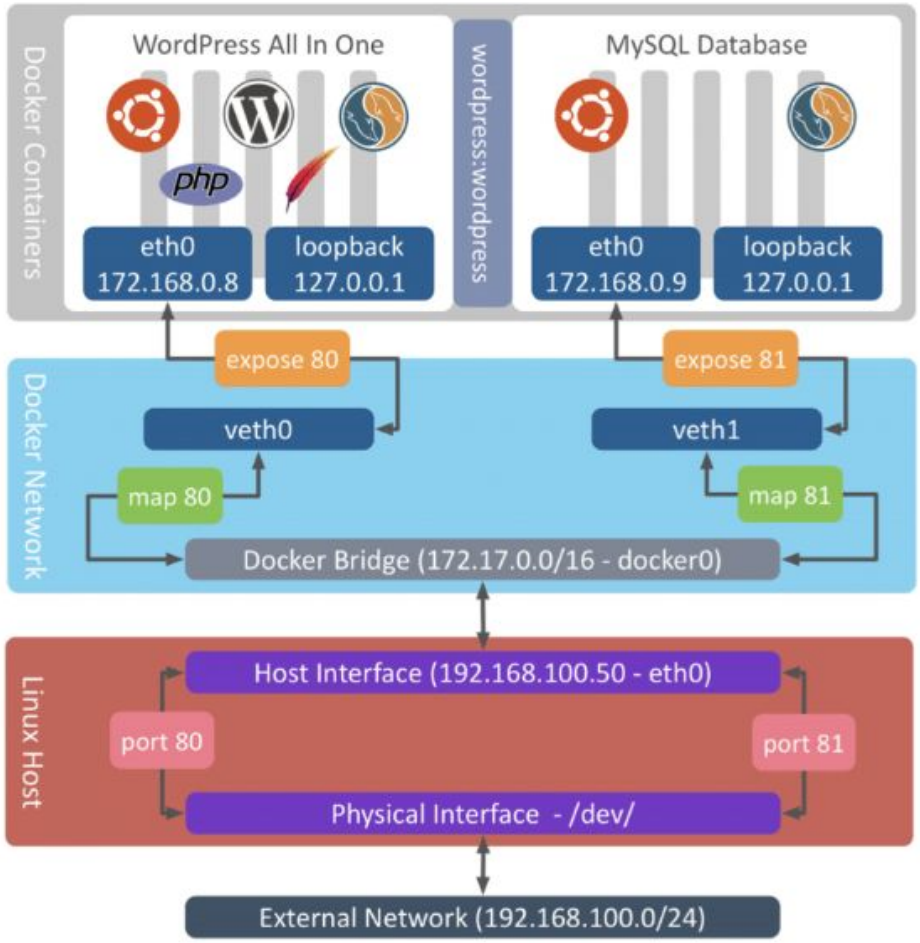
Link containers



Linking:

- Service Ports are known to all containers
- Ports are exposed from the container
- Host ports are mapped to container ports for accessibility
- Linking containers create internal routing rules
- Linking containers uses aliases to connect containers

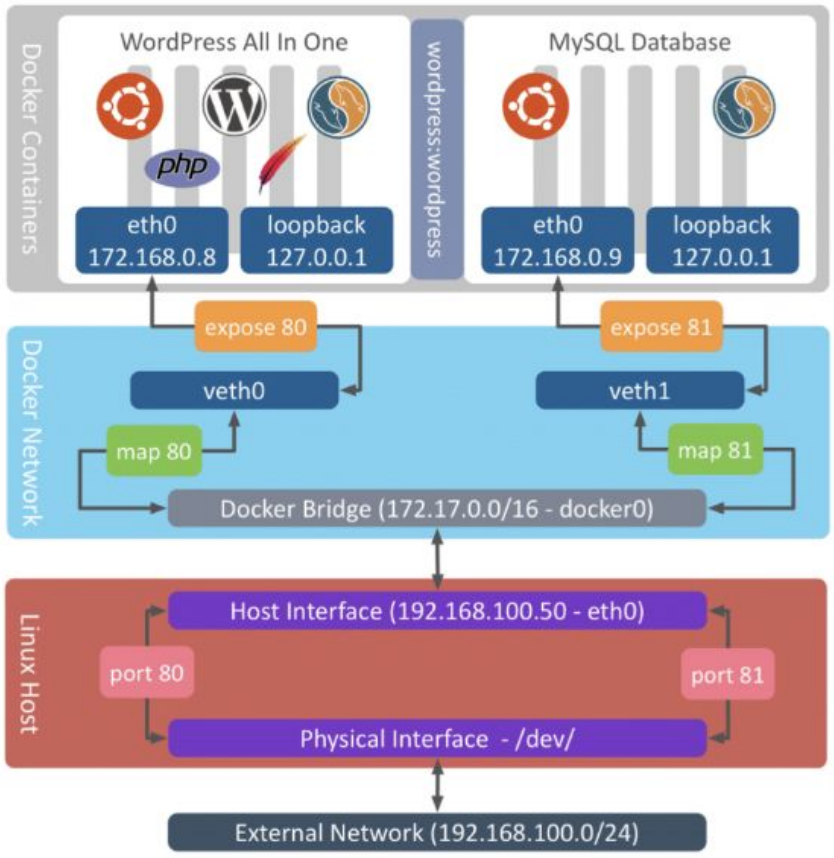
Link containers



Linking:

- Service Ports are known to all containers
- Ports are exposed from the container
- Host ports are mapped to container ports for accessibility
- Linking containers create internal routing rules
- Linking containers uses aliases to connect containers

Link containers



Linking:

- Service Ports are known to all containers
- Ports are exposed from the container
- Host ports are mapped to container ports for accessibility
- Linking containers create internal routing rules
- Linking containers uses aliases to connect containers

Compose:Linking

Create a link between web and db:

```
$ docker-compose up -d
```

docker-compose file (links):

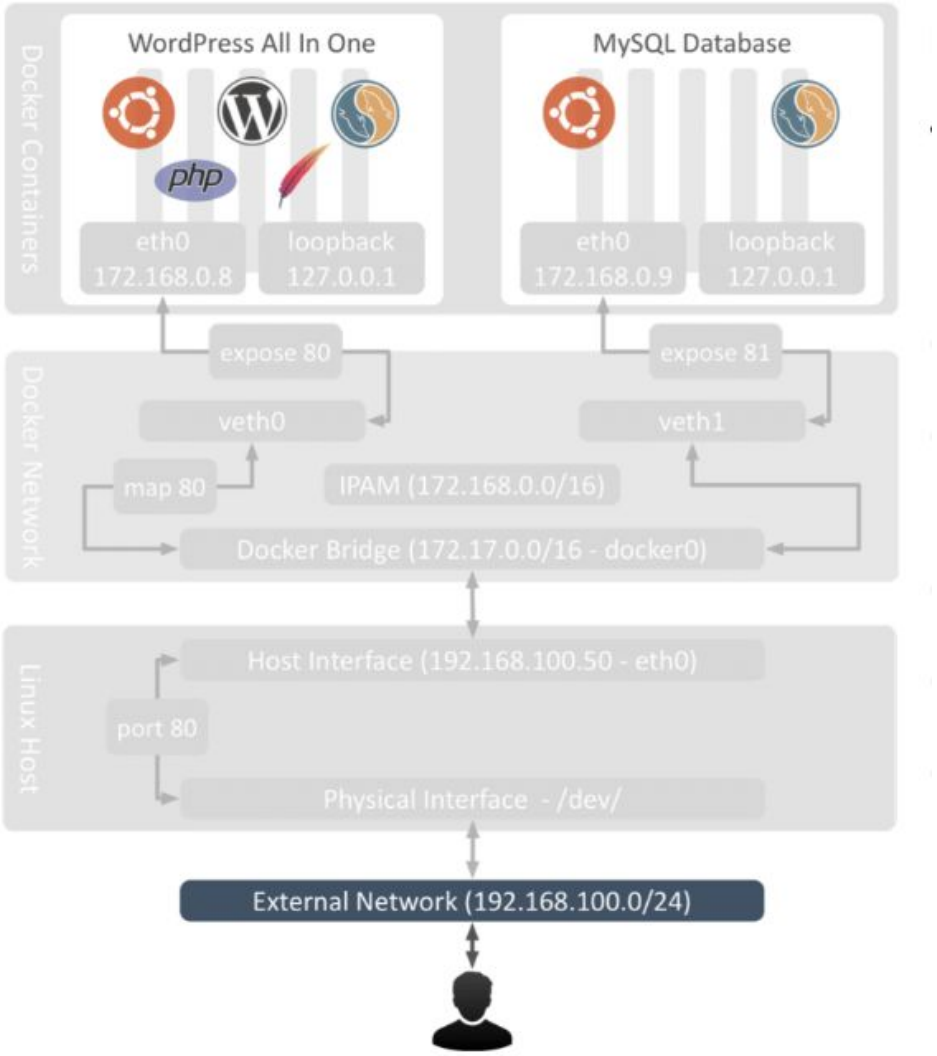
```
version: 2  services:
  web:
    build: .  links:
      - "db:database"
  db:
    image: postgres
```

User Traffic Flow

User Traffic Flow

- **Applications provide:**
  - Service Ports that need to be exposed
- **Docker Engine provides:**
  - Exposure of required Service Ports
  - Mapping of required Service Ports
- **Linux Host provides:**
  - Internal connectivity between containers
  - Internal connectivity between host and containers
  - External connectivity between containers and users

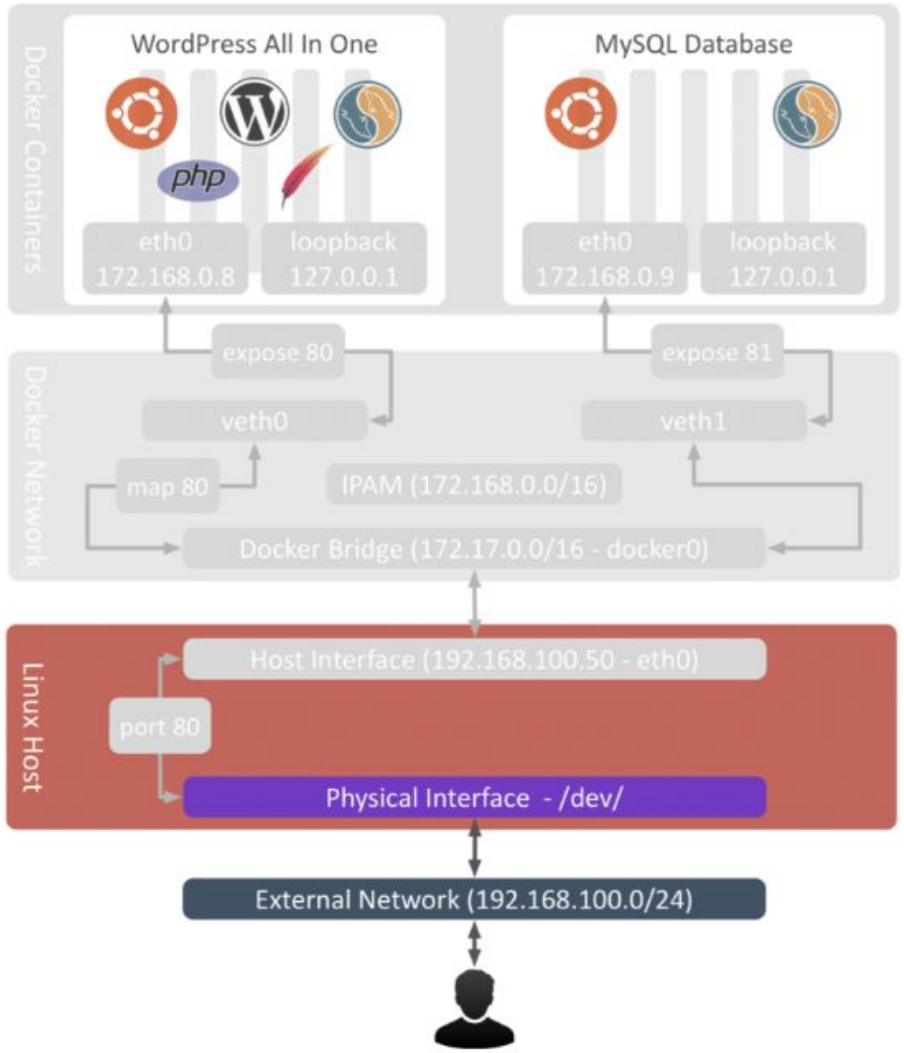
User Traffic Flow



User Traffic:

- Enters from the external network (outside the Docker host)
- Enters the physical interface
- Is passed to the host interface (eth0)
- Is passed to the Docker Bridge
- Is passed to the veth interface
- Is passed to eth0 in the container

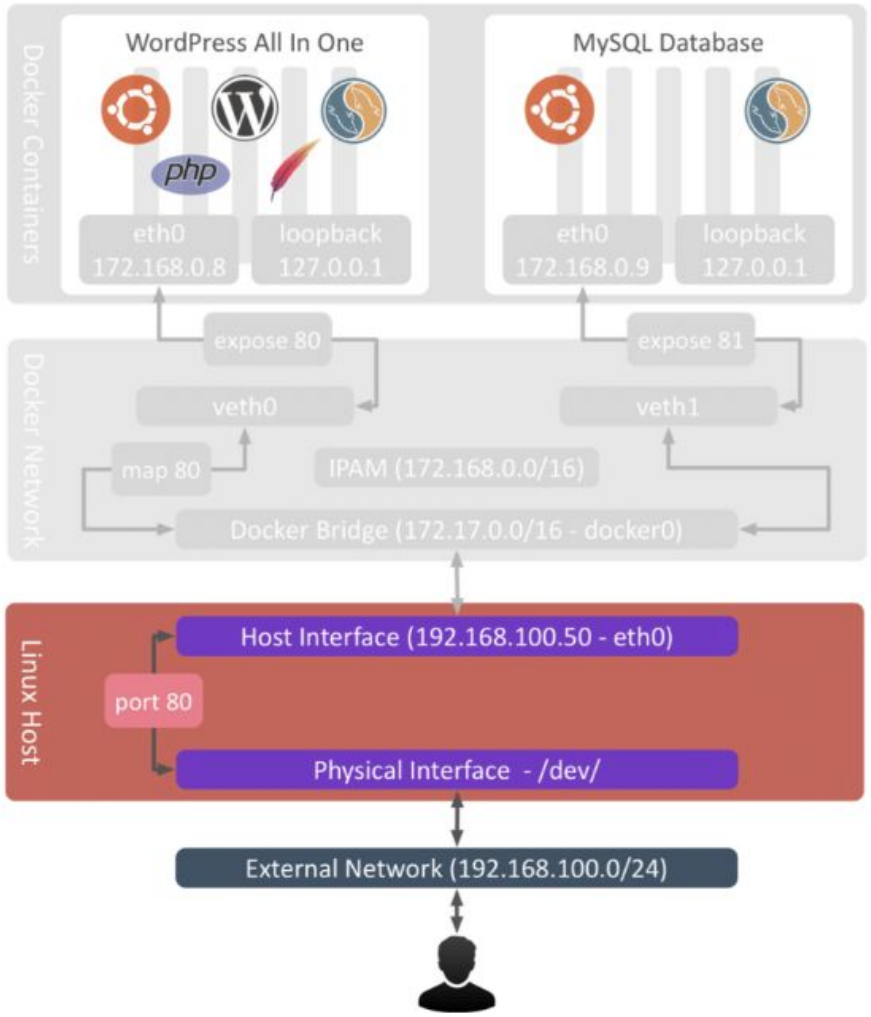
User Traffic Flow



User Traffic:

- Enters from the external network (outside the Docker host)
- Enters the physical interface
- Is passed to the host interface (eth0)
- Is passed to the Docker Bridge
- Is passed to the veth interface
- Is passed to eth0 in the container

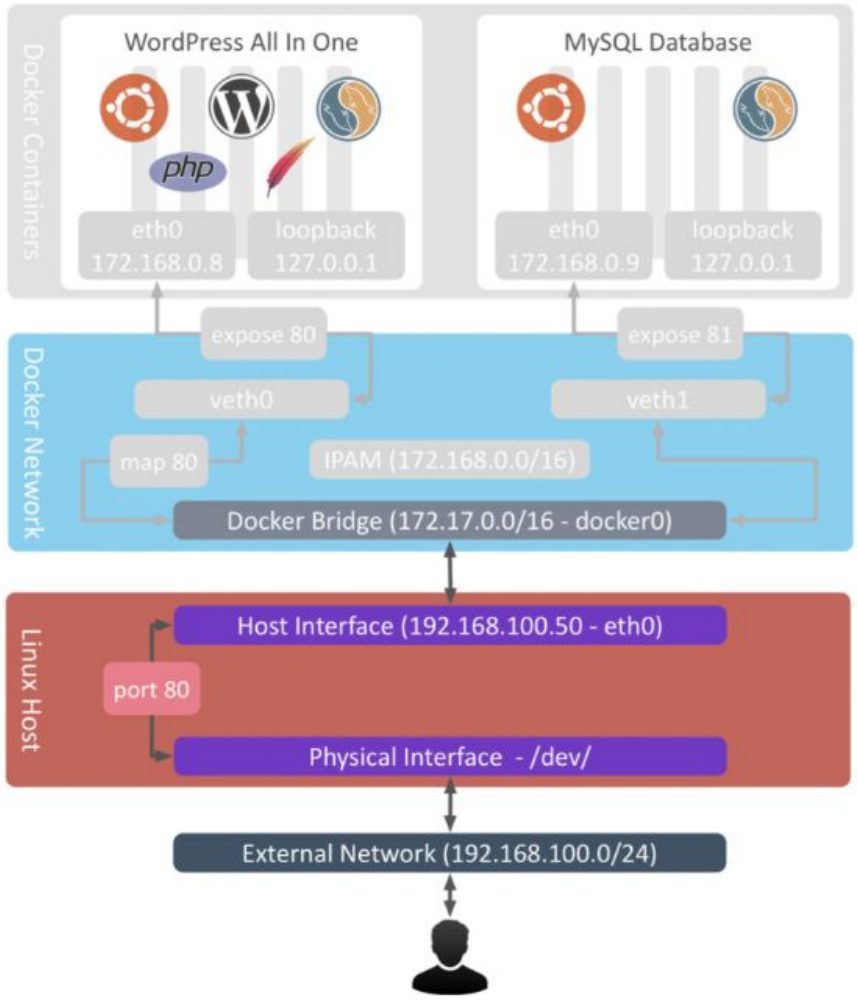
User Traffic Flow



User Traffic:

- Enters from the external network (outside the Docker host)
- Enters the physical interface
- Is passed to the host interface (eth0)
- Is passed to the Docker Bridge
- Is passed to the veth interface
- Is passed to eth0 in the container

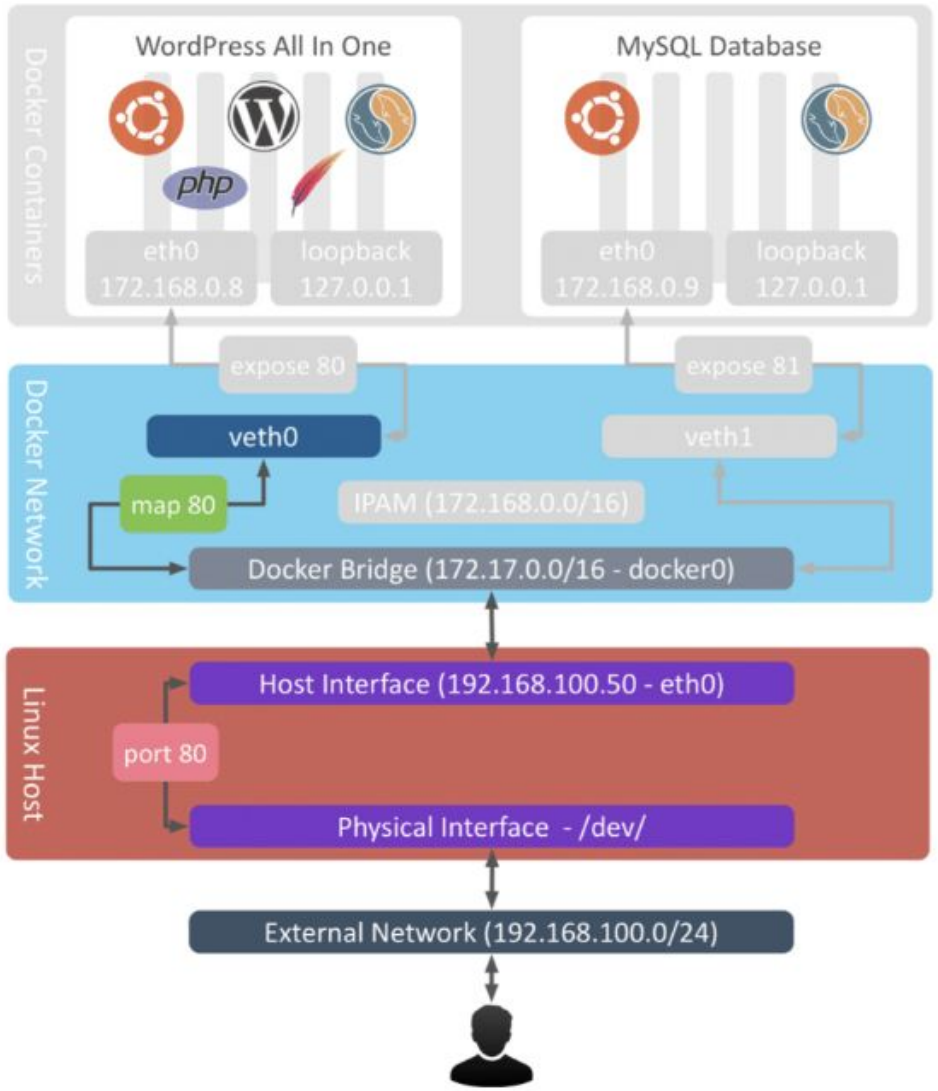
User Traffic Flow



User Traffic:

- Enters from the external network (outside the Docker host)
- Enters the physical interface
- Is passed to the host interface (eth0)
- Is passed to the Docker Bridge
- Is passed to the veth interface
- Is passed to eth0 in the container

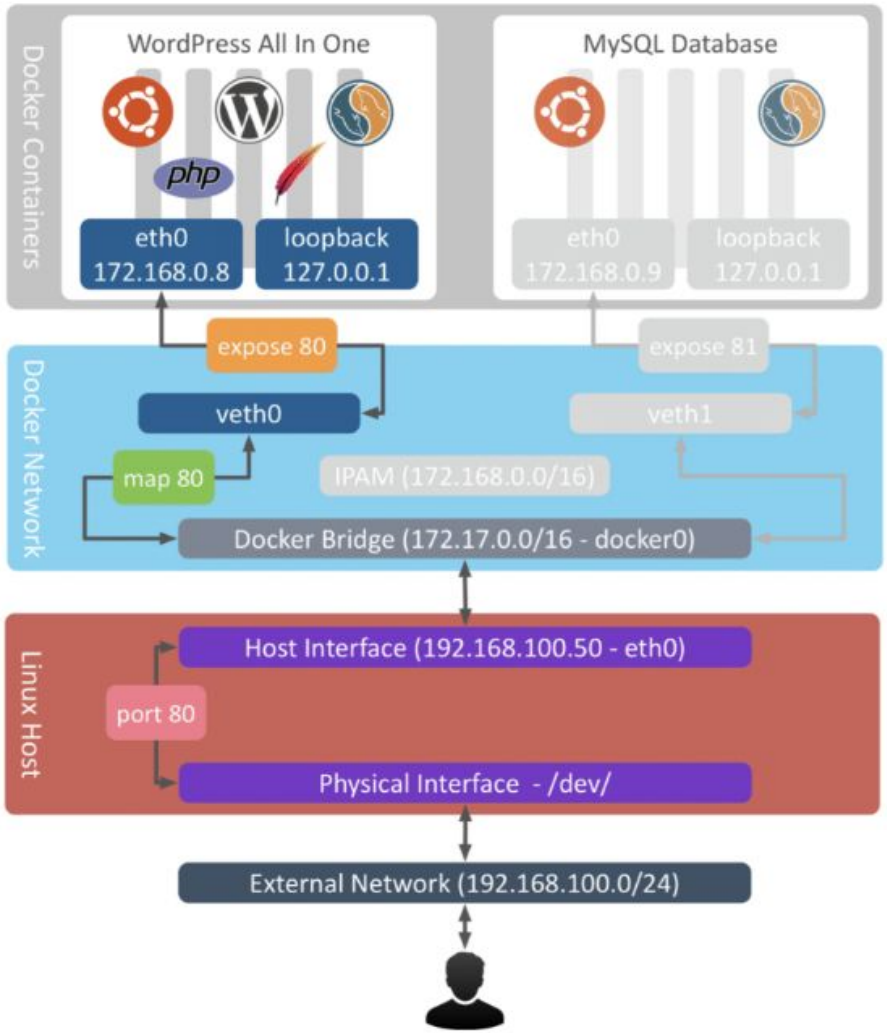
User Traffic Flow



User Traffic:

- Enters from the external network (outside the Docker host)
- Enters the physical interface
- Is passed to the host interface (eth0)
- Is passed to the Docker Bridge
- Is passed to the veth interface
- Is passed to eth0 in the container

User Traffic Flow



User Traffic:

- Enters from the external network (outside the Docker host)
- Enters the physical interface
- Is passed to the host interface (eth0)
- Is passed to the Docker Bridge
- Is passed to the veth interface
- Is passed to eth0 in the container

Lab05: Docker Networks

- Build a multi-container application, using multiple networks and volumes.
- VOTE!
- View results.

Questions

Lab06: Capstone Project

- Build your Docker Compose project, using:
  - Past Docker-Compose Labs
  - Current Open source Projects
  - Your Favorite Languages, Tools
- Time Breakdown
  - Research & Decide on Docker Compose project (20 min.)
  - Execute Docker Compose project (40 min.)
  - Share results with class (30 min.)
- Rules
  - There is no rules!


delete
thank
you