

Docker technology

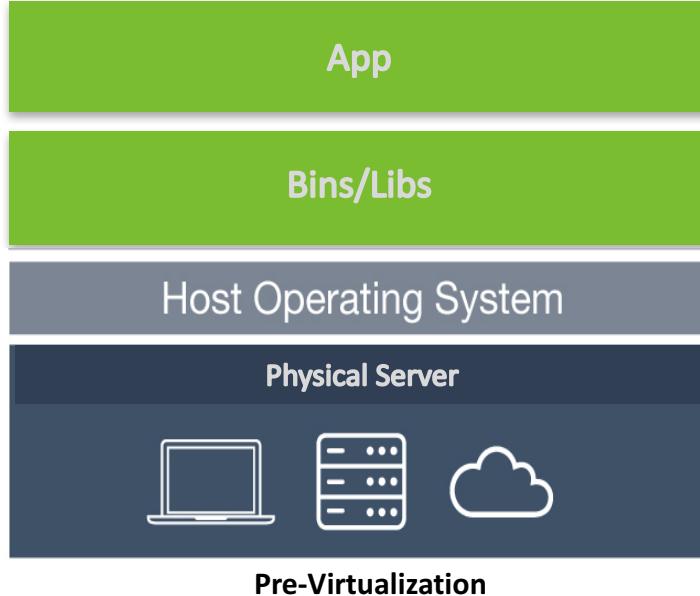
is one implementation of container based
virtualization technologies



LEVEL UP
www.level-up.one

Introduction to Virtualization Technologies

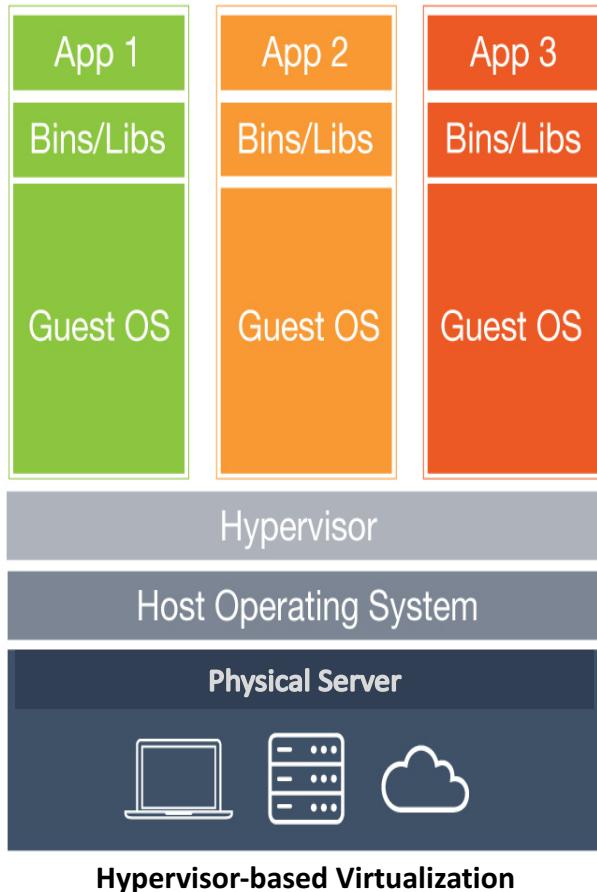
Pre-Virtualization World



Problems:

- Huge Cost
- Slow Deployment
- Hard to Migrate

Hypervisor-based Virtualization



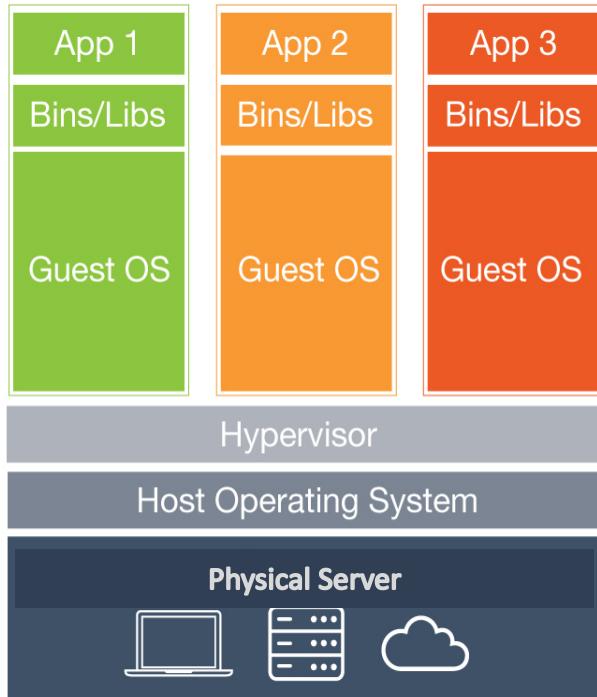
Benefits:

- Cost-Efficient
- Easy to Scale

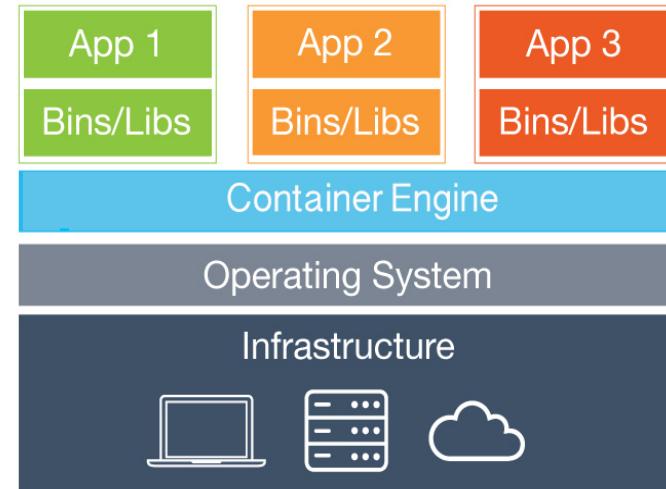
Limitations:

- Kernel Resource Duplication
- Application Portability Issue

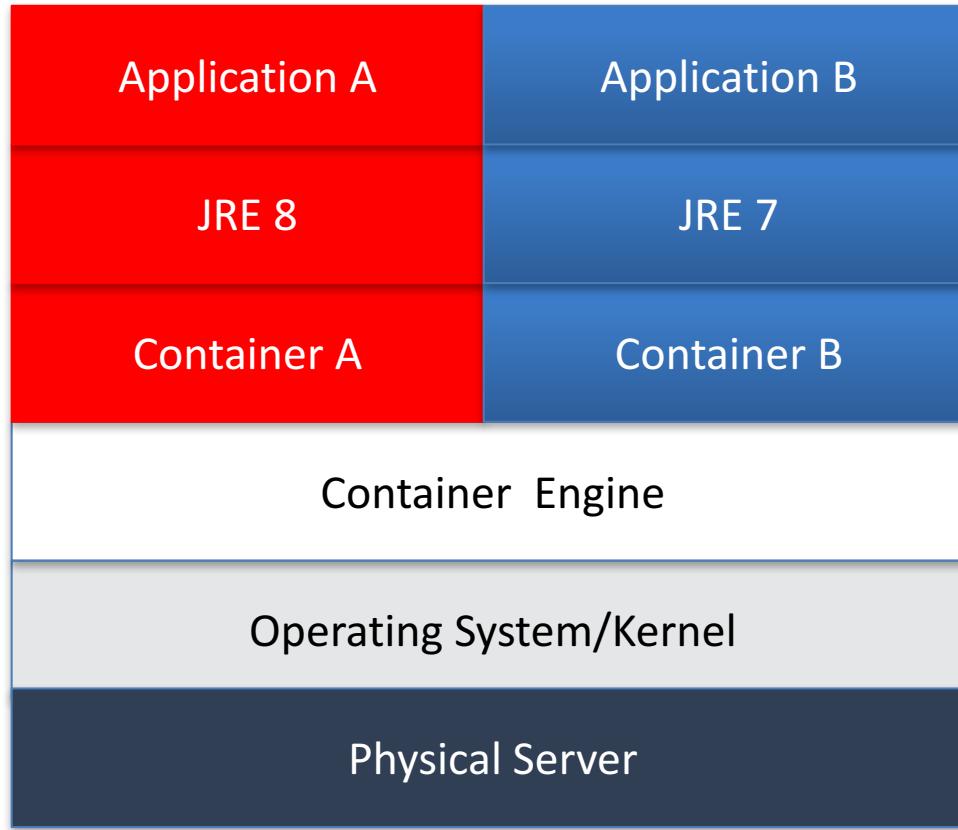
Hypervisor-based VS Container-based Virtualization



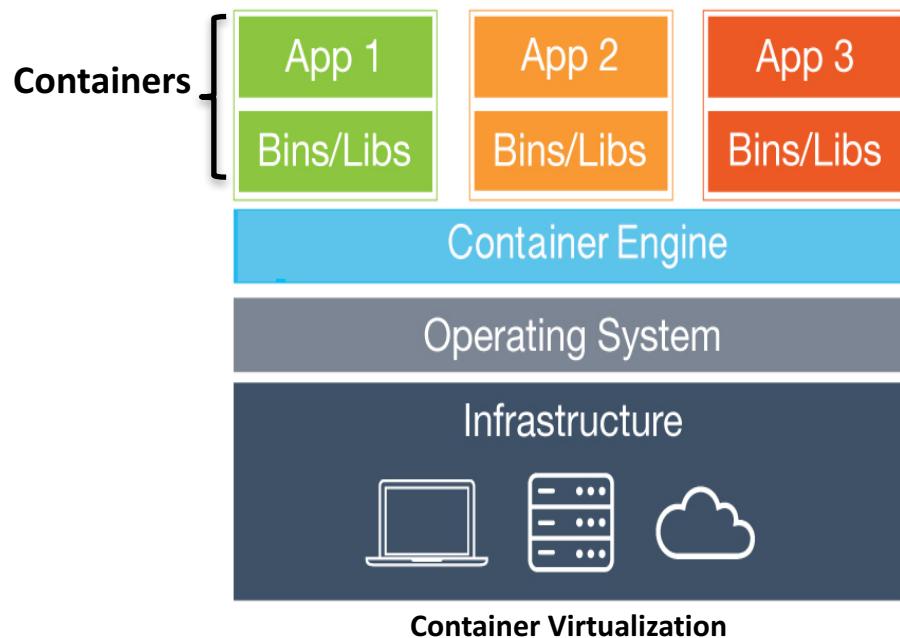
Containers [



Runtime Isolation

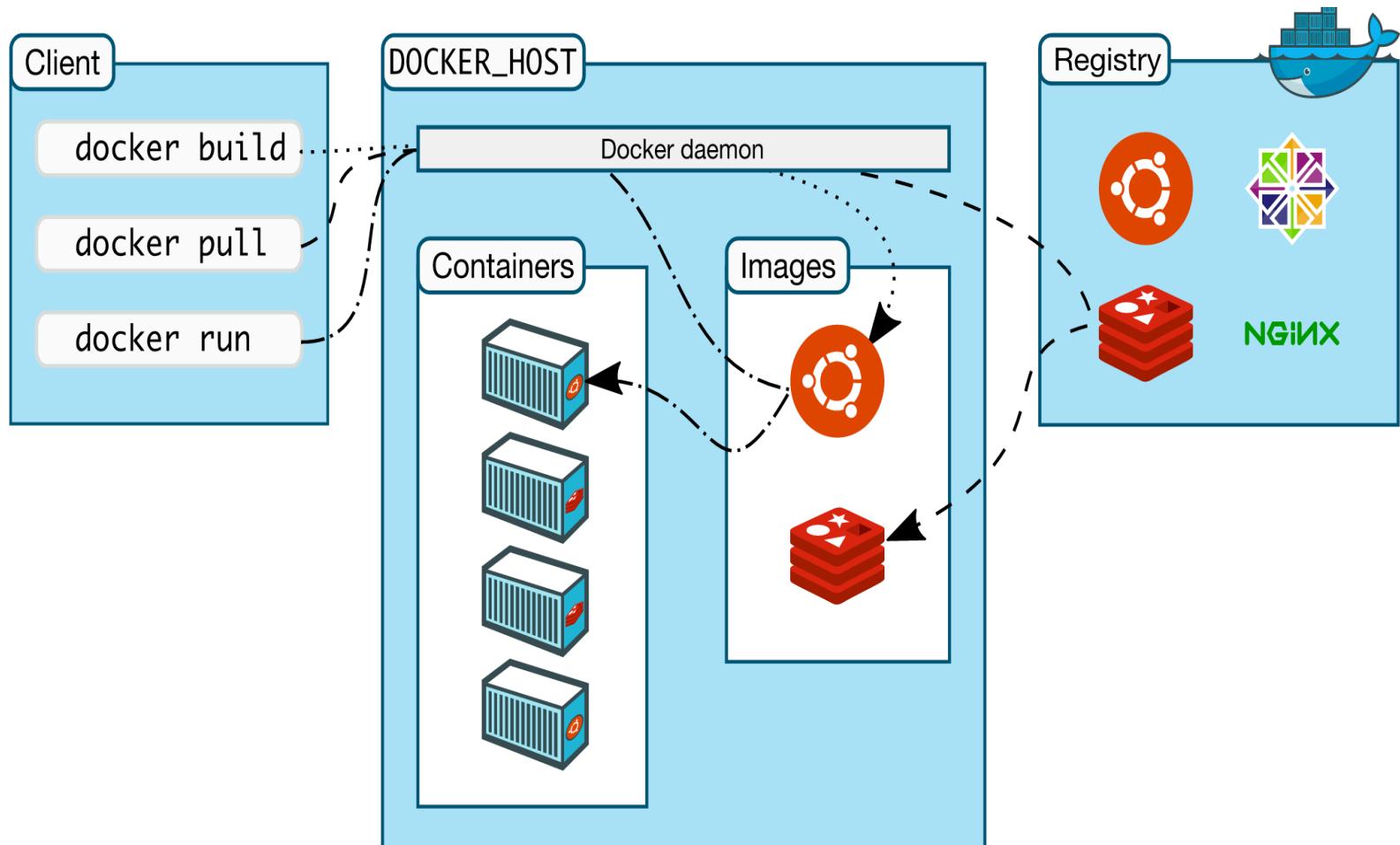


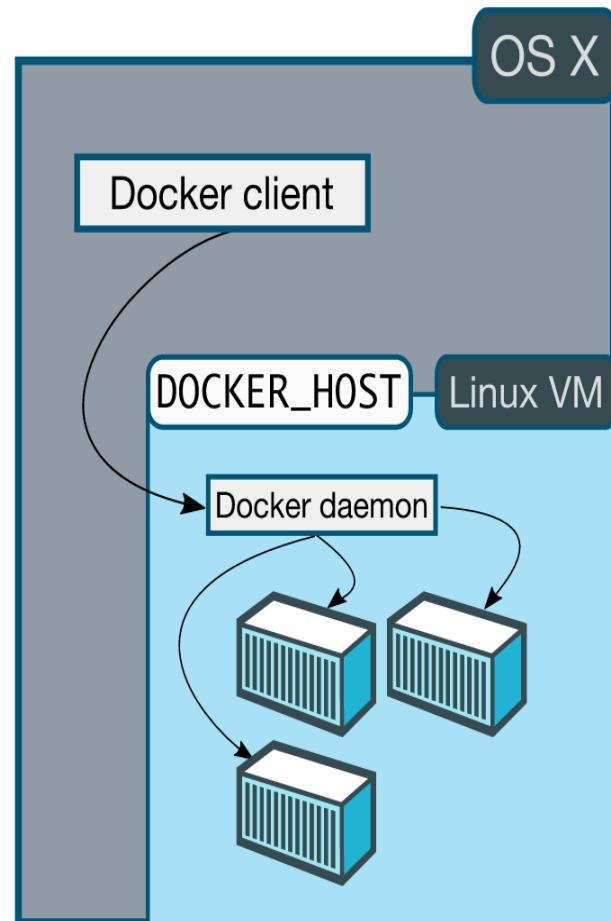
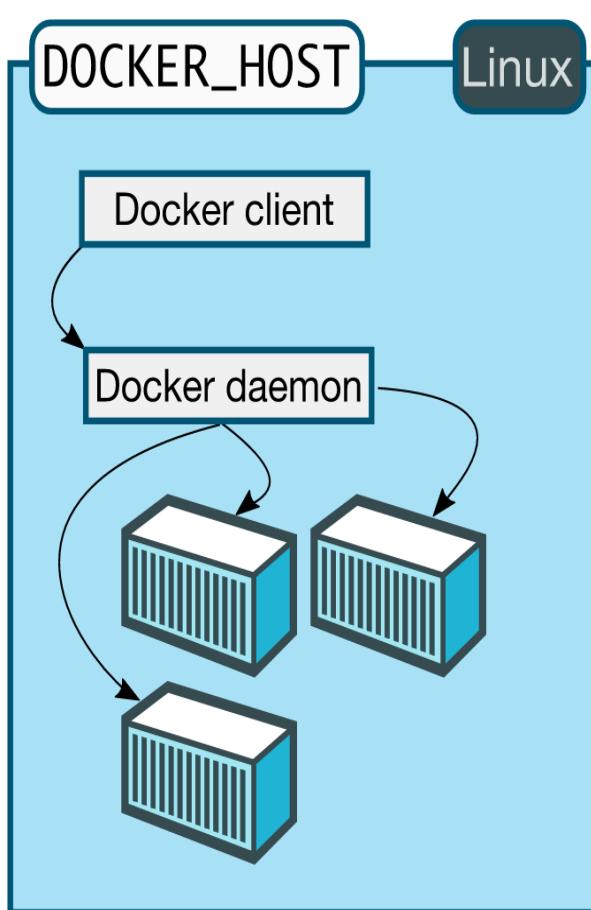
Container Virtualization



Benefits:

- Cost-Efficient
- Fast Deployment
- Guaranteed Portability





Docker

Client-Server Architecture

Install Docker for Mac/Windows

Install Docker Software

This lecture applies to you if:

- You are using **Linux**
- Or you are using Mac and your Mac version is **OS X 10.10.3 or newer**
- Or you are using Windows and your Windows version is **Windows 10 or newer**

Otherwise, you can skip this lecture and follow the installation guide of the next lecture.

Install Docker Toolbox

Install Docker Toolbox

This lecture applies to you if:

- You are using Mac and your Mac version is **older than OS X 10.10.3**.
- Or you are using Windows and your Windows version is **older than Windows 10**.
- Or you want to install Docker Machine or Kitematic instead of Docker Engine.

Otherwise, you can skip this lecture and follow the installation guide of the previous lecture.

If you already installed **Docker for Mac/Windows**, you can skip this lecture for now.



Containers

+ NEW



Search for Docker images from Docker Hub

FILTER BY All Recommended My Repos

busybox
busybox:latestbusybox-2
busybox:latest

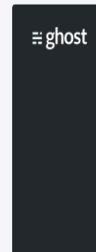
Recommended

kitematic
hello-world-nginxA light-weight nginx container
that demonstrates the features of
Kitematic

♡ 44 ↓ 280K

ooo

CREATE

official
ghostGhost is a free and open source
blogging platform written in
JavaScript

♡ 371 ↓ 2M

ooo

CREATE

official
jenkins

Official Jenkins Docker image

♡ 1.6K ↓ 7M

ooo

CREATE

official
redisRedis is an open source key-
value store that functions as a
data structure server.

♡ 2.3K ↓ 40M

ooo

CREATE

official
rethinkdbRethinkDB is an open-source,
document database that makes it
easy to build and scale realtime...

♡ 266 ↓ 2M

ooo

CREATE

kitematic
minecraftThe Minecraft multiplayer server
allows two or more players to
play Minecraft together

♡ 54 ↓ 24K

ooo

CREATE

official
solrSolr is the popular, blazing-fast,
open source enterprise search
platform built on Apache...

♡ 198 ↓ 177K

ooo

CREATE

official
elasticsearchElasticsearch is a powerful open
source search and analytics
engine that makes data easy to...

♡ 1.3K ↓ 12M

ooo

CREATE

official
postgresThe PostgreSQL object-relational
database system provides
reliability and data integrity.

♡ 2.2K ↓ 13M

ooo

CREATE

official
ubuntu-upstartUpstart is an event-based
replacement for the /sbin/init
daemon which starts processes...

♡ 64 ↓ 209K

ooo

CREATE

official
memcachedFree & open source, high-
performance, distributed memory
object caching system.

♡ 427 ↓ 4M

ooo

CREATE

official
rabbitmqRabbitMQ is a highly reliable
enterprise messaging system
based on the emerging AMQP...

♡ 721 ↓ 6M

ooo

CREATE



official



official



official

Important Docker Concepts

Images

- Images are read only templates used to create containers.
- Images are created with the docker build command, either by us or by other docker users.
- Images are composed of layers of other images.
- Images are stored in a Docker registry.

Containers

- If an image is a class, then a container is an instance of a class - a runtime object.
- Containers are lightweight and portable encapsulations of an environment in which to run applications.
- Containers are created from images. Inside a container, it has all the binaries and dependencies needed to run the application.

Registries and Repositories

- A registry is where we store our images.
- You can host your own registry, or you can use Docker's public registry which is called DockerHub.
- Inside a registry, images are stored in repositories.
- Docker repository is a collection of different docker images with the same name, that have different tags, each tag usually represents a different version of the image.

Why Using Official Images

- **Clear Documentation**
- **Dedicated Team for Reviewing Image Content**
- **Security Update in a Timely Manner**

Run our First Hello World Docker Container

Deep Dive into Docker Containers

- running containers in detached mode
- docker ps command
- docker container name
- docker inspect command

	Run Container in Foreground	Run Container in Background
<i>Description</i>	Docker run starts the process in the container and attaches the console to the process's standard input, output, and standard error.	Containers started in detached mode and exit when the root process used to run the container exits.
<i>How to specify?</i>	default mode	-d option
<i>Can the console be used for other commands after the container is started up?</i>	No	Yes

Docker Port Mapping and Docker Logs

Docker Image Layers

Image Layers

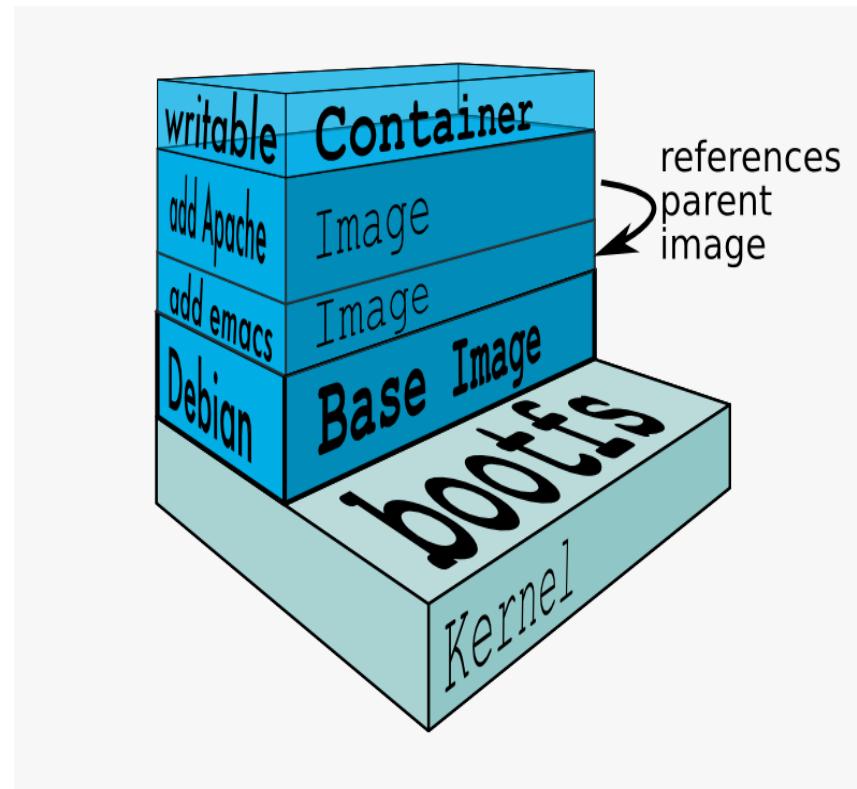
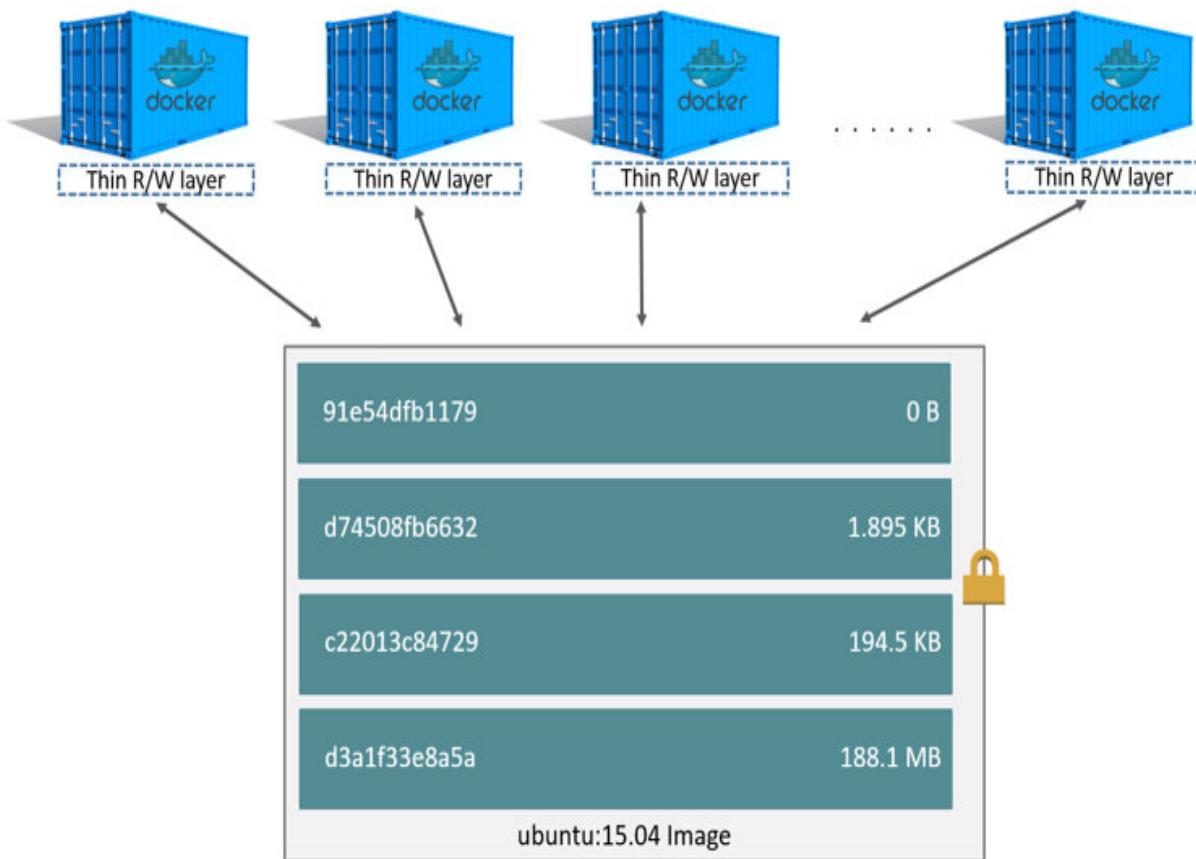


Image Layers



Build Docker Images

Approach 1: committing changes made in a container

Ways to Build a Docker Image

- Commit changes made in a Docker container.
- Write a Dockerfile.

Steps

1. Spin up a container from a base image.
2. Install Git package in the container.
3. Commit changes made in the container.

Docker commit

- Docker commit command would save the changes we made to the Docker container's file system to a new image.

docker commit container_ID repository_name:tag

Build Docker Images

Approach 2: Writing a Dockerfile

Dockerfile and Instructions

- A Dockerfile is a text document that contains all the instructions users provide to assemble an image.
- Each instruction will create a new image layer to the image.
- Instructions specify what to do when building the image.

Docker Build Context

- Docker build command takes the path to the build context as an argument.
- When build starts, docker client would pack all the files in the build context into a tarball then transfer the tarball file to the daemon.
- By default, docker would search for the Dockerfile in the build context path.

Dockerfile In Depth

Steps

1. Spin up a container from a base image.
2. Install Git package in the container.
3. Commit changes made in the container.

Chain RUN Instructions

- Each RUN command will execute the command on the top writable layer of the container, then commit the container as a new image.
- The new image is used for the next step in the Dockerfile. So each RUN instruction will create a new image layer.
- It is recommended to chain the RUN instructions in the Dockerfile to reduce the number of image layers it creates.

Sort Multi-line Arguments Alphanumerically

- This will help you avoid duplication of packages and make the list much easier to update.

CMD Instructions

- CMD instruction specifies what command you want to run when the container starts up.
- If we don't specify CMD instruction in the Dockerfile, Docker will use the default command defined in the base image.
- The CMD instruction doesn't run when building the image, it only runs when the container starts up.
- You can specify the command in either exec form which is preferred or in shell form.

Docker Cache

- Each time Docker executes an instruction it builds a new image layer.
- The next time, if the instruction doesn't change, Docker will simply reuse the existing layer.

Docker Cache

- Each time Docker executes an instruction it builds a new image layer.
- The next time, if the instruction doesn't change, Docker will simply reuse the existing layer.

Dockerfile with Aggressive Caching

```
FROM ubuntu:14.04
```

reusing cache

```
RUN apt-get update
```

reusing cache

```
RUN apt-get install -y git
```

curl

Cache Busting

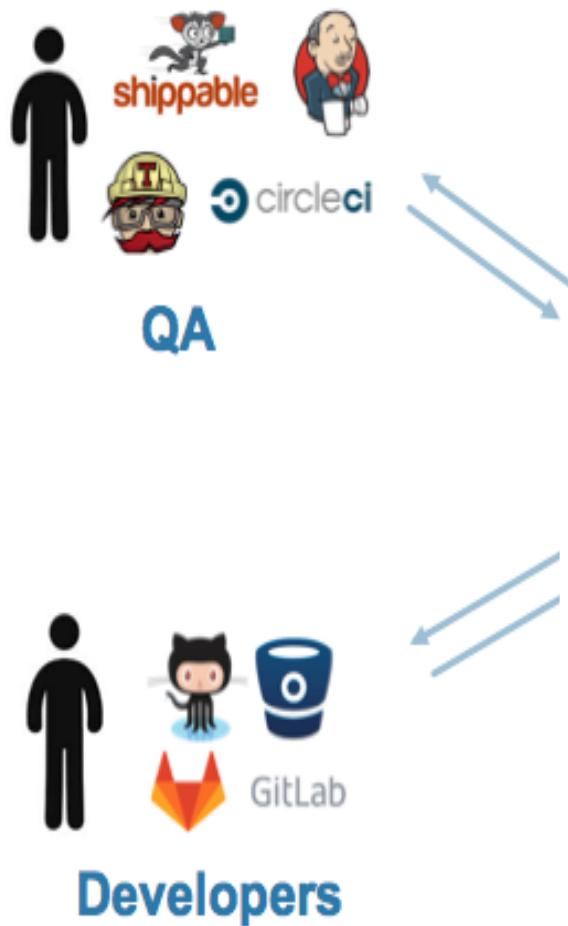
```
FROM ubuntu:14.04
```

```
RUN apt-get update && apt-get install -y \  
git \  
curl
```

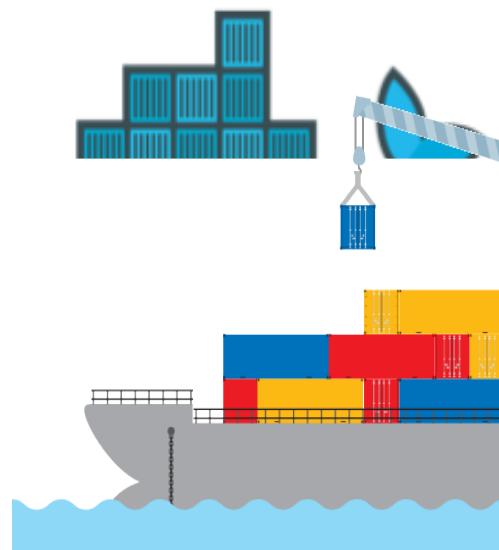
Cache Busting

- You can also achieve cache-busting by specifying a package version. This is known as version pinning.

```
RUN apt-get update && apt-get install -y \
  package-bar \
  package-baz \
  package-foo=1.3.*
```



Docker Hub



Containerize a Hello World Web Application

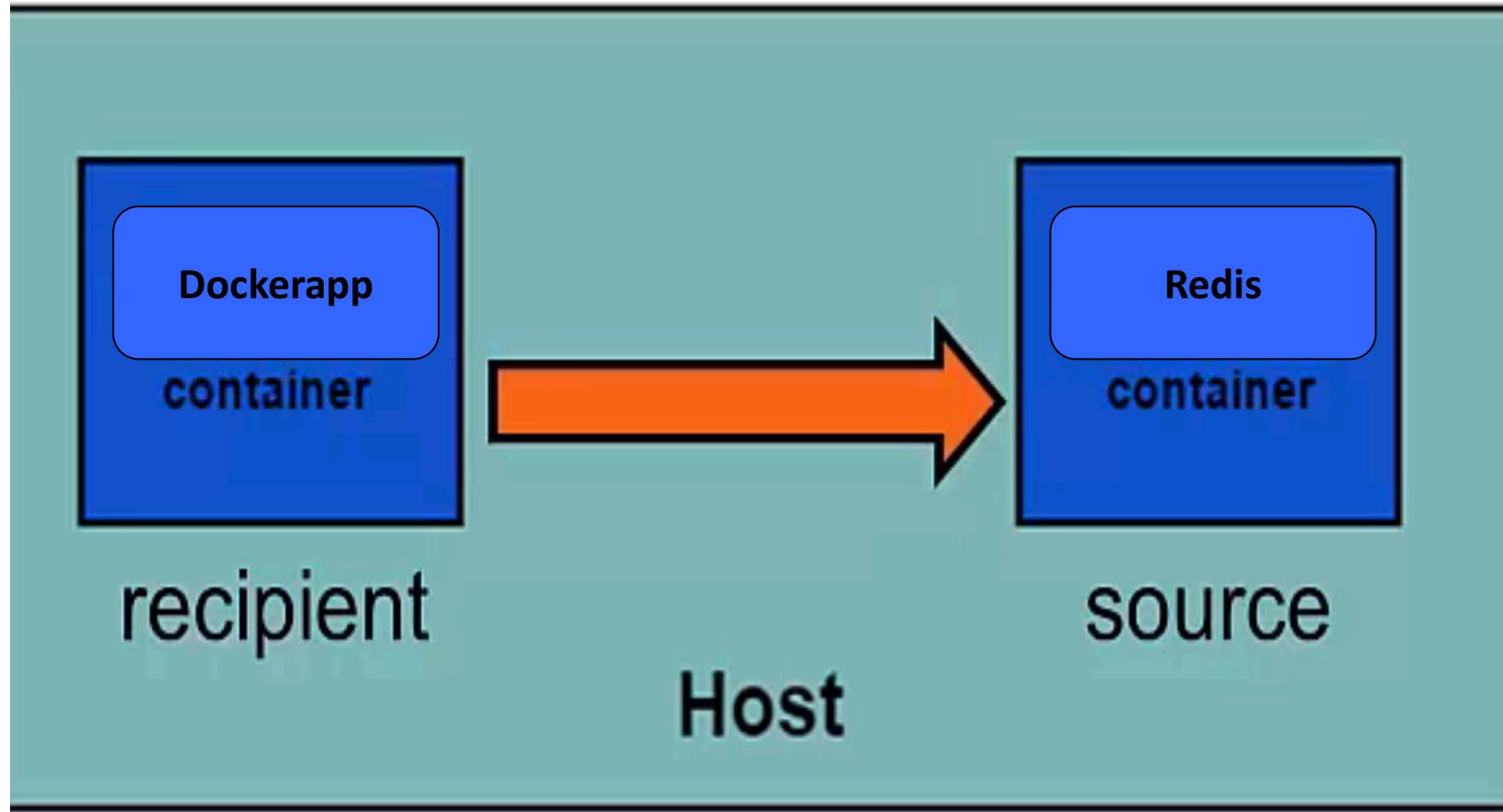
Text Direction: Dockerize a Hello World Web Application

Check out source code:

```
git clone -b v0.1 https://github.com/jleetutorial/dockerapp.git
```

Docker Container Links

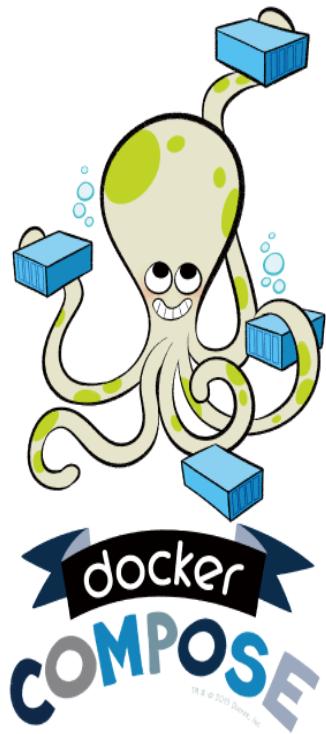
Docker Container Links



How container links work behind the scenes?

Benefits of Docker Container Links

- The main use for docker container links is when we build an application with a microservice architecture, we are able to run many independent components in different containers.
- Docker creates a secure tunnel between the containers that doesn't need to expose any ports externally on the container.



Automate the Docker Workflow with Docker Compose

Deep Dive into Docker Compose Workflow

Why Docker Compose?

Manual linking containers and configuring services become impractical when the number of containers grows.

Docker Compose

- Docker compose is a very handy tool to quickly get docker environment up and running.
- Docker compose uses yaml files to store the configuration of all the containers, which removes the burden to maintain our scripts for docker orchestration.

Docker Compose Commands

- **docker compose up** starts up all the containers.
- **docker compose ps** checks the status of the containers managed by docker compose.
- **docker compose logs** outputs colored and aggregated logs for the compose-managed containers.
- **docker compose logs** with dash f option outputs appended log when the log grows.
- **docker compose logs** with the container name in the end outputs the logs of a specific container.
- **docker compose stop** stops all the running containers without removing them.
- **docker compose rm** removes all the containers.
- **docker compose build** rebuilds all the images.

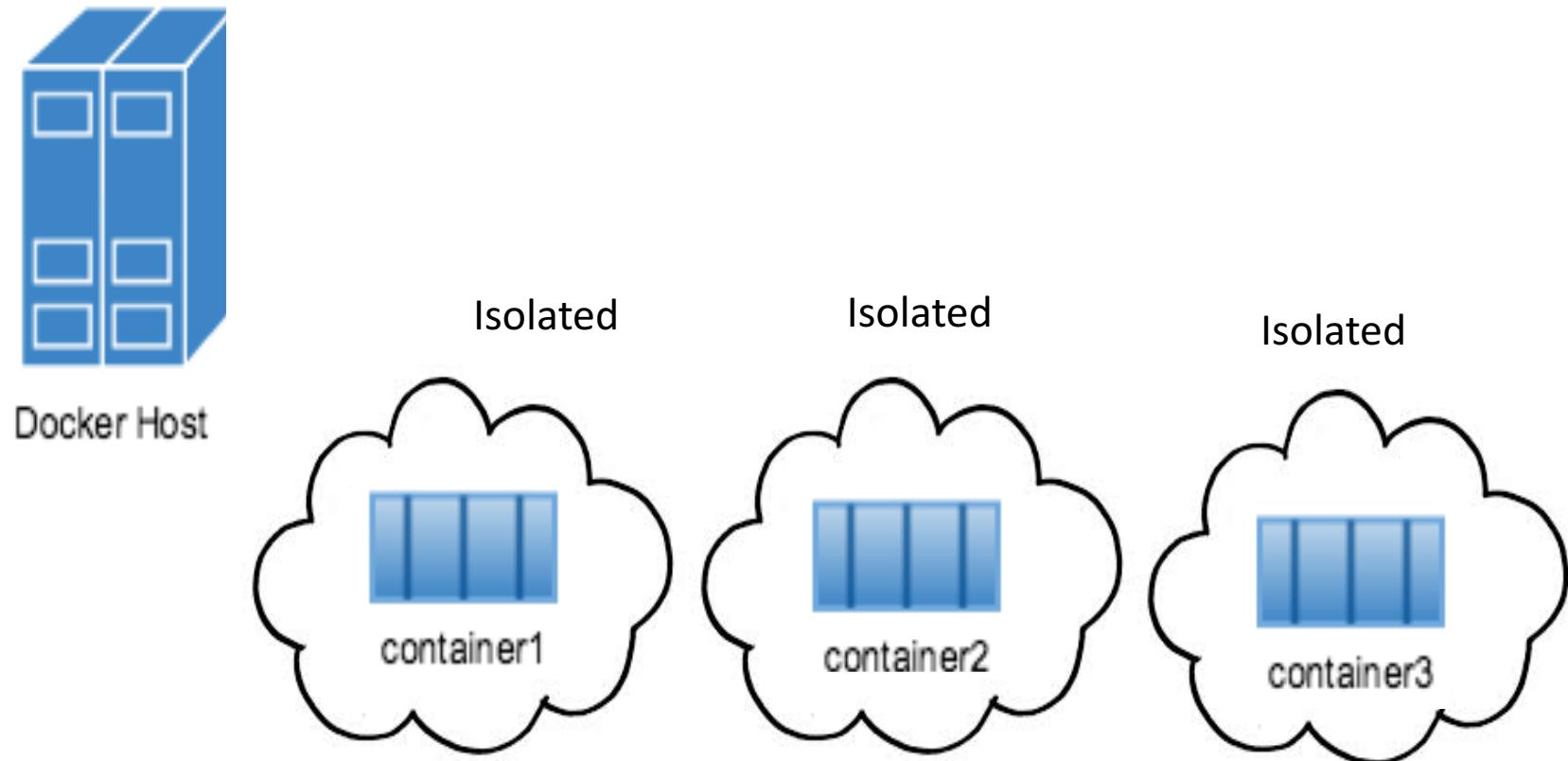
Introduction to Docker Networking

Docker Network Types

- Closed Network / None Network
- Bridge Network
- Host Network
- Overlay Network

None Network

None Network

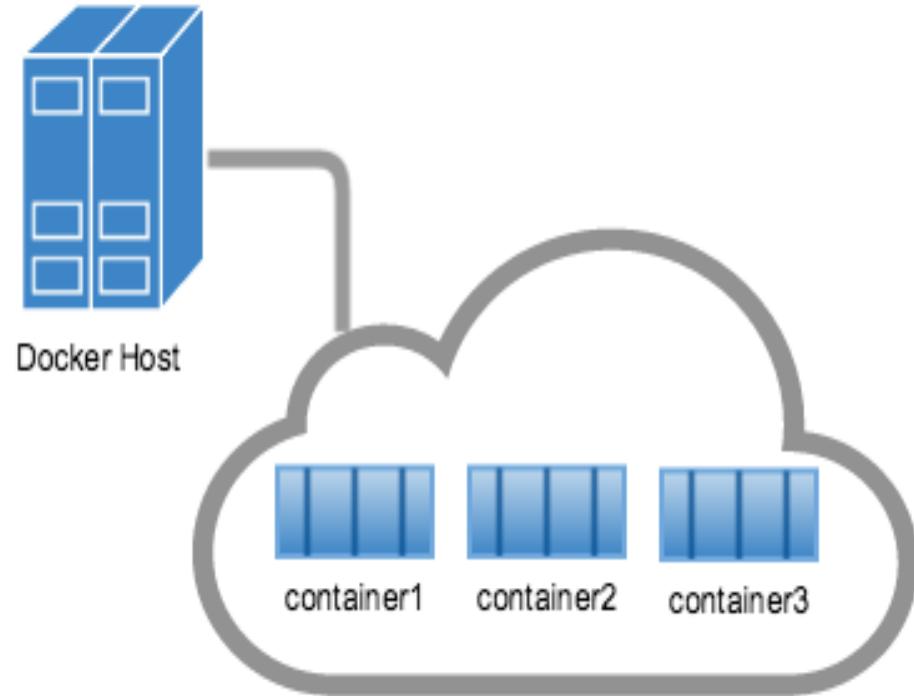


None Network

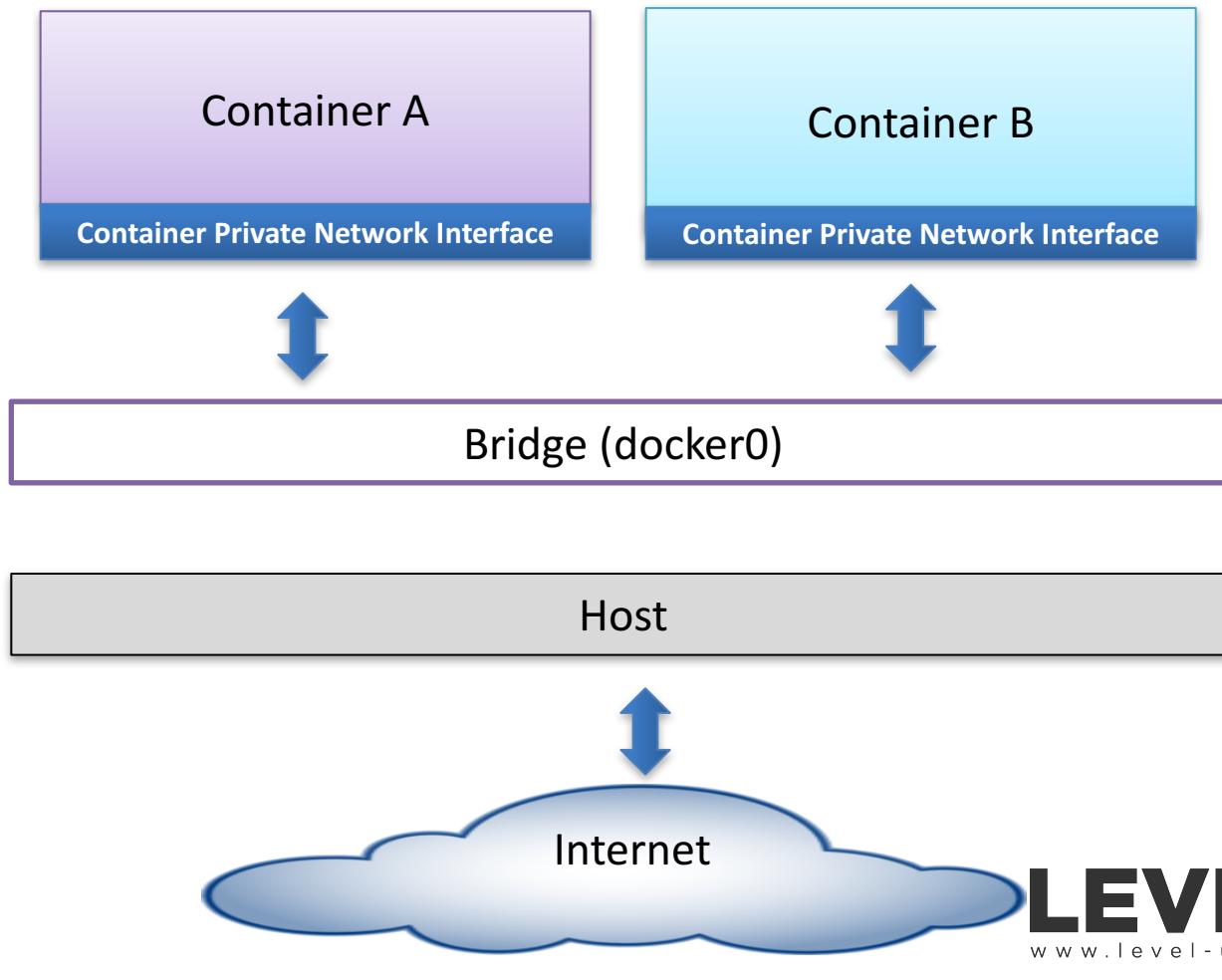
- Provides the maximum level of network protection.
- Not a good choice if network or Internet connection is required.
- Suites well where the container require the maximum level of network security and network access is not necessary.

Bridge Network

Bridge Network



Bridge Network



Host and Overlay Network

Host Network

- The least protected network model, it adds a container on the host's network stack.
- Containers deployed on the host stack have full access to the host's interface.
- This kind of containers are usually called open containers.

Host Network

- Minimum network security level.
- No isolation on this type of open containers, thus leave the container widely unprotected.
- Containers running in the host network stack should see a higher level of performance than those traversing the docker0 bridge and iptables port mappings.

Define Container Networks with Docker Compose

Write and Run Unit Tests in Docker Containers

Unit Tests in Containers

- Unit tests should test some basic functionality of our docker app code, with no reliance on external services.
- Unit tests should run as quickly as possible so that developers can iterate much faster without being blocked by waiting for the tests results.
- Docker containers can spin up in seconds and can create a clean and isolated environment which is great tool to run unit tests with.

Incorporating Unit Tests into Docker Images

Pros:

- A single image is used through development, testing and production, which greatly ensures the reliability of our tests.

Cons:

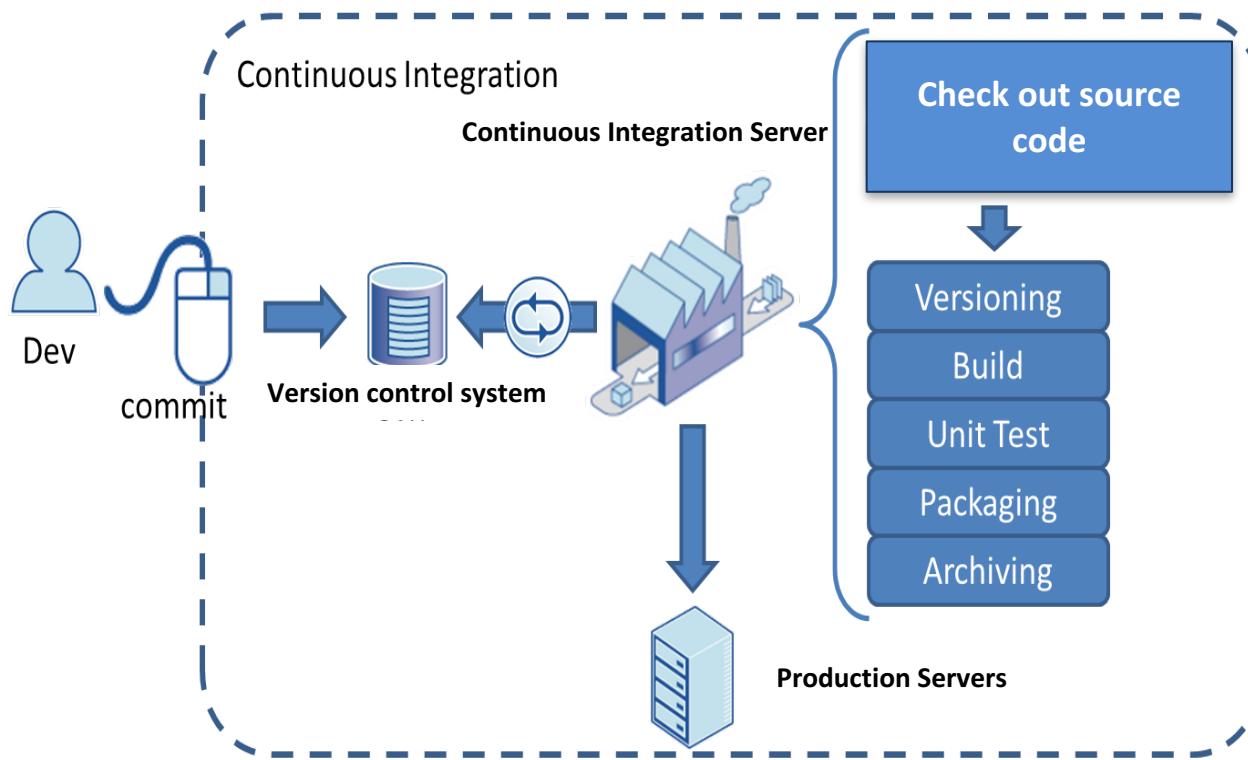
- It increases the size of the image.

Fit Docker Technology into Continuous Integration(CI) Process

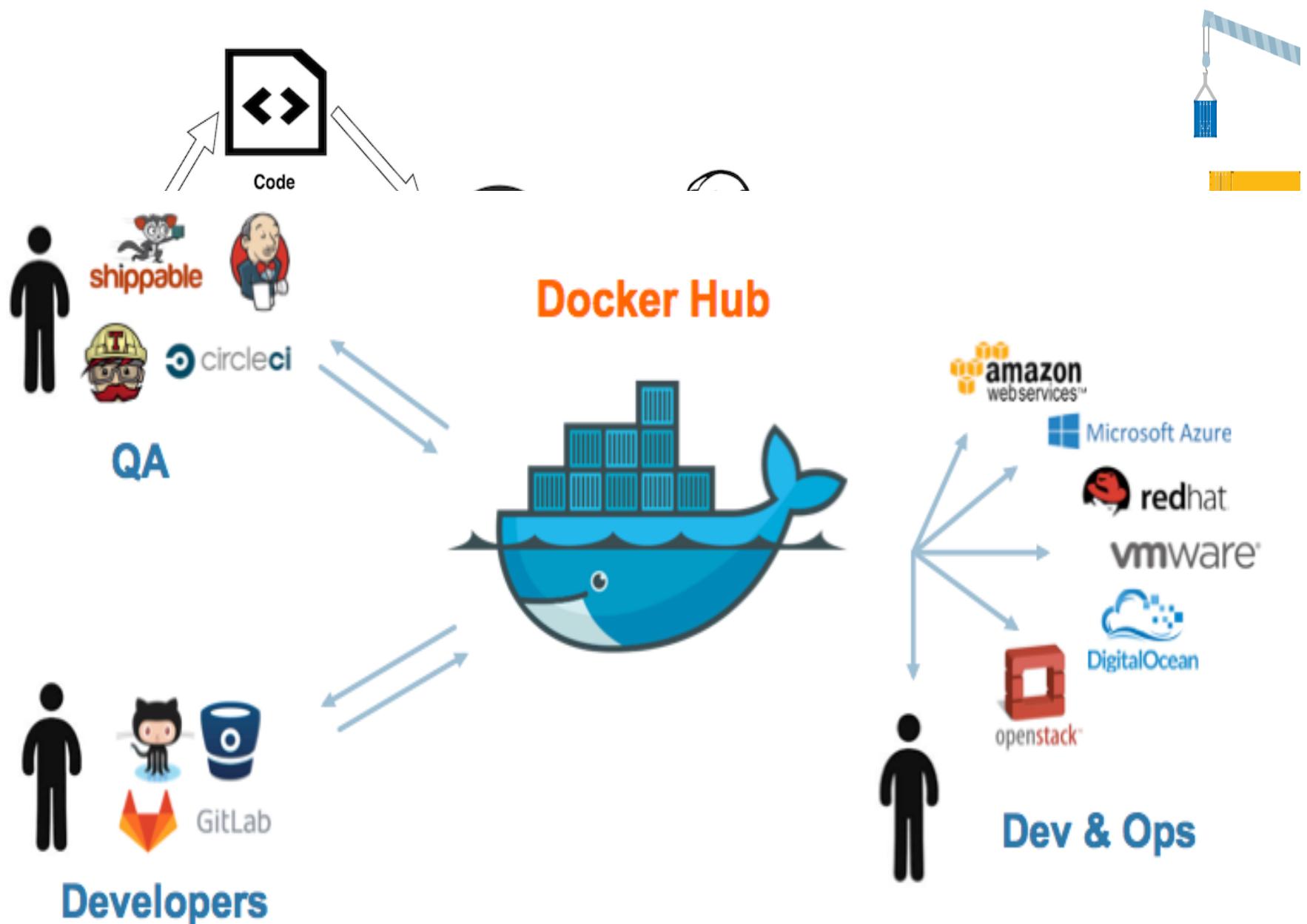
What is Continuous Integration?

- Continuous integration is a software engineering practice in which isolated changes are immediately tested and reported when they are added to a larger code base.
- The goal of Continuous integration is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible.

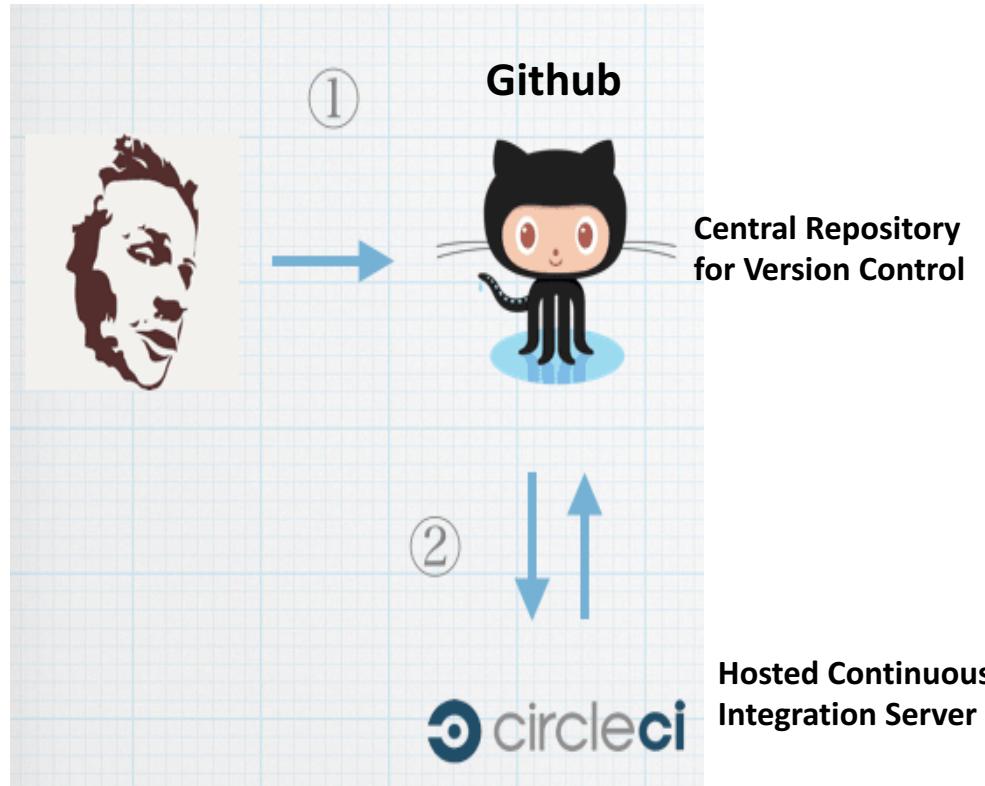
A Typical CI Pipeline without Docker



CI process with Docker technologies involved



Our Continuous Integration Pipeline



Text Direction: Introduction to Continuous Integration

URL of the Github account to fork:

<https://github.com/jleetutorial/dockerapp>

Checking for existing SSH keys:

<https://help.github.com/articles/checking-for-existing-ssh-keys/>

Generating a new SSH key and adding it to the ssh-agent:

<https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

Adding a new SSH key to your GitHub account:

<https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/>

Link Circle CI with GitHub Account

to build a Continuous Integration pipeline

Set up SSH keys for Github Account

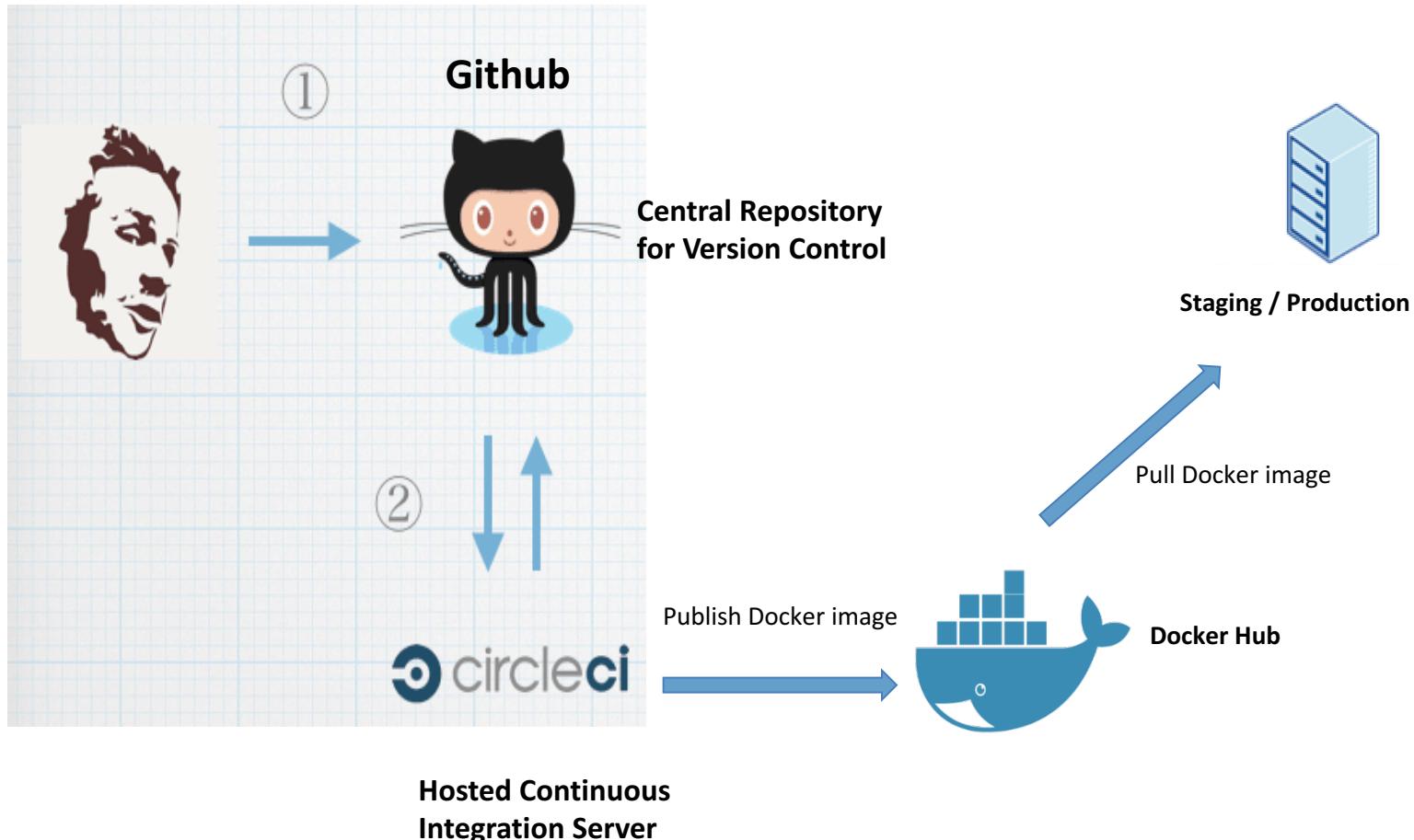
- SSH keys are a way to identify trusted computers without involving password.
- Generate a SSH key pair and save the private SSH key in your local box and add the public key to your GitHub account.
- Then you can directly push your changes to github repository without typing password.

How to check if SSH public key files are available on your local box?

The SSH public key file usually sits under `~/.ssh/` directory and ends with `.pub` extension.

Link Circle CI with GitHub Account

Complete CI Workflow



Tag the Docker Images with Two Tags

1. commit hash of the source code
2. latest

Introduction to Running Docker in Production

Opinions about Running Docker in Production

- On one hand, many docker pioneers are confident that a distributed web app can be deployed at scale using Docker and have incorporated Docker into their production environment.
- On the other hand, there are still some people who are reluctant to use Docker in production as they think docker workflow is too complex or unstable for real life use cases.

Is Docker Production Ready Now?

Concerns about Running Docker in Production

- There are still some missing pieces about Docker around data persistence, networking, security and identity management.
- The ecosystem of supporting Dockerized applications in production such as tools for monitoring and logging are still not fully ready yet.

Companies which already run Docker in Production



Why Running Docker Containers inside VMs?

- To address security concerns.
- Hardware level isolation.

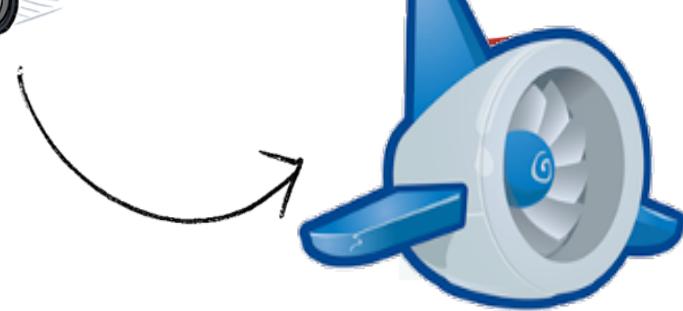
They all run containers inside VMs



Docker Machine



VirtualBox



Google app engine



VirtualBox

LEVEL UP
www.level-up.one

Register Digital Ocean Account to Deploy Containerized Applications

Deploy Docker App to the Cloud with Docker Machine

Text Direction: Deploy Docker App to the Cloud with Docker Machine

Docker Machine Create command

```
docker-machine create --driver digitalocean --digitalocean-access-token  
<xxxxx> docker-app-machine
```

Introduction to Docker Swarm and Set up Swarm cluster

How Swarm cluster works

- To deploy your application to a swarm, you submit your service to a manager node.
- The manager node dispatches units of work called tasks to worker nodes.
- Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm.
- Worker nodes receive and execute tasks dispatched from manager nodes.
- An agent runs on each worker node and reports on the tasks assigned to it. The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker.

Deploy Docker App Services to the Cloud via Docker Swarm

Docker
Swarm



Docker Services

- The services can be defined in our Docker compose file.
- The service definition includes which Docker images to run, the port mapping and dependency between services.

```
version: "3.0"
services:
  dockerapp:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - redis
  redis:
    image: redis:3.2.0
```

Docker Services

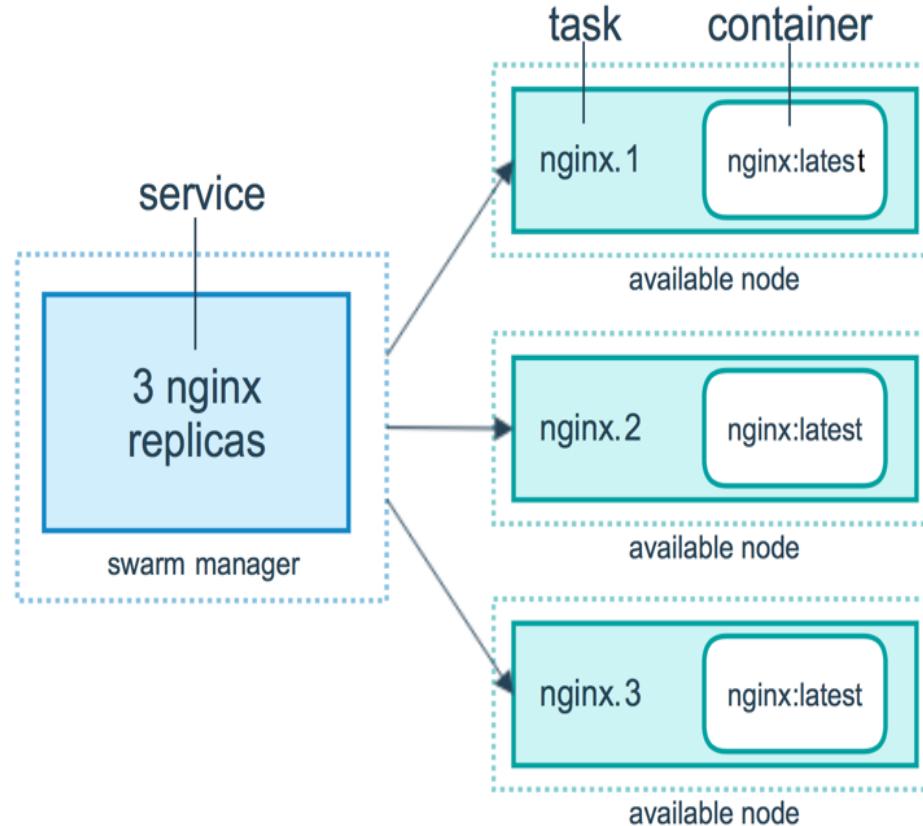
- When we are deploying services in the swarm mode, we can also set another important configuration, which is the deploy key and it is only available on Compose file formats version 3.x and up.
- The deploy key and its sub-options can be used to load balance and optimize performance for each service.

```
version: "3.0"
services:
  dockerapp:
    image: jleetutorial/dockerapp
    ports:
      - "5000:5000"
    depends_on:
      - redis
    deploy:
      replicas: 2
    redis:
      image: redis:3.2.0
```

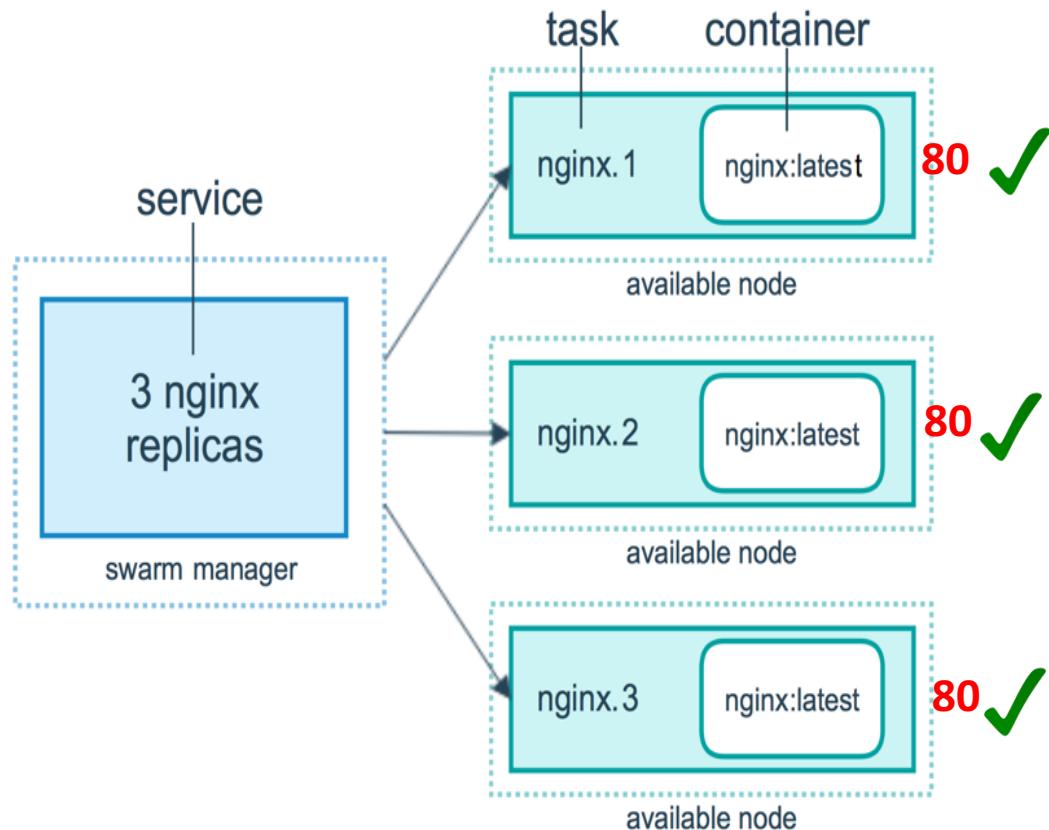
Deploy Key in Docker Compose file

```
docker-compose.yml •
1 version: "3.3"
2 services:
3   nginx:
4     image: nginx:latest
5     ports:
6       - "80:80"
7     deploy:
8       replicas: 3
9       resources:
10      limits:
11        cpus: '0.1'
12        memory: 50M
13     restart_policy:
14       condition: on-failure
15
```

Replicas

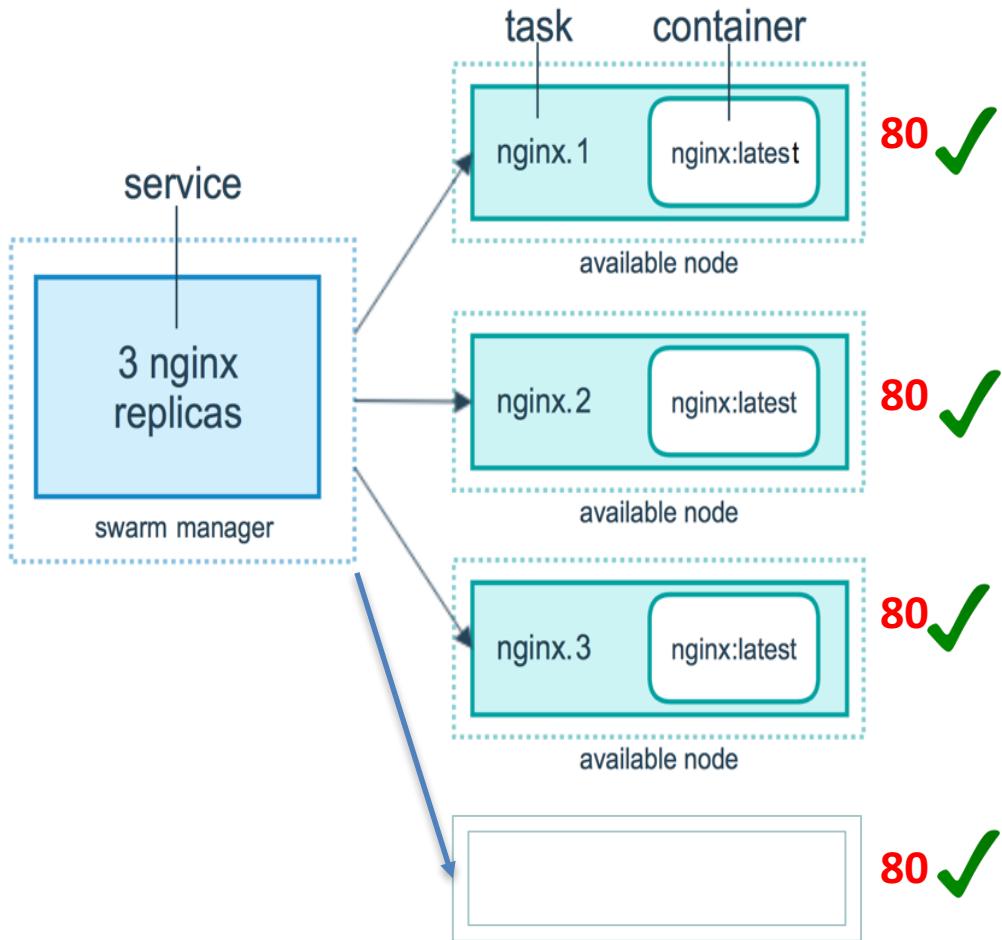


Replicas



```
docker-compose.yml
1 version: "3.3"
2 services:
3   nginx:
4     image: nginx:latest
5     ports:
6       - "80:80"
7     deploy:
8       replicas: 3
9     resources:
10       limits:
11         cpus: '0.1'
12         memory: 50M
13     restart_policy:
14       condition: on-failure
15
```

Replicas



- We can connect to the nginx service through a node which does NOT have nginx replicas.
- *Ingress load balancing*
 - All nodes listen for connections to published service ports.
 - When that service is called by external systems, the receiving node will accept the traffic and internally load balance it using an internal DNS service that Docker maintains.

Docker Stack

- A docker stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together.
- You can image that a stack is a live collection of all the services defined in your docker compose file.
- Create a stack from your docker compose file:
 - `docker stack deploy`
- In the Swarm mode,
 - Docker compose files can be used for service definitions.
 - Docker compose commands can't be reused. Docker compose commands can only schedule the containers to a single node.
 - We have to use `docker stack` command. You can think of `docker stack` as the docker compose in the swarm mode.

How to update our services in Production?

Provision a Swarm Cluster

- Step 1: Deploy two VMs, one will be used for the Swarm manager node, and the other one will be used as a worker node.
- Step 2: Appoint the first VM as Swarm manager node and initialize a Swarm cluster.
 - `docker swarm init`
- Step 3: Let the second VM join the Swarm cluster as a worker node.
 - `docker swarm join`

Provision a Swarm Cluster

- Step 1: Deploy two VMs, one will be used for the Swarm manager node, and the other one will be used as a worker node.
- Step 2: Appoint the first VM as Swarm manager node and initialize a Swarm cluster.
 - `docker swarm init`
- Step 3: Let the second VM join the Swarm cluster as a worker node.
 - `docker swarm join`

Provision a Swarm Cluster

- Step 1: Deploy two VMs, one will be used for the Swarm manager node, and the other one will be used as a worker node.
- Step 2: Appoint the first VM as Swarm manager node and initialize a Swarm cluster.
 - `docker swarm init`
- Step 3: Let the second VM join the Swarm cluster as a worker node.
 - `docker swarm join`

Docker Swarm commands

- **docker swarm init**
 - Initialize a swarm. The docker engine targeted by this command becomes a manager in the newly created single-node swarm.
- **docker swarm join**
 - Join a swarm as a Swarm node.
- **docker swarm leave**
 - Leave the swarm.