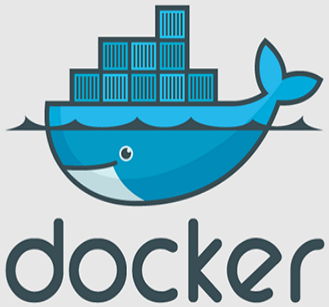
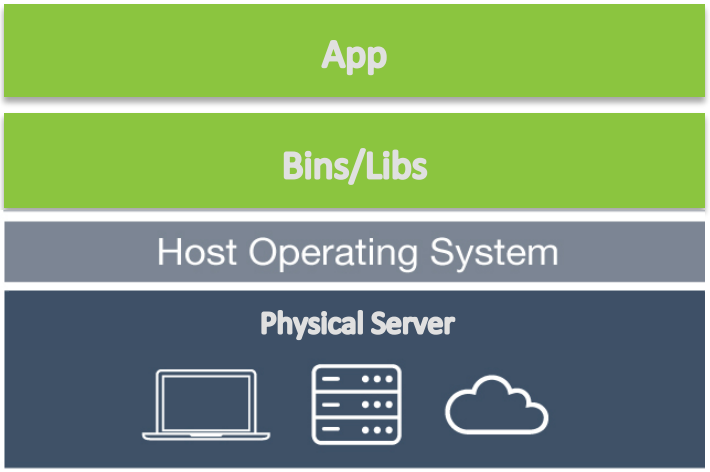
Docker Tutorial for DevOps: Run Docker Containers



Introduction to Virtualization Technologies

Pre-Virtualization World

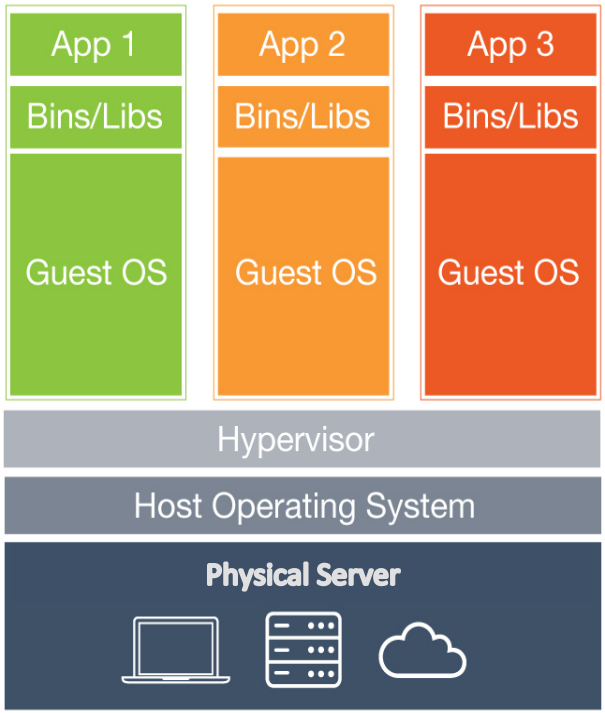


Pre-Virtualization

Problems:

- Huge Cost
- Slow Deployment
- Hard to Migrate

Hypervisor-based Virtualization



Hypervisor-based Virtualization

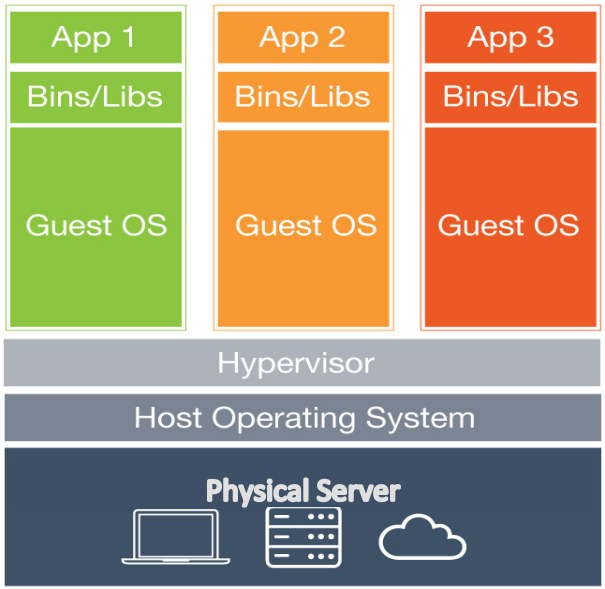
Benefits:

- Cost-Efficient
- Easy to Scale

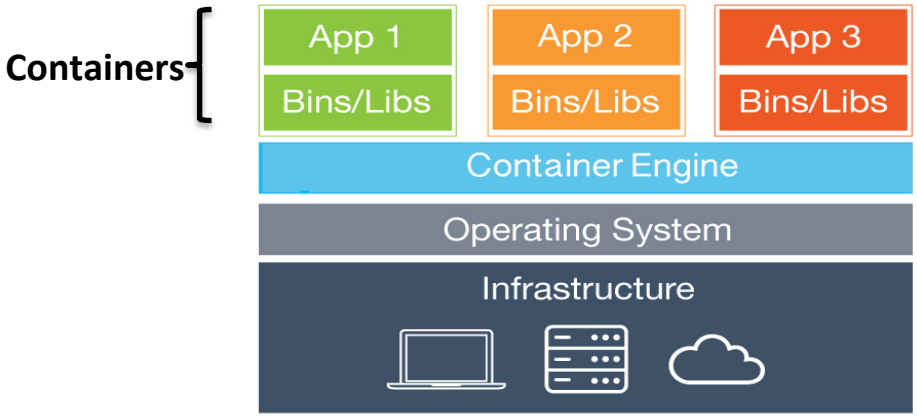
Limitations:

- Kernel Resource Duplication
- Application Portability Issue

Hypervisor-based *VS* Container-based Virtualization

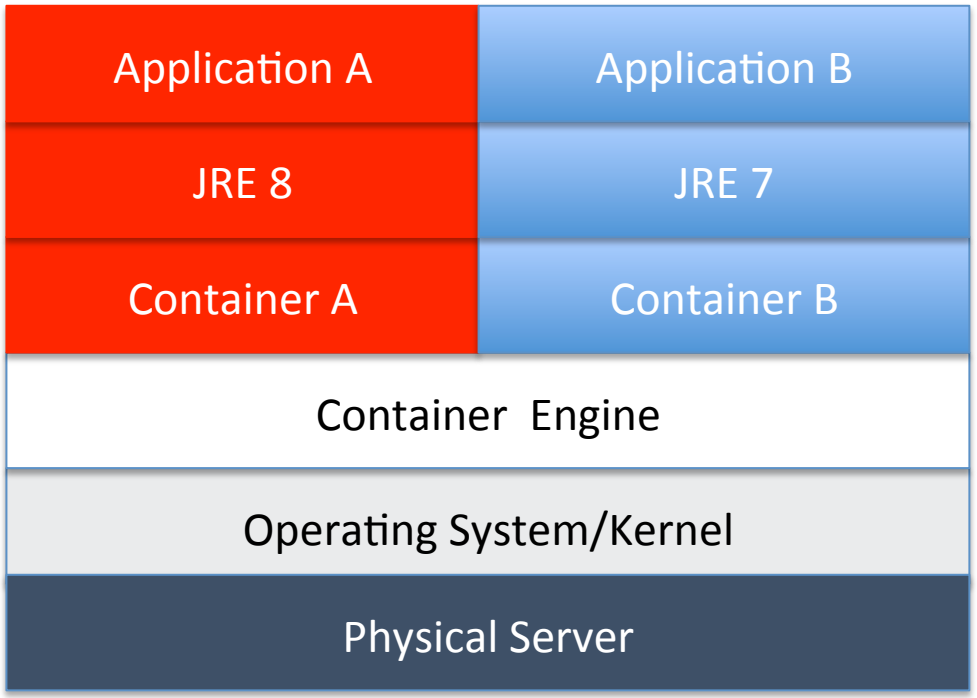


Hypervisor-based Virtualization

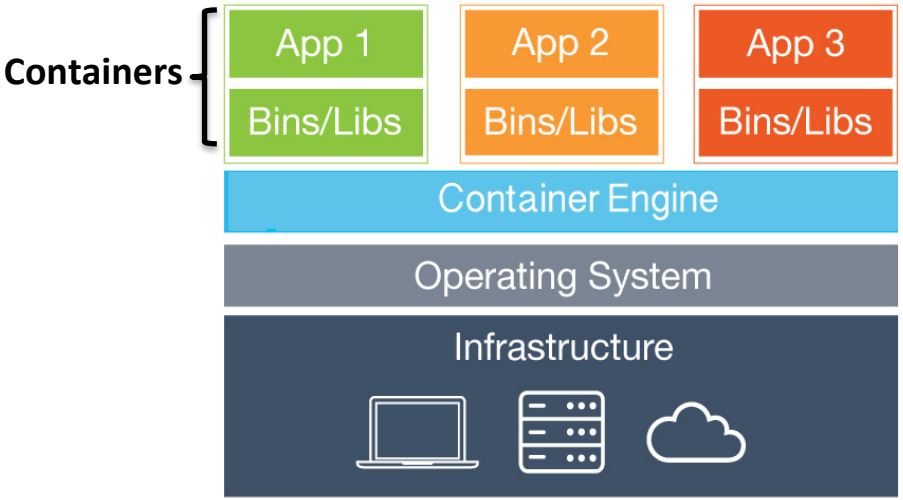


Container-based Virtualization

Runtime Isolation



Container Virtualization



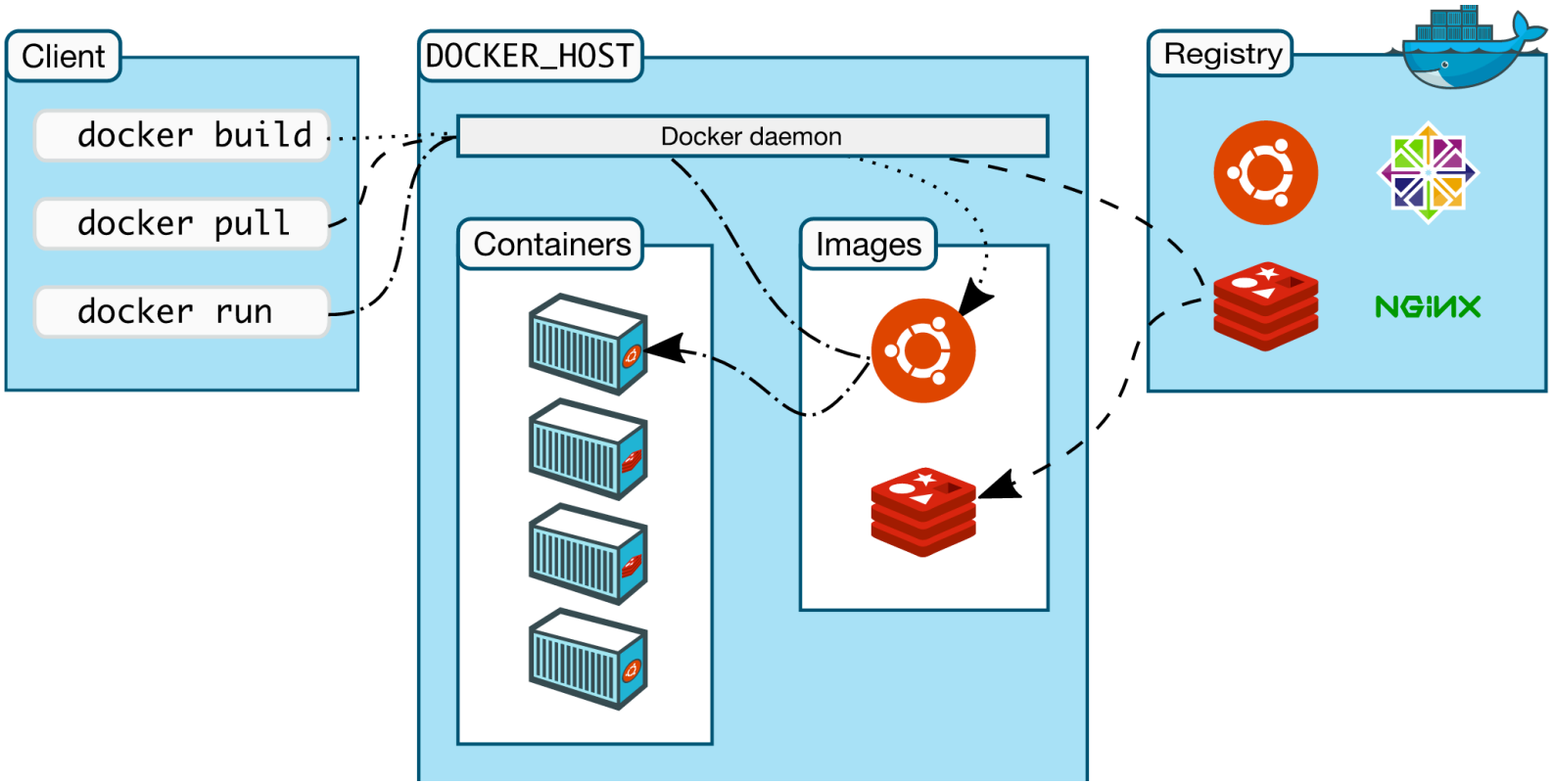
Container Virtualization

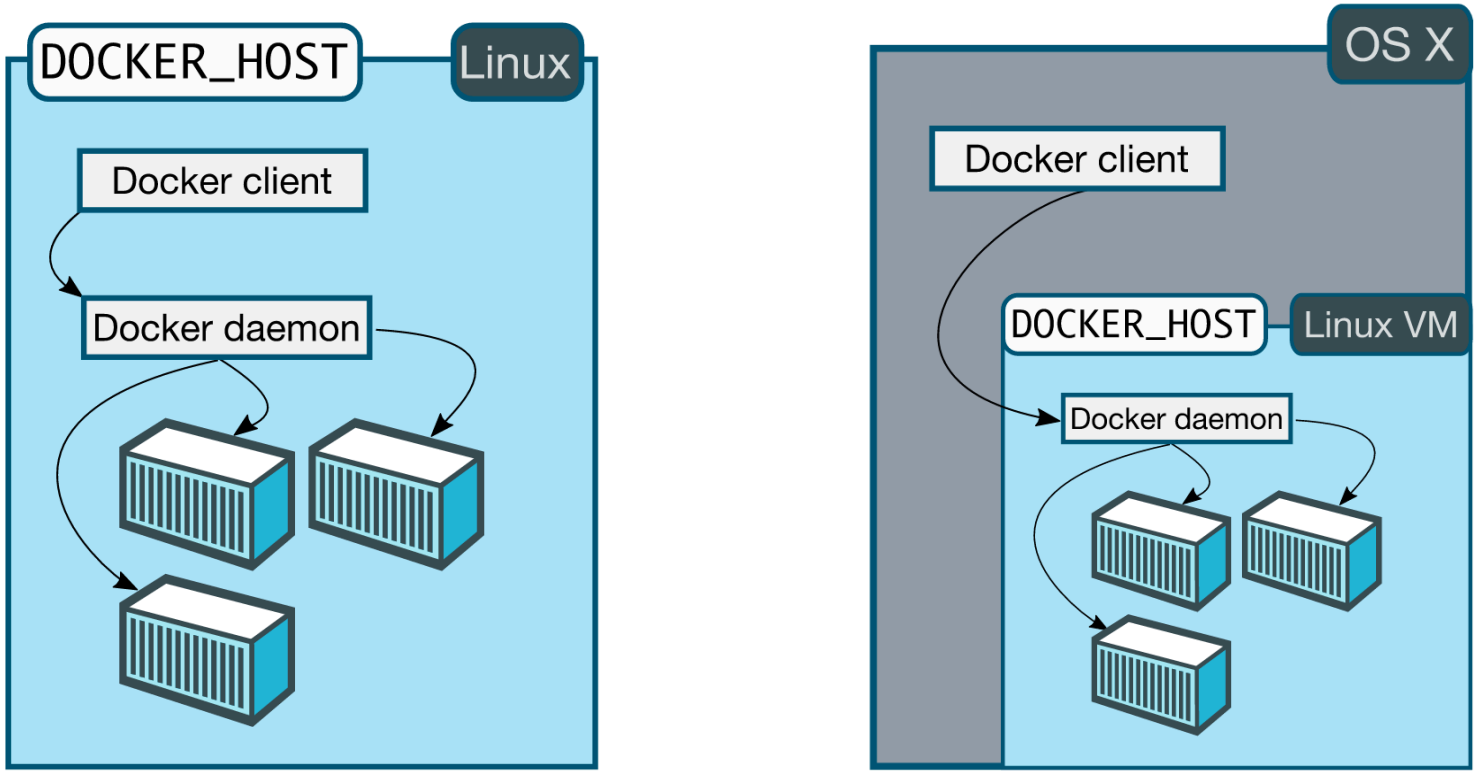
Benefits:

- Cost-Efficient
- Fast Deployment
- Guaranteed Portability

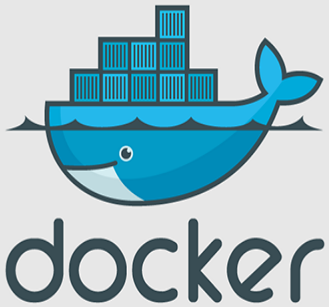
Docker

Client-Server Architecture


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX


DOCKER_HOST
Linux
Docker client
Docker daemon
OS X
Docker client
DOCKER_HOST
Linux VM
Docker daemon

Install Docker for Mac/Windows



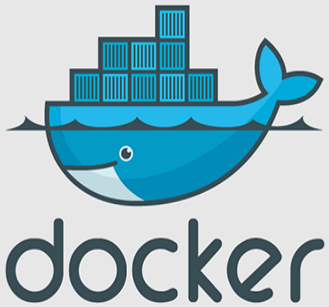
Install Docker

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 Compose, 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.

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.

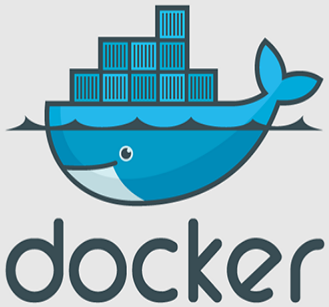
Registries and Repositories

- 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



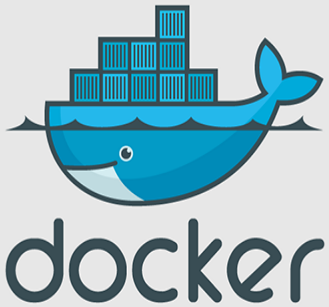
Deep Dive into Docker Containers

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

Foreground vs Detached

	Run Container in Foreground	Run Container in Background
Description	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.
How to specify?	default mode	-d option
Can the console be used for other commands after the container is started up?	No	Yes

Docker Port Mapping and Docker Logs



Docker Image Layers

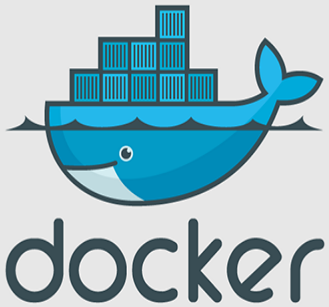


Image Layers

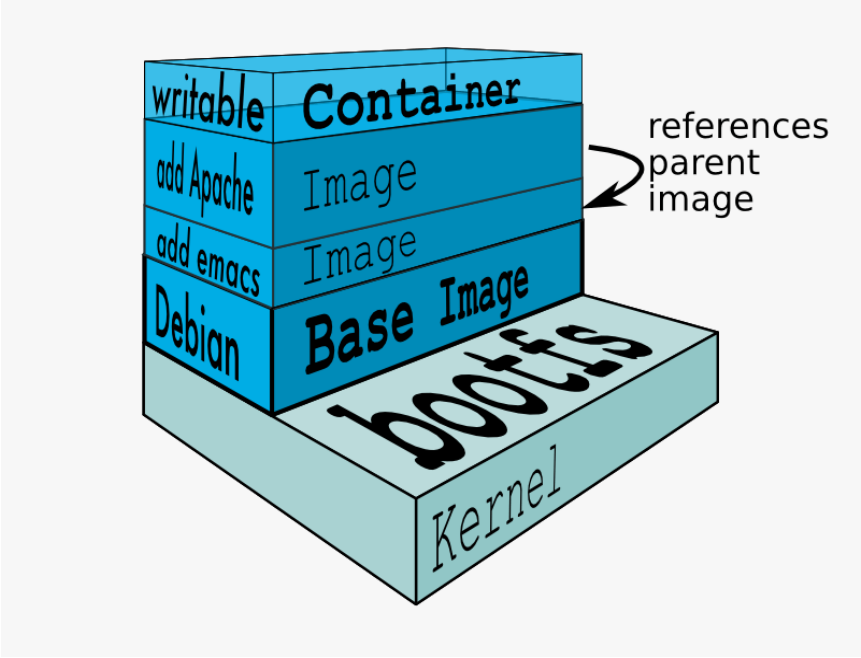


Image Layers

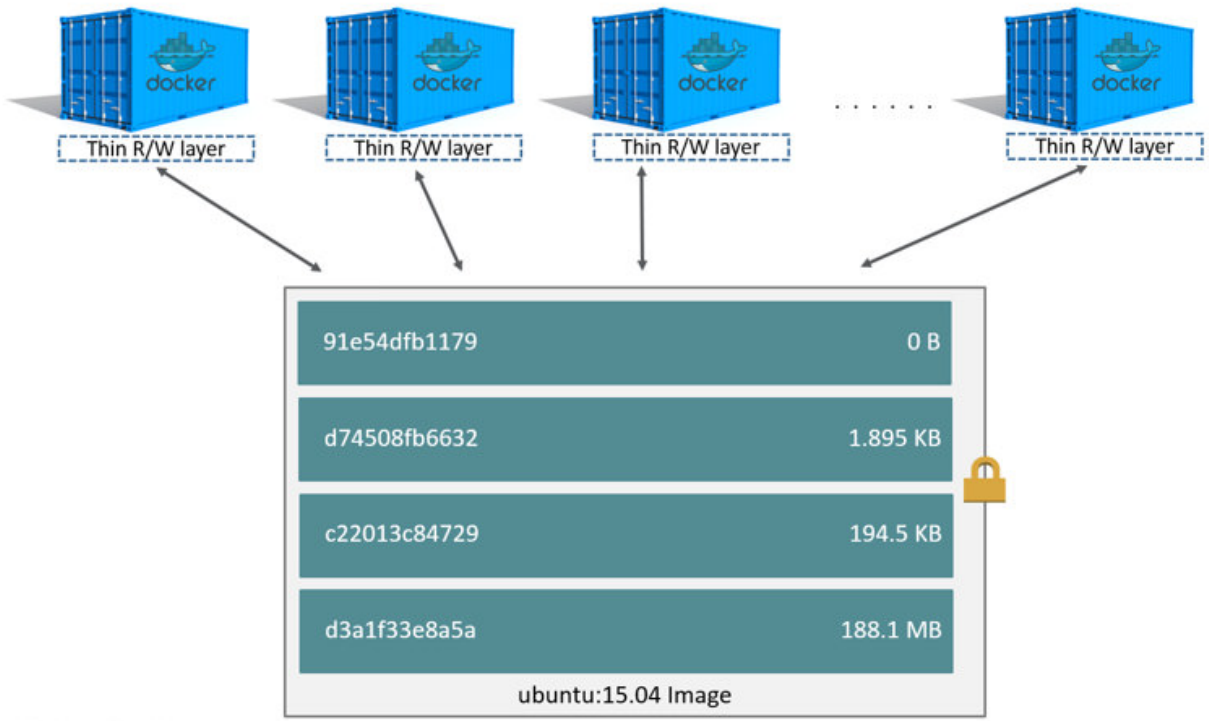
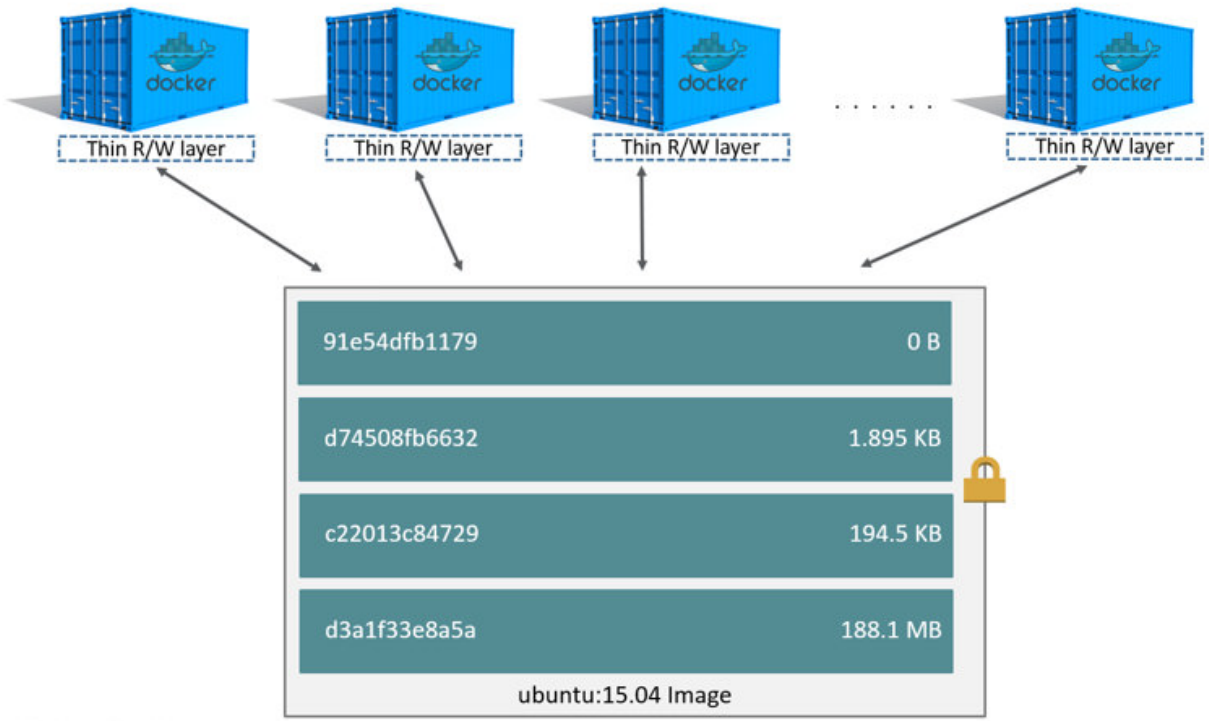


Image Layers



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

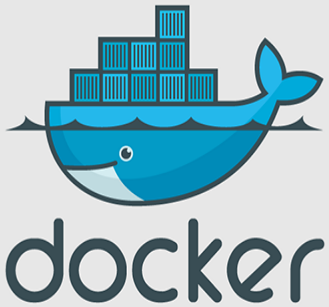
Dockerfile and Instructions

- 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



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.

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

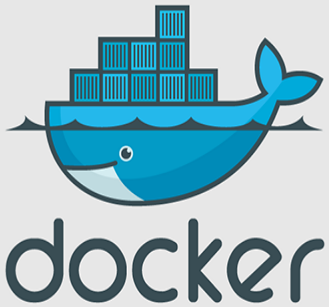
```
FROM ubuntu:14.04

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

Cache Busting

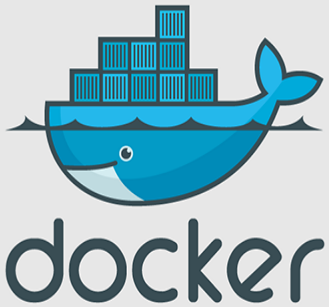
- You can also achieve cache-busting by specifying a package version. This is known as version pinning.

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

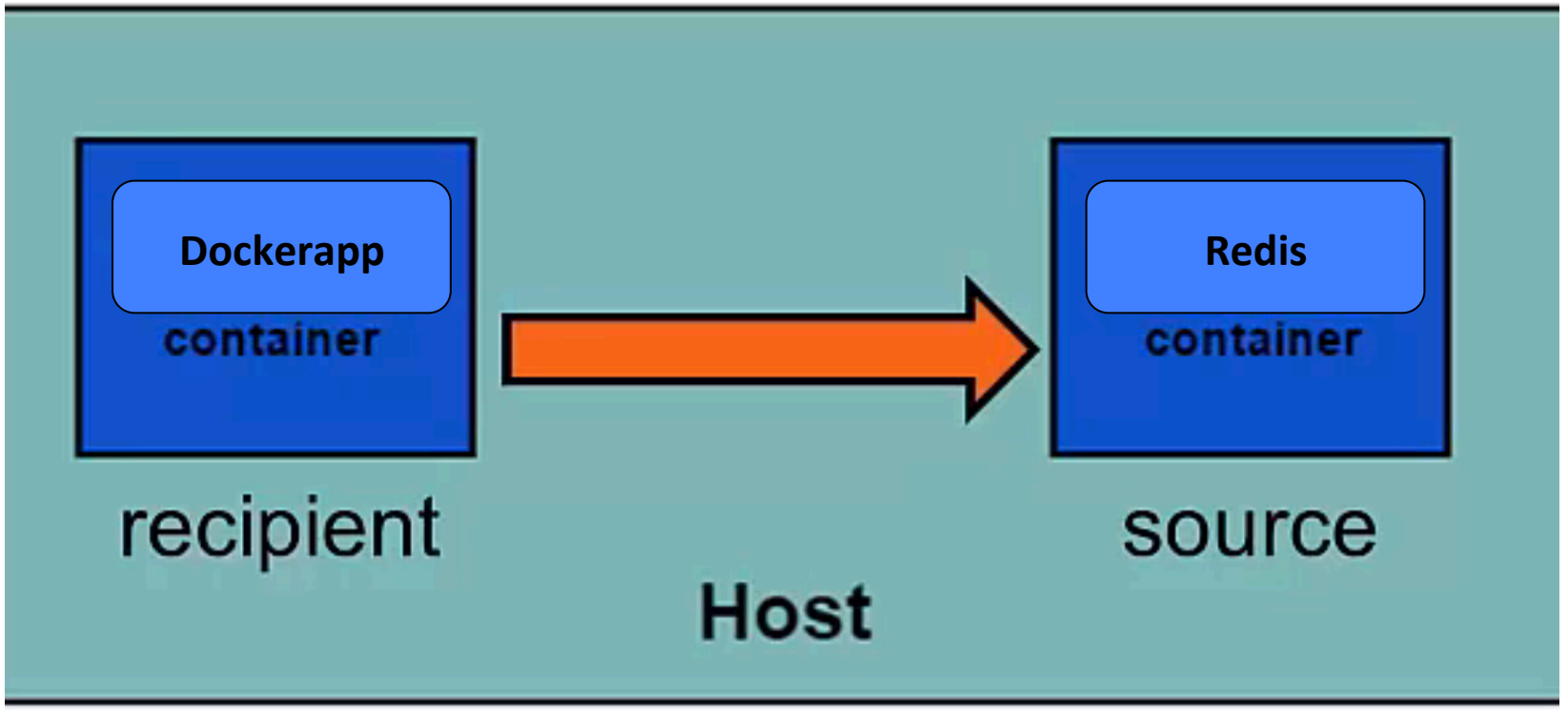
Dockerize a Hello World Web Application



Docker Container Links



Docker Container Links



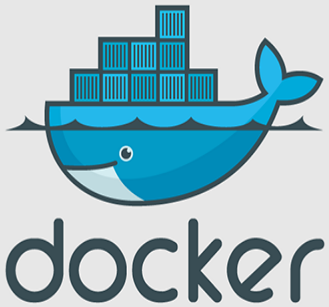
How container links work behind the scenes?

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.

Docker Compose



Why Docker Compose?

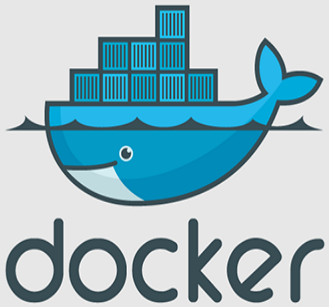
Manual linking containers and configuring services become impractical when the number of containers grows.

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.

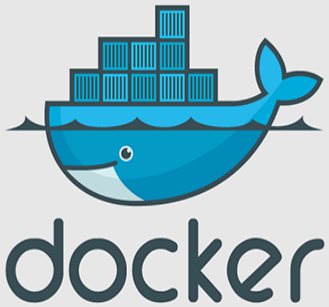
Deep Dive into
Docker Compose Workflow



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.**

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

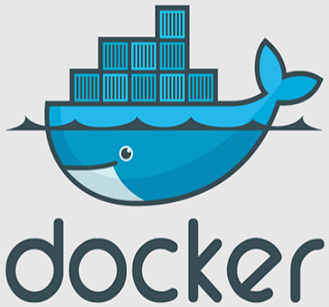
Pros:

- 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 into Continuous Integration(CI) Process

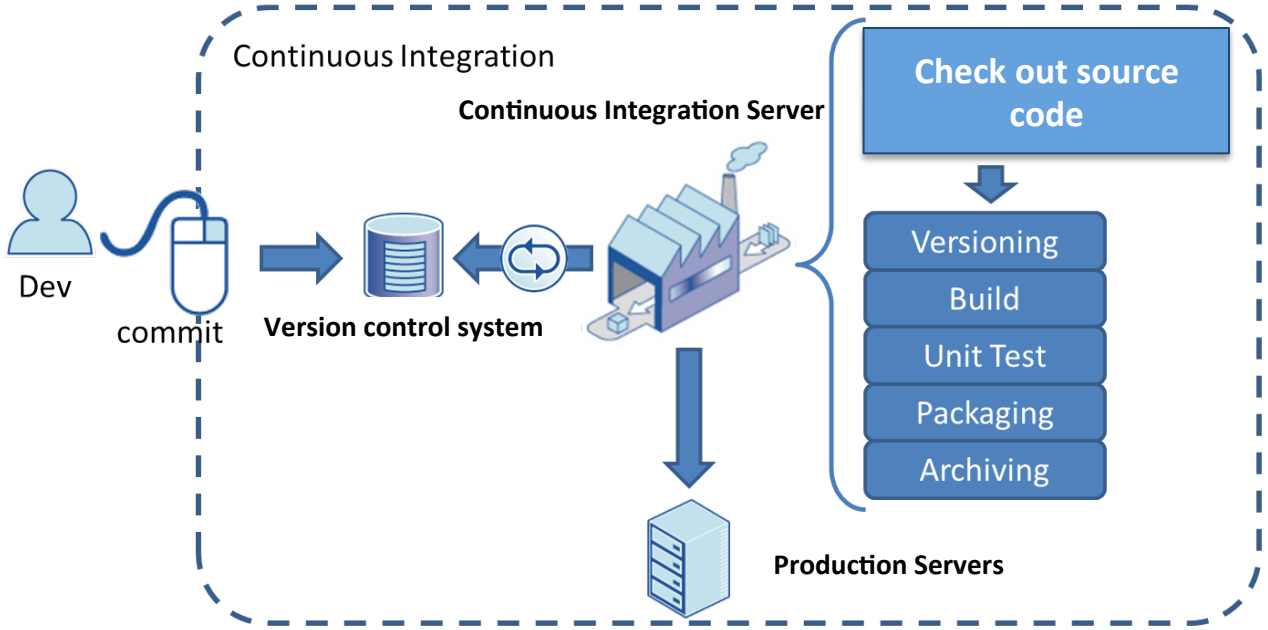


What is Continuous Integration?

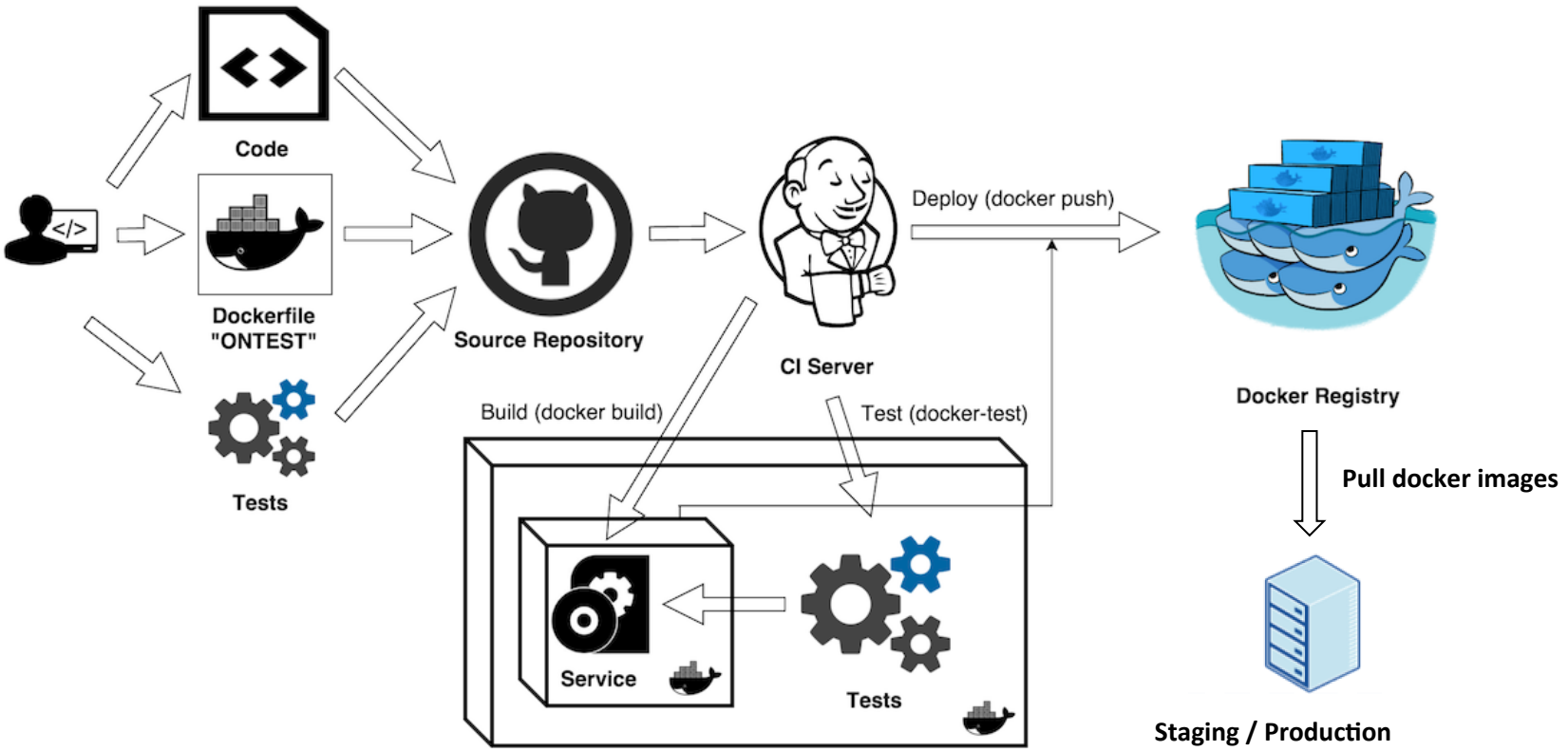
- Continuous integration is a software engineering practice in which isolated changes are immediately tested and reported when they are added to a larger code base.

- 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.

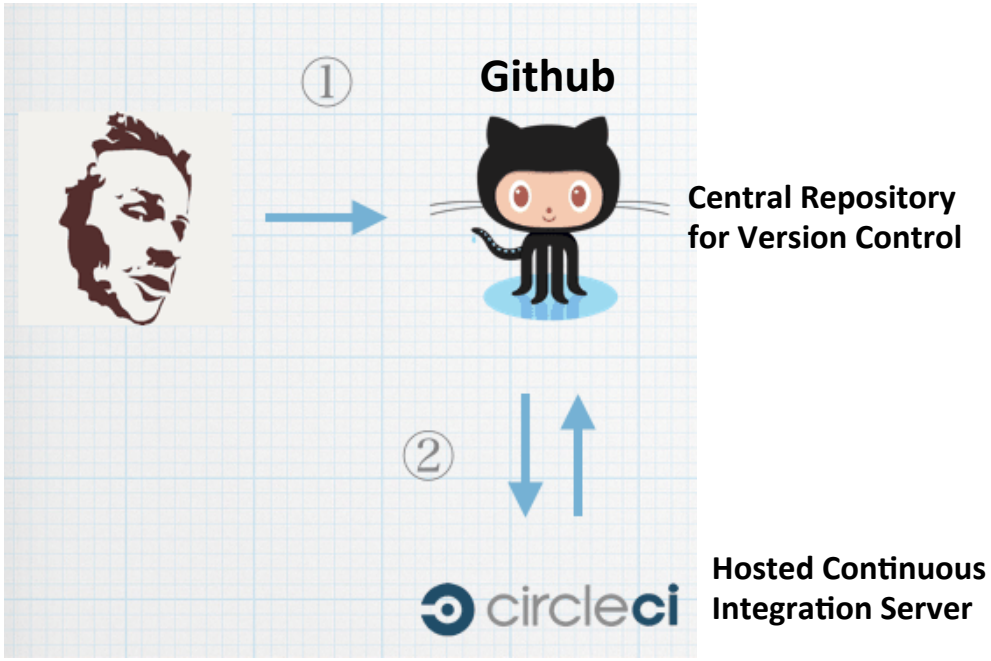
A Typical CI Pipeline without Docker



CI process with Docker involved



Our 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

to build a Continuous Integration pipeline

Text Direction: Introduction to Continuous Integration

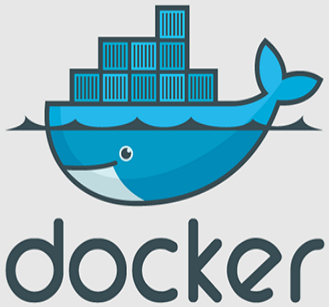
URL of the Github account to fork:
https://github.com/jleetutorial/dockerapp

Checking for existing SSH keys:
https://help.github.com/articles/checking-for-existing-ssh-keys/

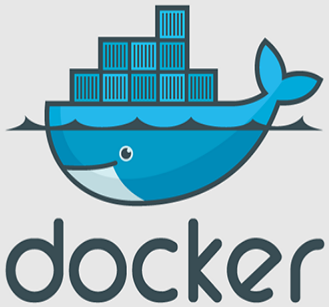
Generating a new SSH key and adding it to the ssh-agent:
https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Adding a new SSH key to your GitHub account:
https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/

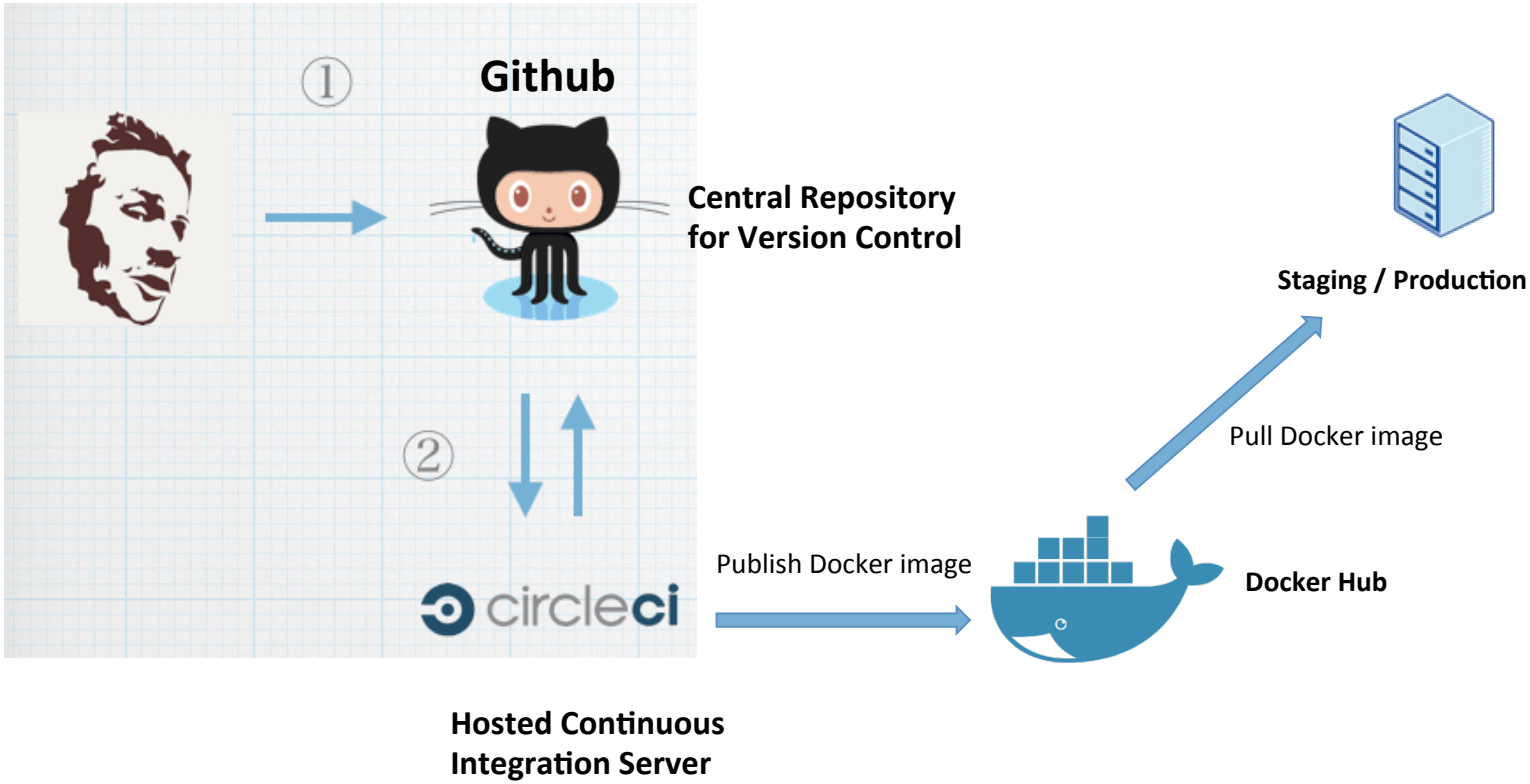
Link Circle CI with GitHub Account



Publish Docker Images from CircleCI



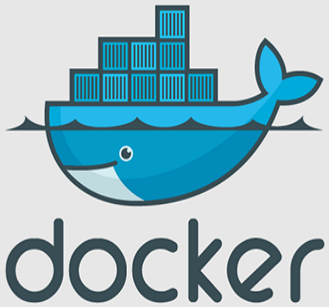
Complete CI Workflow



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

- One 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.

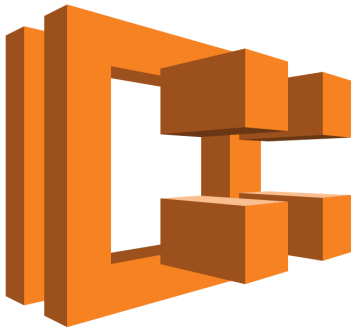
- The ecosystem of supporting Dockerized applications in production such as tools for monitoring and logging are still not fully ready yet.

Companies which already run Docker in Production

Why Running Docker Containers inside VMs?

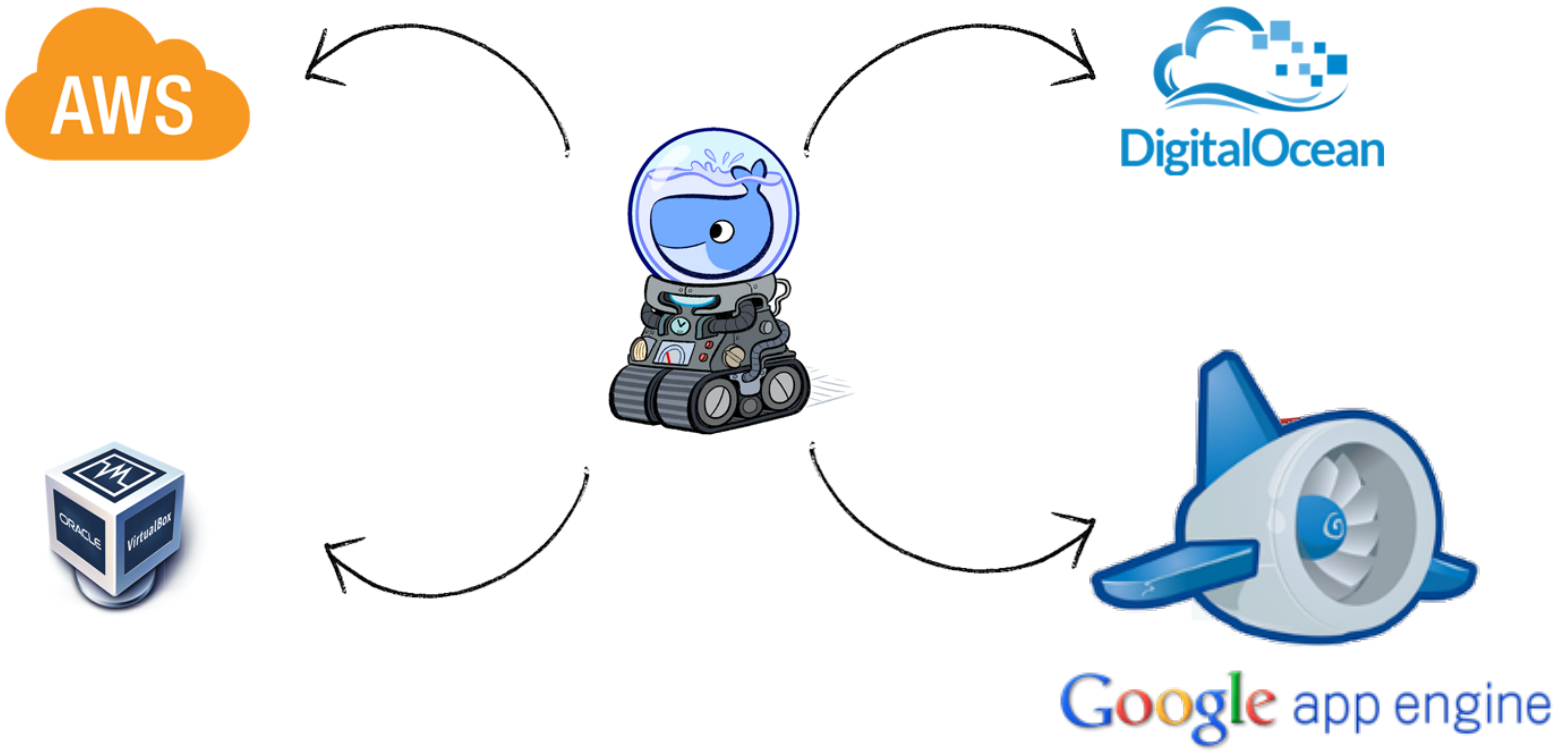
- To address security concerns.
- Hardware level isolation.

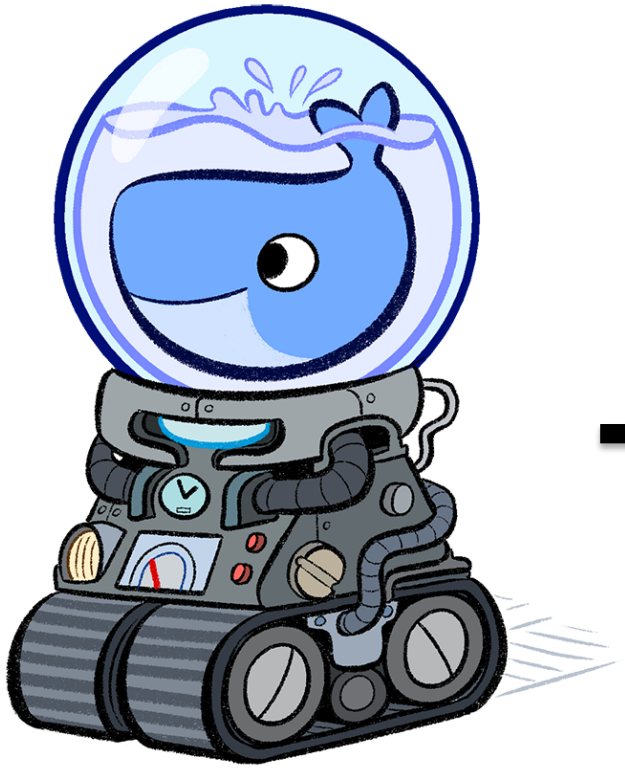
They all run containers inside VMs



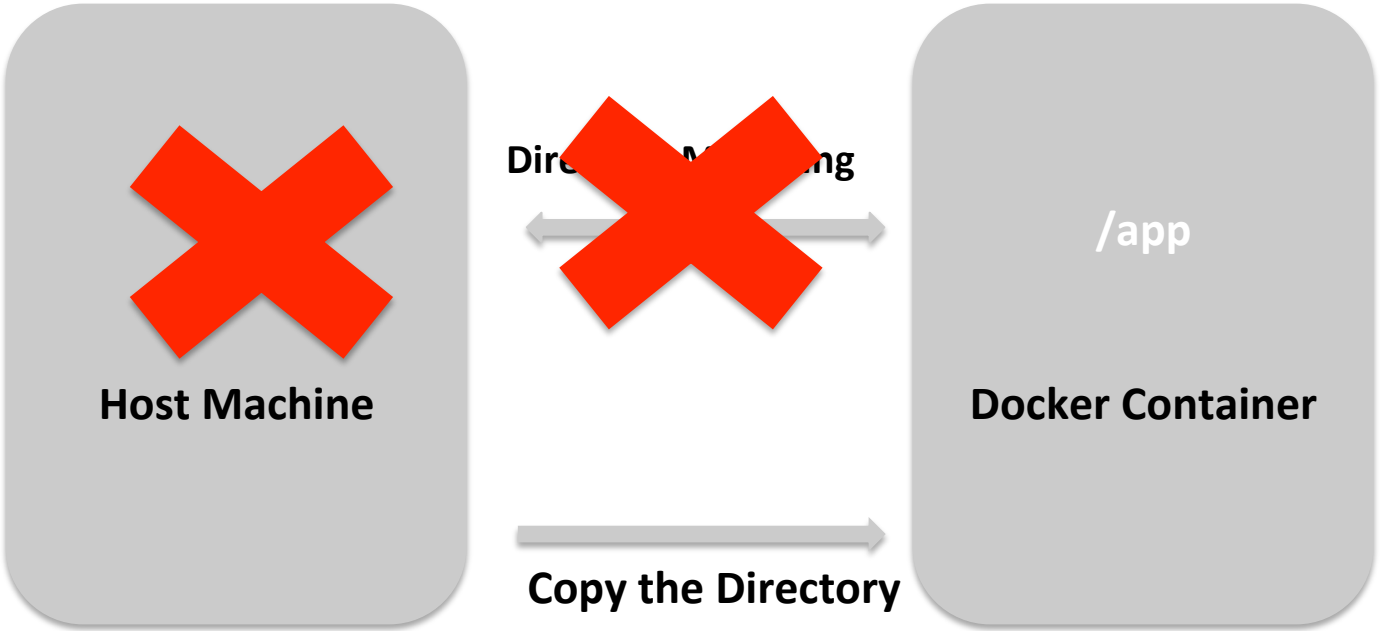
Docker Machine

Various Docker Machine Drivers

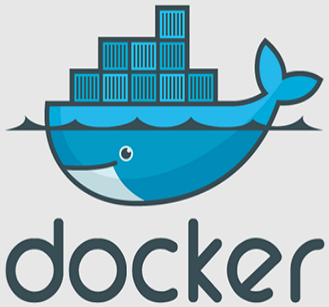




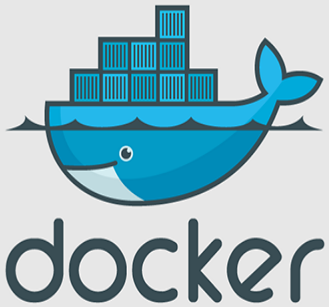
VirtualBox


Host Machine
Docker Container
/app
Copy the Directory

Register Digital Ocean Account to Deploy Dockerized Applications



Deploy Docker App to the Cloud with Docker Machine

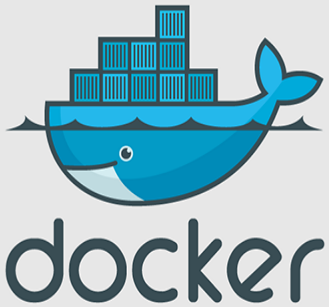


Text Direction: Deploy Docker App to the Cloud with Docker Machine

Docker Machine Create command

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

Refactor Docker Compose File

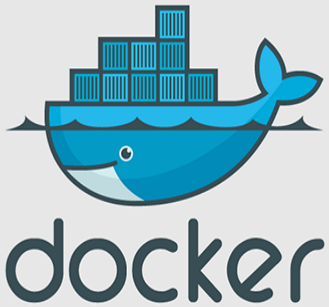


“Extends” keywords in Docker Compose File

- The extends keyword enables sharing of common configurations among different files or even different projects entirely.

- Extending services is useful if you have several different environments that reuse a common set of configuration options.

Introduction to Docker Swarm and Service Discovery



Text Direction: Introduction to Docker Swarm and Service Discovery

- **Step 1: Expose environment variables (If you are using Windows, replace "export" with "set" command)**

  export DIGITALOCEAN_ACCESS_TOKEN=<YOUR_DIGITALOCEAN_TOKEN>

  export DIGITALOCEAN_PRIVATE_NETWORKING=true

  export DIGITALOCEAN_IMAGE=debian-8-x64

- **Step 2: Provision consul machine**

  docker-machine create -d digitalocean consul

- **Step 3: Display the network configuration of the consul machine**

  docker-machine ssh consul ifconfig

- **Step 4: Ping the private and public IP address of the consul machine.**

  $(docker-machine ssh consul 'ifconfig eth0 | grep "inet addr:" | cut -d: -f2 | cut -d" " -f1')

  $(docker-machine ssh consul 'ifconfig eth1 | grep "inet addr:" | cut -d: -f2 | cut -d" " -f1')

- **Step 5: Export the private IP to KV_IP environment variable**

  export KV_IP=$(docker-machine ssh consul 'ifconfig eth1 | grep "inet addr:" | cut -d: -f2 | cut -d" " -f1')

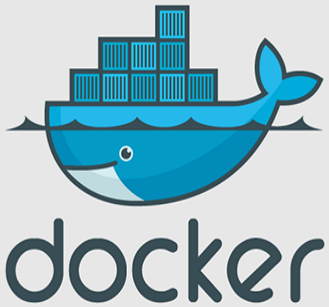
- **Step 6: Configure Docker client to connect to the consul machine**

  eval $(docker-machine env consul)

- **Step 7: Start the consul container in the consul machine**

  docker run -d -p ${KV_IP}:8500:8500 -h consul --restart always gliderlabs/consul-server -bootstrap

Deploy Docker App to the Cloud via Docker Swarm



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

- **Create Swarm master:**

```
docker-machine create -d digitalocean --swarm \
        --swarm-master \
        --swarm-discovery="consul://${KV_IP}:8500" \
        --engine-opt="cluster-store=consul://${KV_IP}:8500" \
        --engine-opt="cluster-advertise=eth1:2376" \
        master
```

- **Create Swarm slave:**

```
docker-machine create \
        -d digitalocean \
        --swarm \
        --swarm-discovery="consul://${KV_IP}:8500" \
         --engine-opt="cluster-store=consul://${KV_IP}:8500" \
        --engine-opt="cluster-advertise=eth1:2376" \
        slave
```