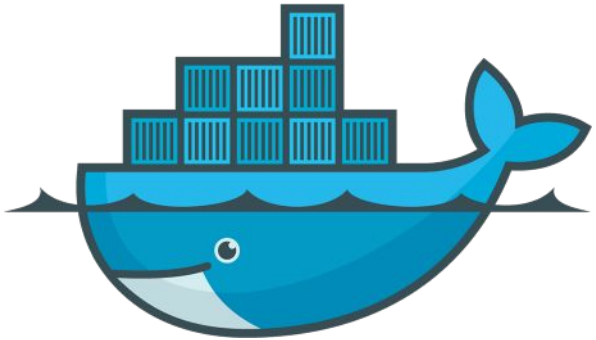


Docker Workshop

Chanwit Kaewkasi,
Docker Captain & Swarm Maintainer

As a Docker Swarm Maintainer

- Contributed to the project in December 2014
- Promoted to be a Maintainer in August 2015

Docker Swarm

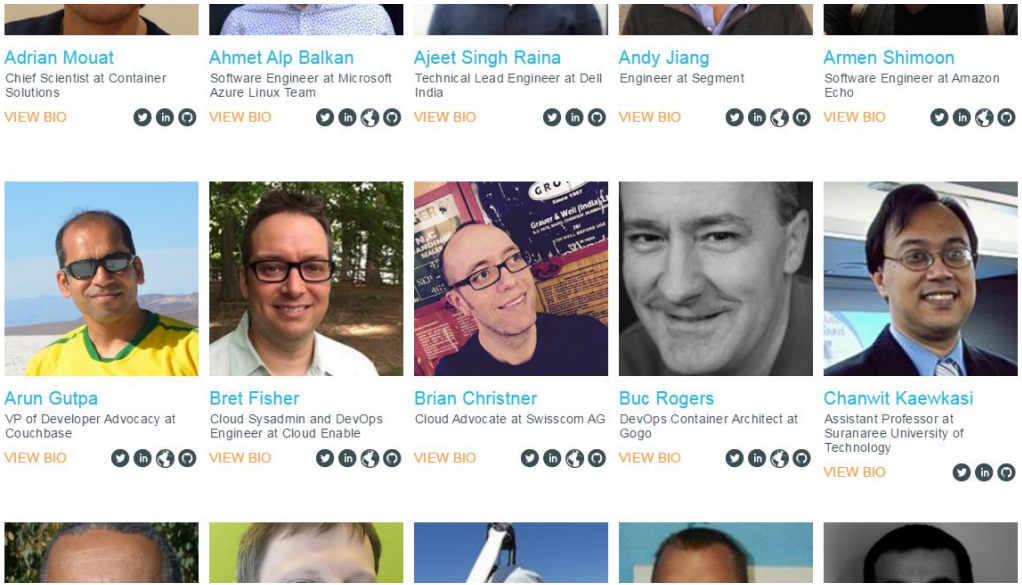
- พัฒนาระบบจัดการคลัสเตอร์สำหรับ Docker
- Scale ได้ถึงระดับ 1,000 โหนด

Docker Swarm

10+ Million Installations

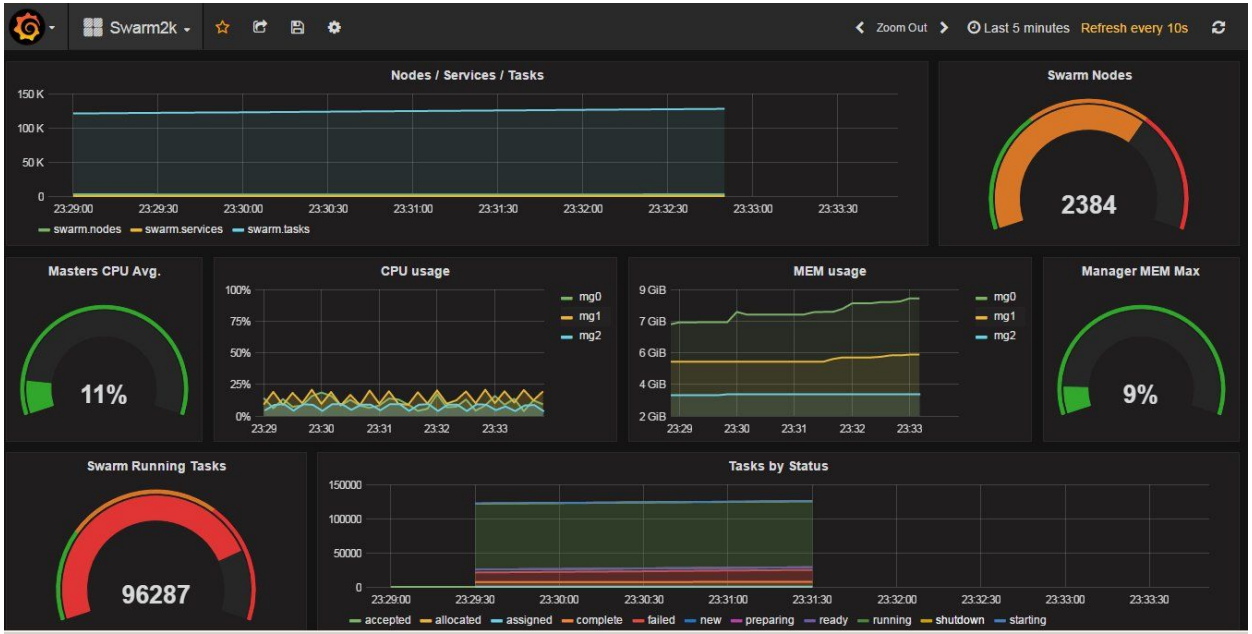
As a Docker Captain

- กลุ่มของผู้เชี่ยวชาญด้าน Docker จำนวน 46 คน แต่งตั้งโดย Docker inc.
- https://docker.com/captains



Project Swarm2K

- Swarm คลัสเตอร์ที่ใหญ่ที่สุดในโลก ขนาด 2,300 + node
- สร้างขึ้นจากความช่วยเหลือของสมาชิกใน Docker community จากทั่วโลก



Workshop Agenda

- Container concept reviews
- Docker for Developer
- Intermediate Concepts
- Advanced Concepts

Workshop Outcome

- Docker Concepts
- Docker Basics
- สร้าง / จัดการ Docker Cluster
- Docker Networking
- Multi-Cloud Cluster

What’s new in Docker 1.10 ?

- มี DNS ฝังใน libnetwork ใช้แทน /etc/hosts
- สนับสนุน --ip สำหรับระบุให้ container มี fixed IP ของตัวเอง
    - เช่น ใช้สร้าง Data Plane ที่ต้องระบุ IP

What’s new in Docker 1.11 ?

- ใช้ runC เป็น Runtime
  - สนับสนุน OCI standard
  - Runtime Portability

- DNS round-robin
  - ถ้า containers ใช้ alias เดียวกัน

What's new in Docker 1.12 ?

- Swarm mode
- Distributed Discovery model
- Routing Mesh load-balancer

แนะนำ Docker

คอนเทนเนอร์ ในตลาดนัด

simple tools, not big compl

IV

Define s interfaces f

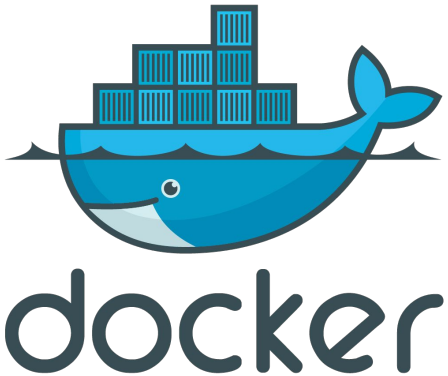
assemb r systems.

Software Container ?

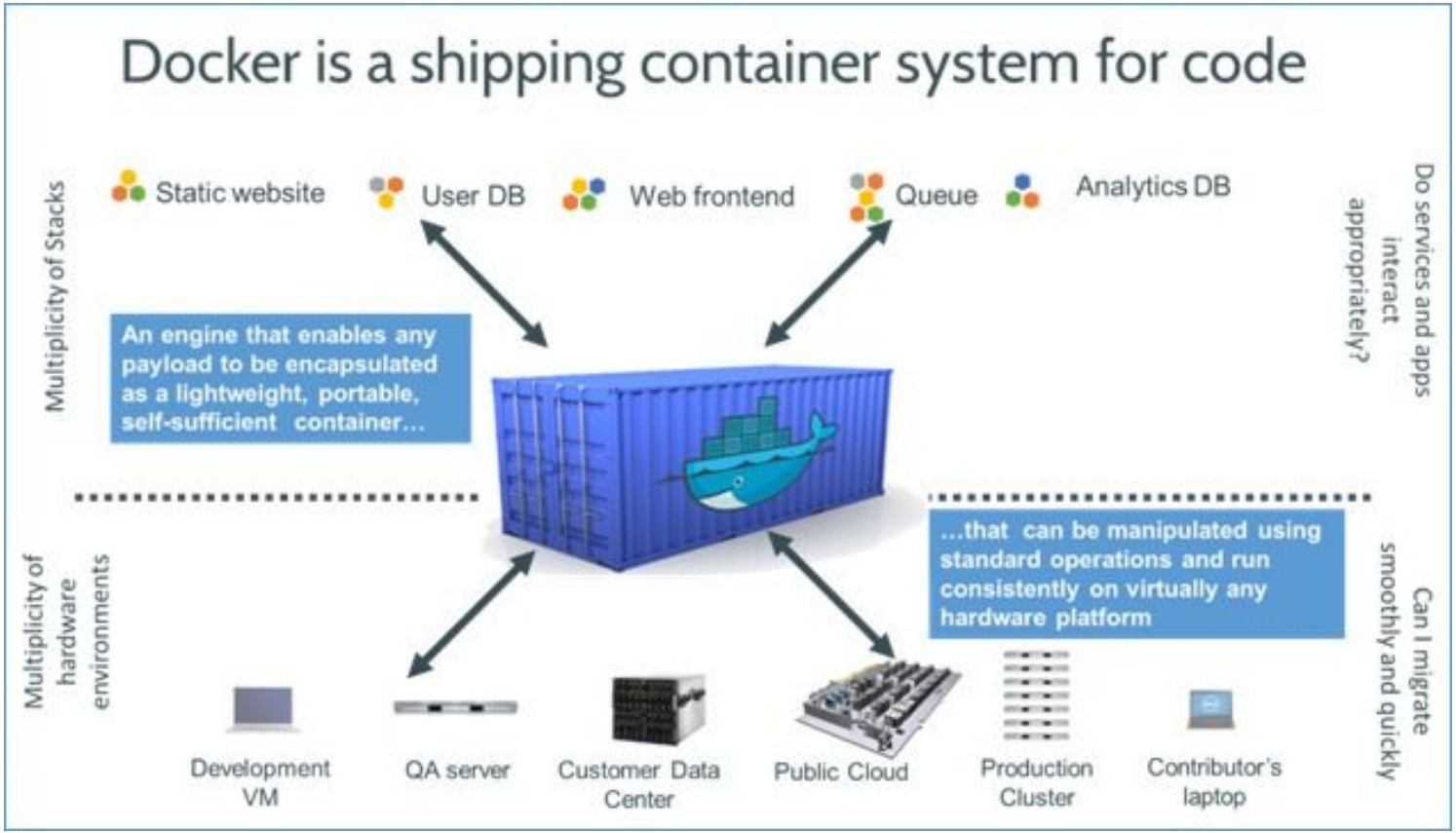
Software Container ?

- Solaris Zones
- LXC - Linux Containers

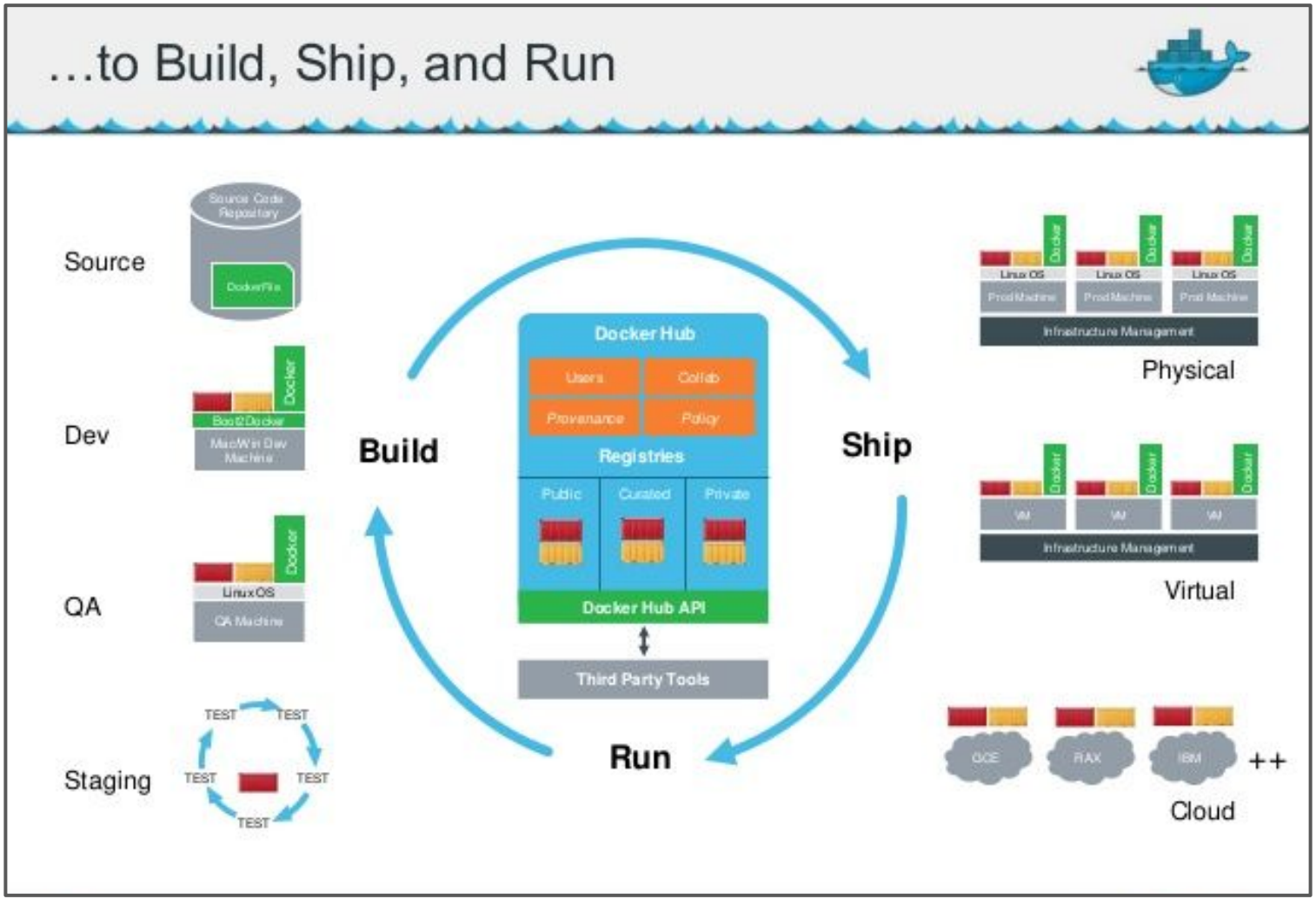
- Docker ทำให้ Container ใช้ง่ายขึ้น (Commodity)



Container และ Docker



…to Build, Ship, and Run

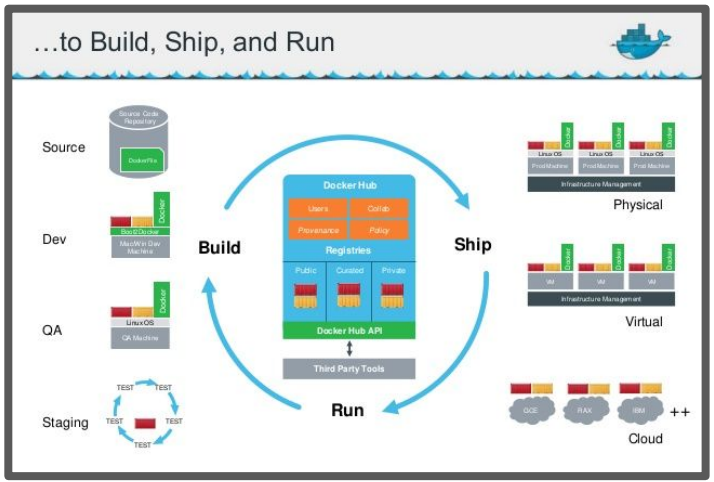


A sweet & colorful layered #Thai dessert is the analogy of @Docker images used in my training:

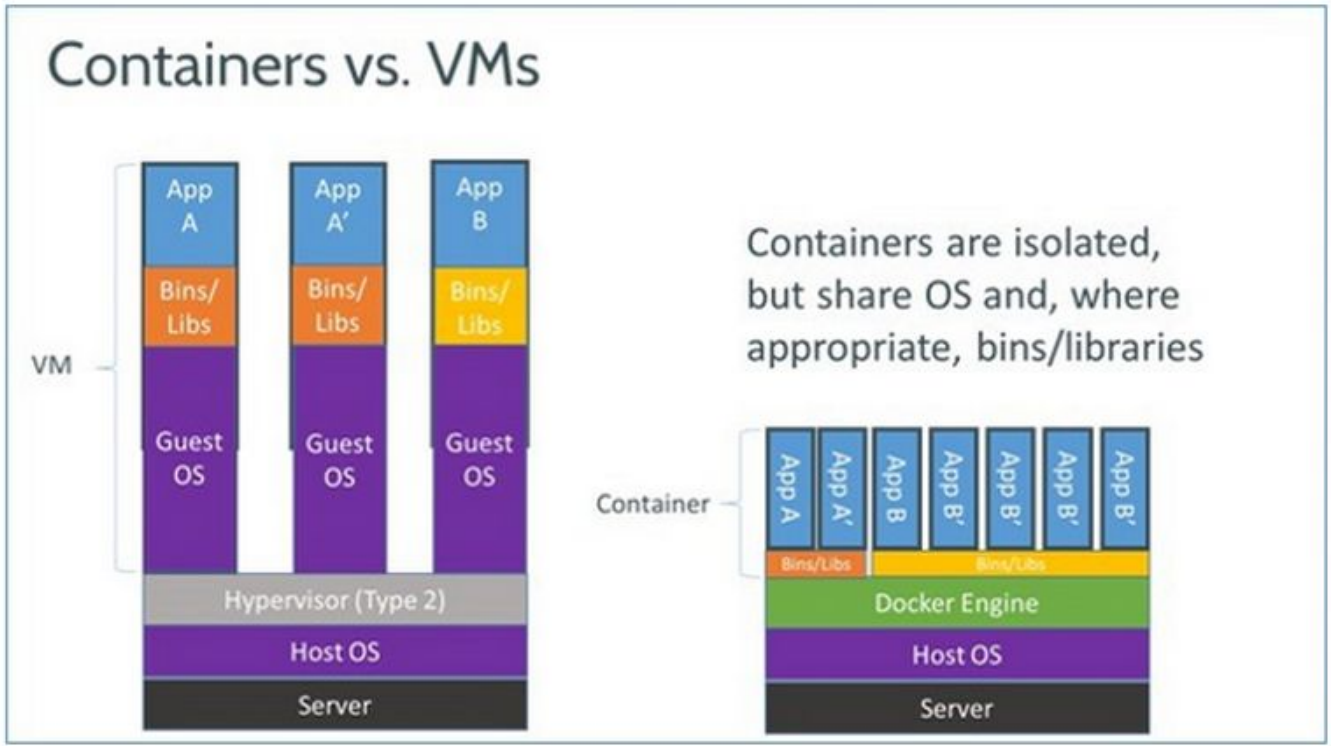
RETWEETS 4 LIKES 15

Tool that makes Container Commodity

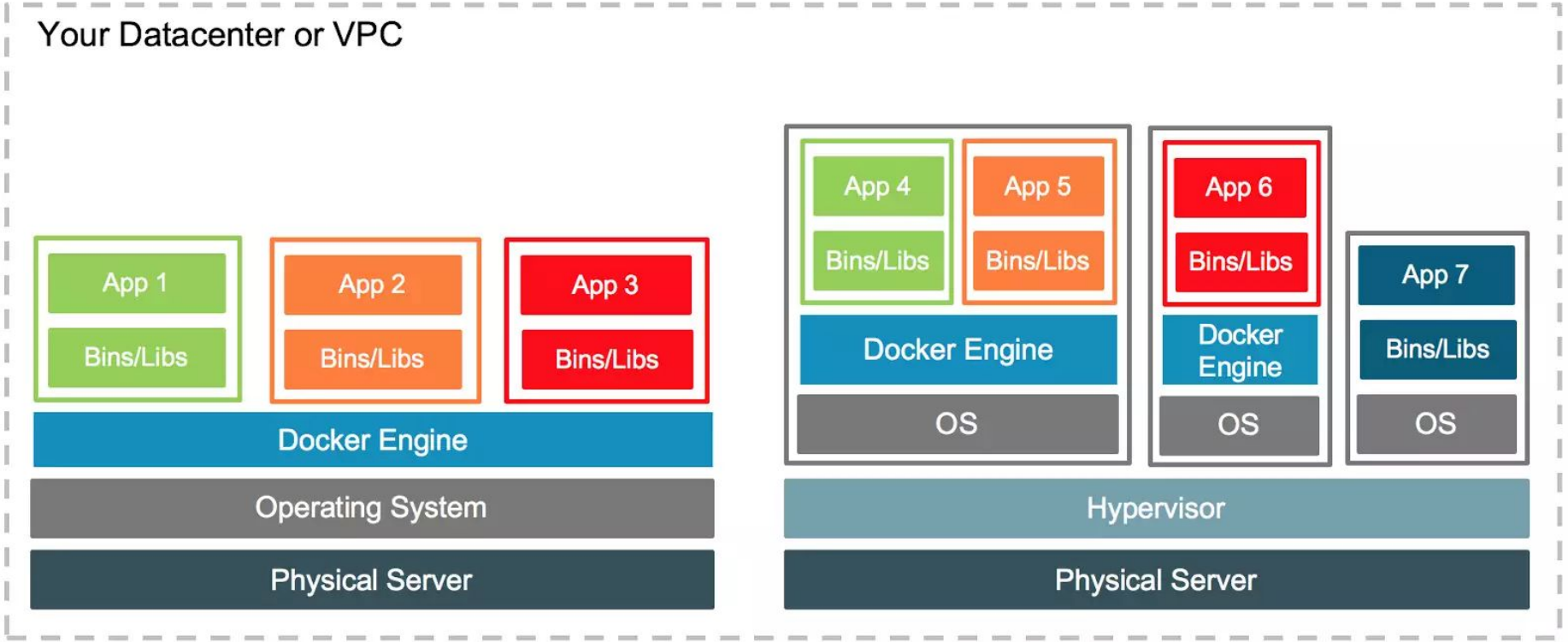
- **Commodity** ยังไง?
- **Build** - การสร้าง / ทำลาย Container ทำได้ง่ายขึ้นมาก
- **Ship** - การย้าย Container ทำได้ง่าย เพราะใช้ Union File System เข้ามาช่วย
  - AUFS
  - DeviceMapper Thin Provision
  - BTRFS
  - Overlay File System
- **Run** - การสั่งรัน Container สะดวก
  - $ docker run nginx



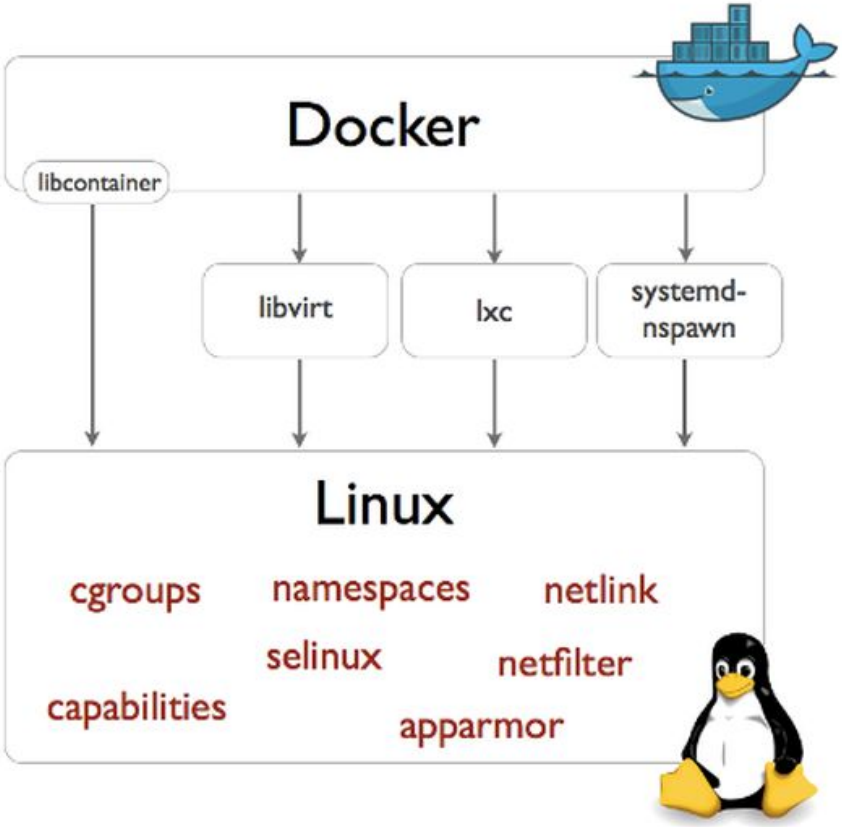
Containers เทียบกับ Virtual Machines



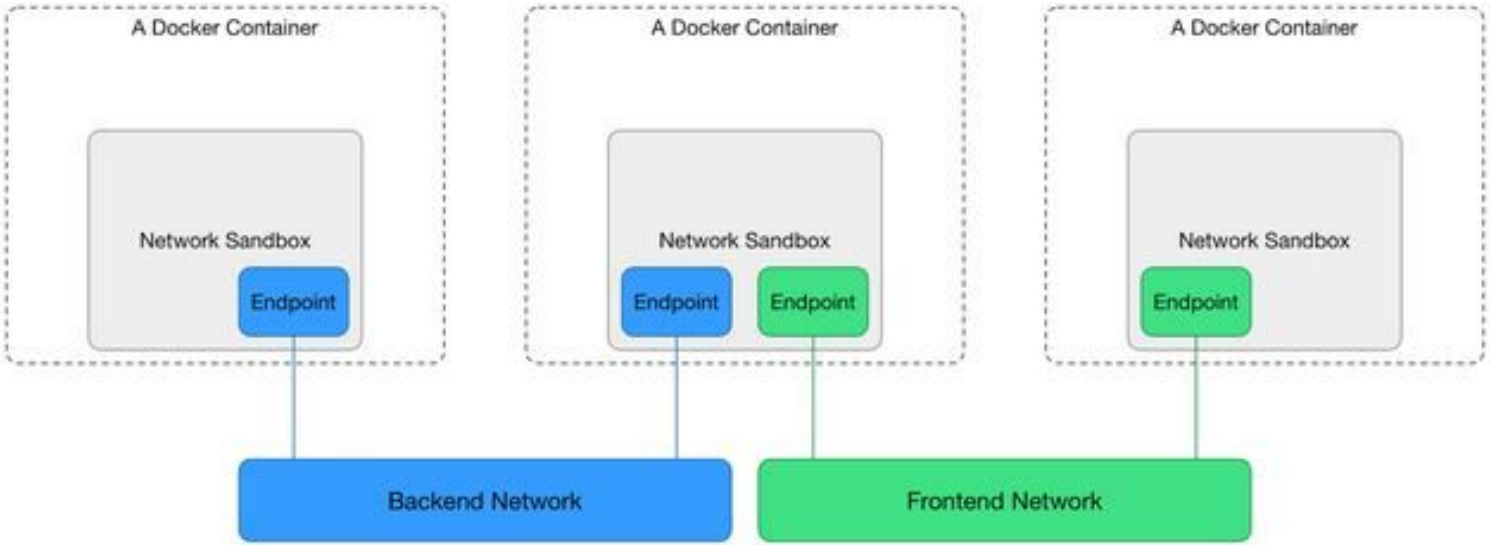
Container ใช้ร่วมกับ VM เป็นรูปแบบปกติ



libcontainer - Overhead ต่ำ



สนับสนุน Software Defined Network ระดับ Application



รูปแบบการ Deployment ระบบปี 2000 เทียบปี 2016

ปี 2000	ปี 2016
Deploy ไม่บ่อย 1 เดือนครั้ง, 3 เดือนครั้ง	Deploy วันละ 2 ครั้ง
Monolithic (ภาษาเดียว เฟรมเวิร์คเดียว)	Microservices (หลายภาษา หลายเฟรมเวิร์ค, เกาะกันหลวม ๆ)
Scaled Up (เครื่องใหญ่ เครื่องเดียว)	Scaled Out (หลายเครื่อง ข้ามคลาวด์ เช่า/ผสม)

ทำอย่างไร ระบบบน Production จึงจะไม่พัง?

- ไม่ install package
- ไม่ upgrade
- ไม่ remove หรือ downgrade
- ไม่ patch แม้แต่ security bug
- ไม่แก้ไฟล์ config

เป็นไปไม่ได้

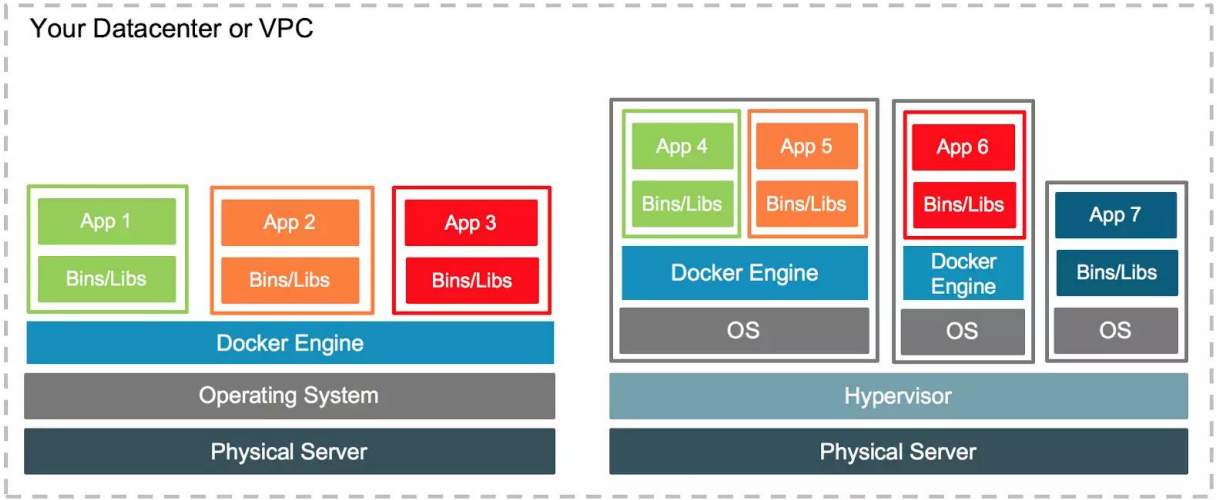
ทำอย่างไร ระบบบน Production จึงจะไม่พัง?

- ไม่ install package
- ไม่ upgrade
- ไม่ remove หรือ downgrade
- ไม่ patch แม้แต่ security bug
- ไม่แก้ไฟล์ config

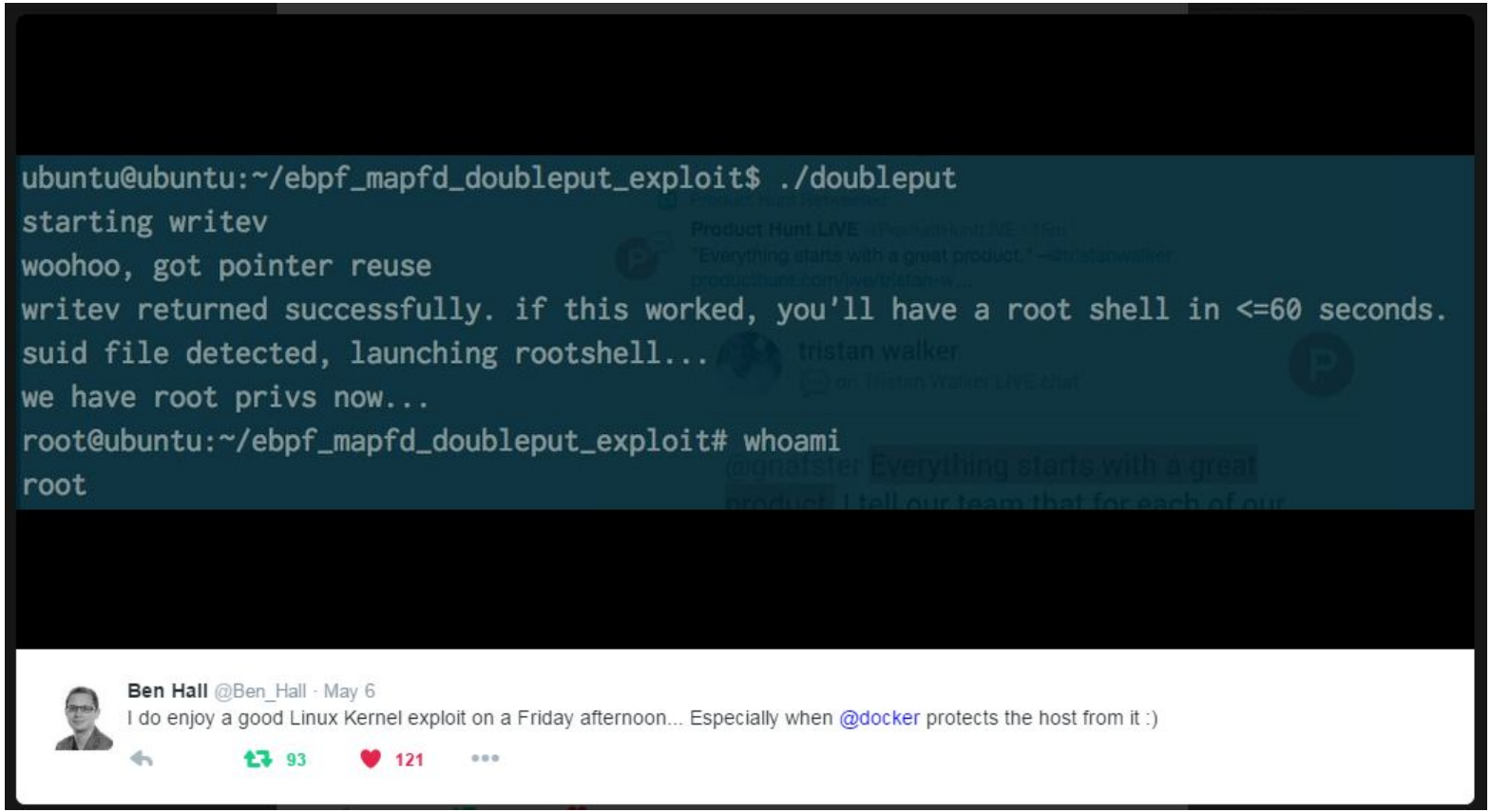
Software Container

ช่วยได้

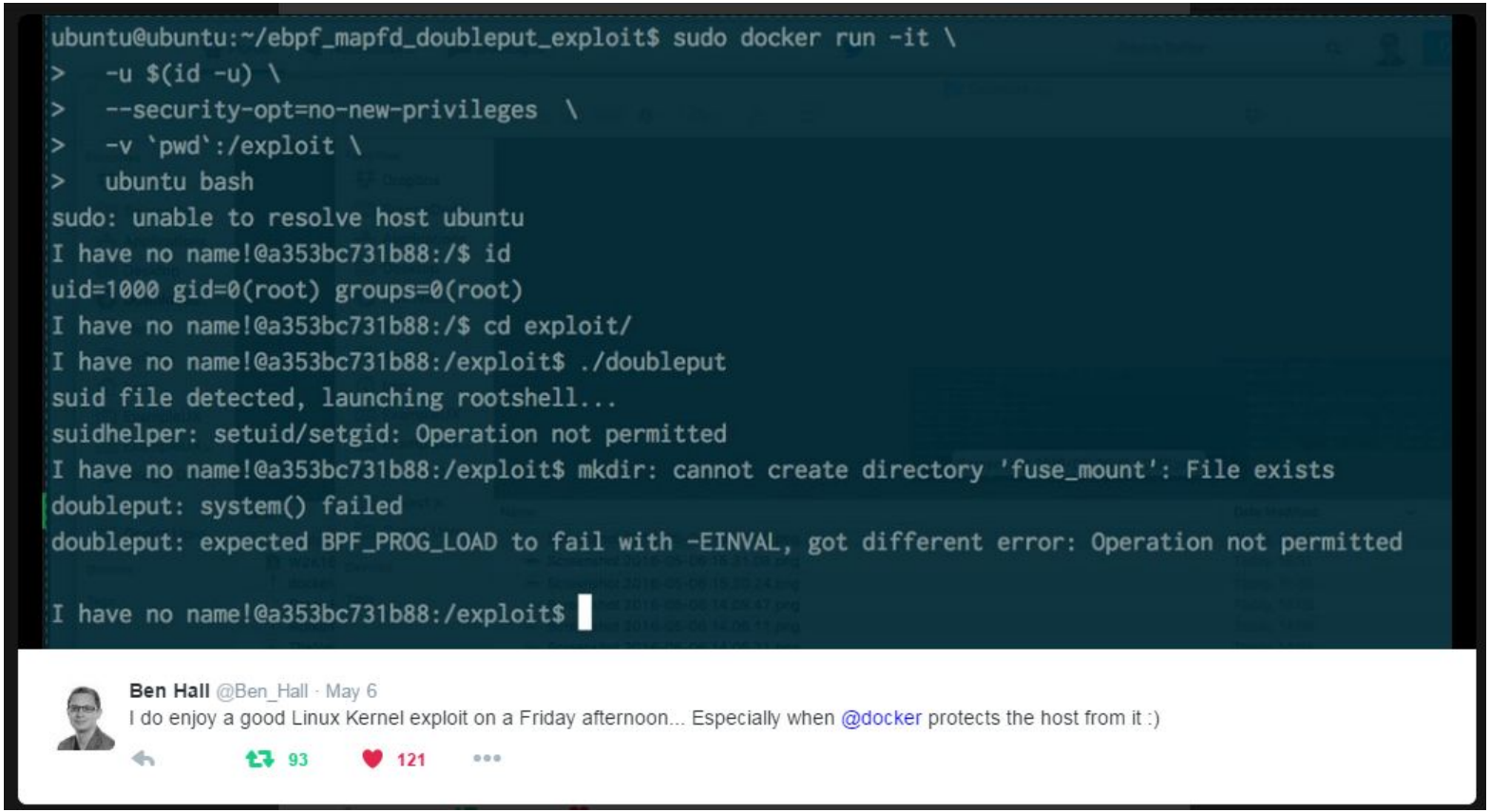
Container Isolation



Isolation ในระดับที่กันช่องโหว่ของ Kernel ได้



Isolation ในระดับที่กันช่องโหว่ของ Kernel ได้



Pet vs. Cattle

https://pixabay.com/en/kittens-cat-cat-puppy-rush-555822/

Pet vs. Cattle

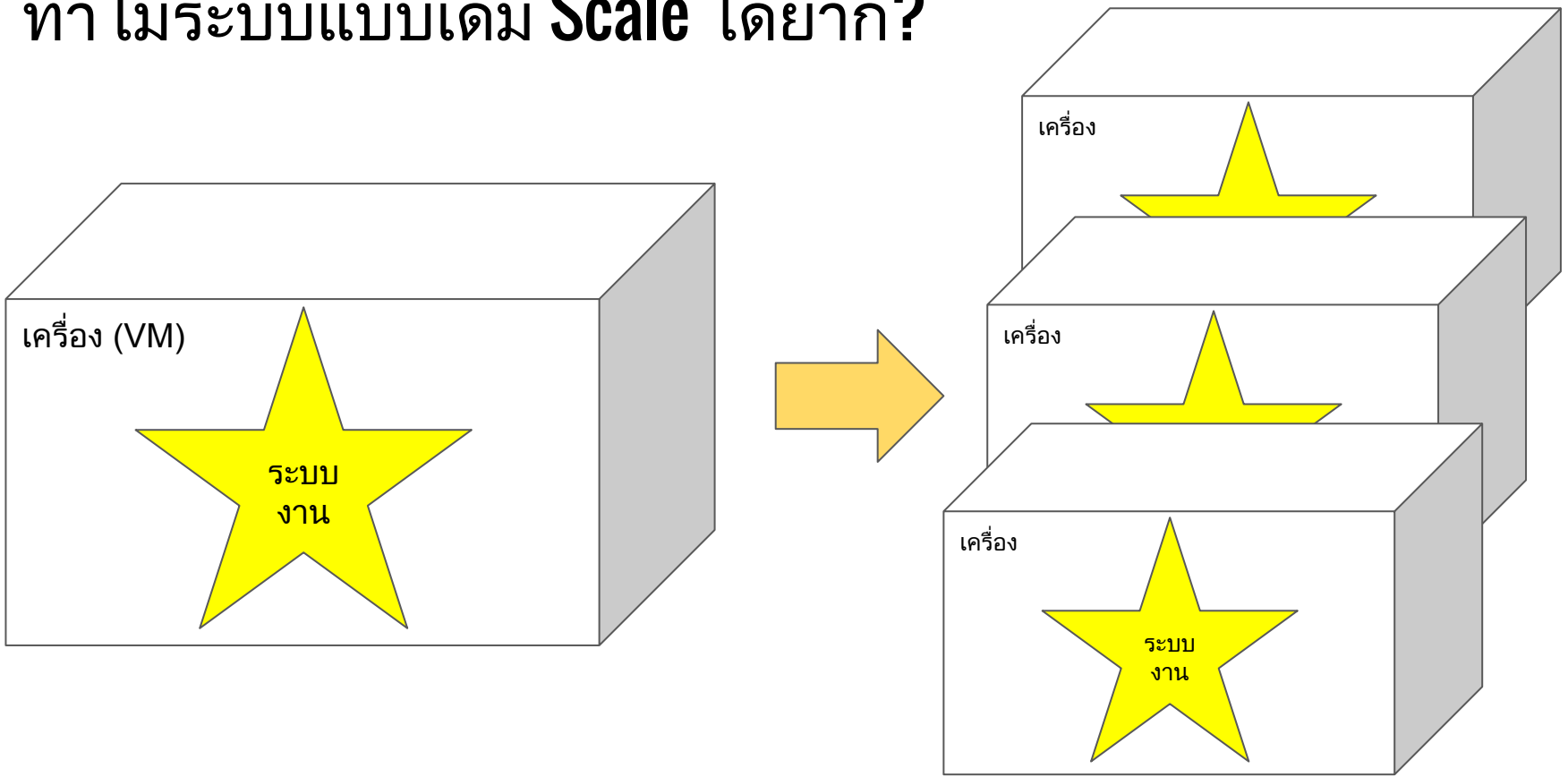
http://www.geograph.org.uk/photo/236877

ใช้ Cloud ให้เป็น Cloud

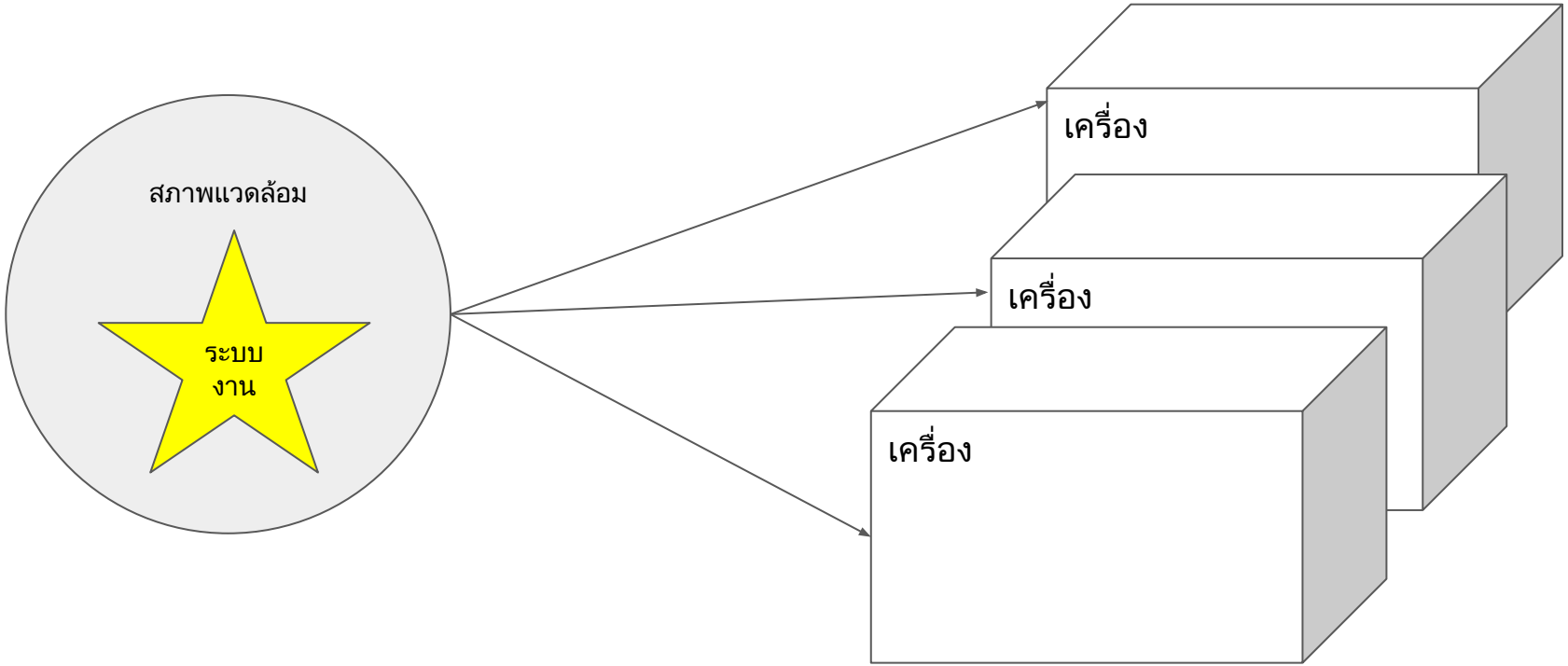
https://pixabay.com/en/photos/clouds/

ทำไมระบบแบบเดิม Scale ได้ยาก?

ทำไมระบบแบบเดิม Scale ได้ยาก?



แยก ระบบ (และสภาพแวดล้อม) ออกจาก เครื่อง



Container Isolation - เริ่มที่ Dev

SPEC.yml

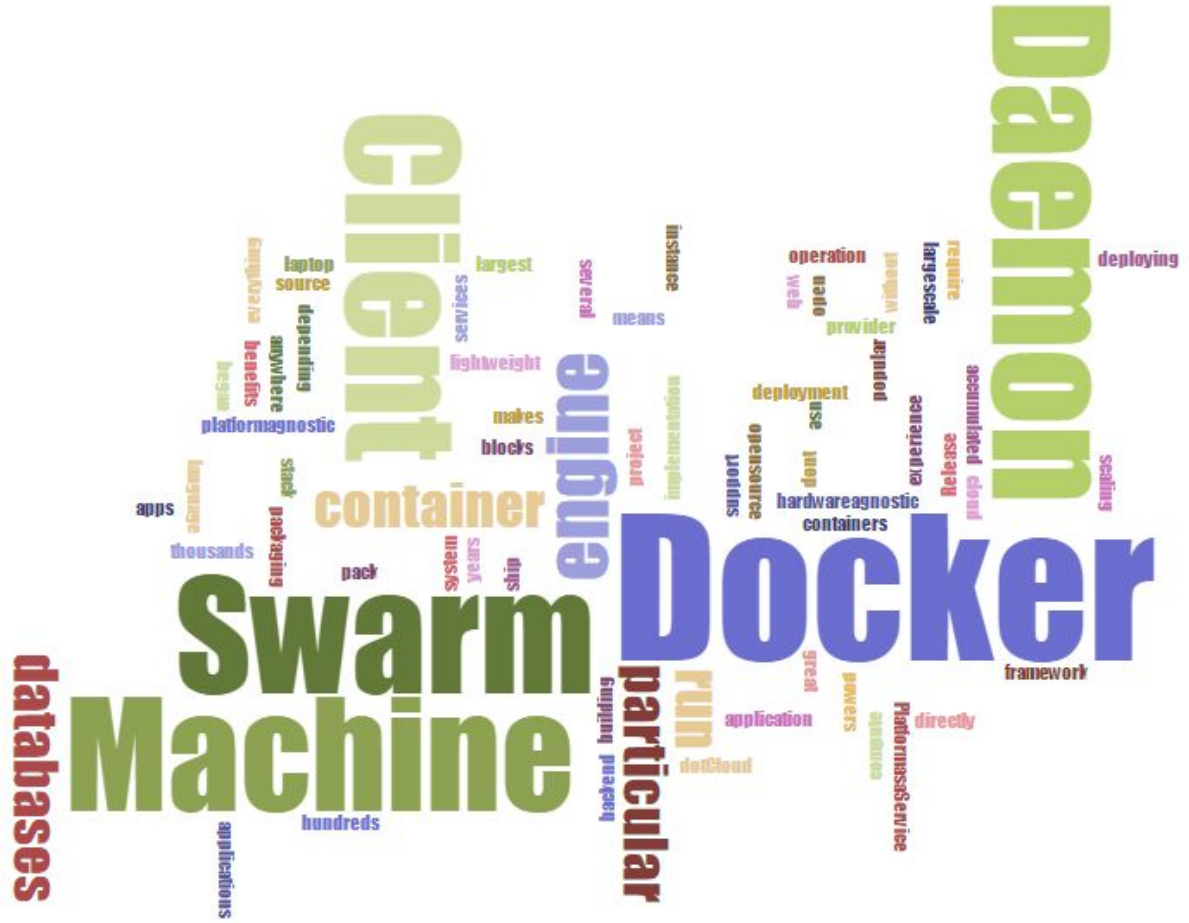
```
specVersion: 0.1.0
spec:
  name: base-cluster
  version: 0.1.0
  title: Base cluster package
  dependencies:
    consul-discovery: version=0.1.0
  provision: provision.yml
  composition: composition.yml
  description: >
    This package provides a Swarm master and a set of nodes
    that form cluster via the consul discovery service.
    An overlay network is pre-configured
    and available as the `multihost` network.
```

provision.yml

```
machines:
  ocean-master:
    driver: digitalocean
    export: true
    options:
      digitalocean-image: debian-8-x64
      digitalocean-region: sgp1
      engine-opt:
        cluster-store: consul://${consul}:8500
        cluster-advertise: eth0:2376
      swarm: true
      swarm-master: true
      swarm-discovery: consul://${consul}:8500/discovery
    post-provision:
      - docker network create --driver overlay multihost

  ocean:
    driver: digitalocean
    instances: 2
    options:
      digitalocean-image: debian-8-x64
      engine-opt:
        cluster-store: consul://${consul}:8500
        cluster-advertise: eth0:2376
      swarm: true
      swarm-discovery: consul://${consul}:8500/discovery
```

Glossary

- Docker
- Docker Engine
- Docker Client
- Docker Daemon
- Docker Cluster
- Docker Containers
- Docker Images
- Docker Registry (Hub)



ติดตั้ง Docker


docker

Docker for Mac & Windows

- วิธีการที่ง่ายที่สุดสำหรับเริ่มใช้ Docker
  - บน OS ที่ไม่ใช่ Linux

- https://docs.docker.com/engine/installation/windows/
- https://docs.docker.com/engine/installation/mac/

บน Linux - Docker Installation

```
#!/bin/sh
set -e
#
# This script is meant for quick & easy install via:
#   'curl -sSL https://get.docker.com/ | sh'
# or:
#   'wget -qO- https://get.docker.com/ | sh'
#
# For test builds (ie. release candidates):
#   'curl -sSL https://test.docker.com/ | sh'
# or:
#   'wget -qO- https://test.docker.com/ | sh'
#
# For experimental builds:
#   'curl -sSL https://experimental.docker.com/ | sh'
# or:
#   'wget -qO- https://experimental.docker.com/ | sh'
#
# Docker Maintainers:
#   To update this script on https://get.docker.com,
#   use hack/release.sh during a normal release,
#   or the following one-liner for script hotfixes:
#     s3cmd put --acl-public -P hack/install.sh s3://get.docker.com/index
#
```

Test and Experimental Channels

- Test Channel

https://test.docker.com

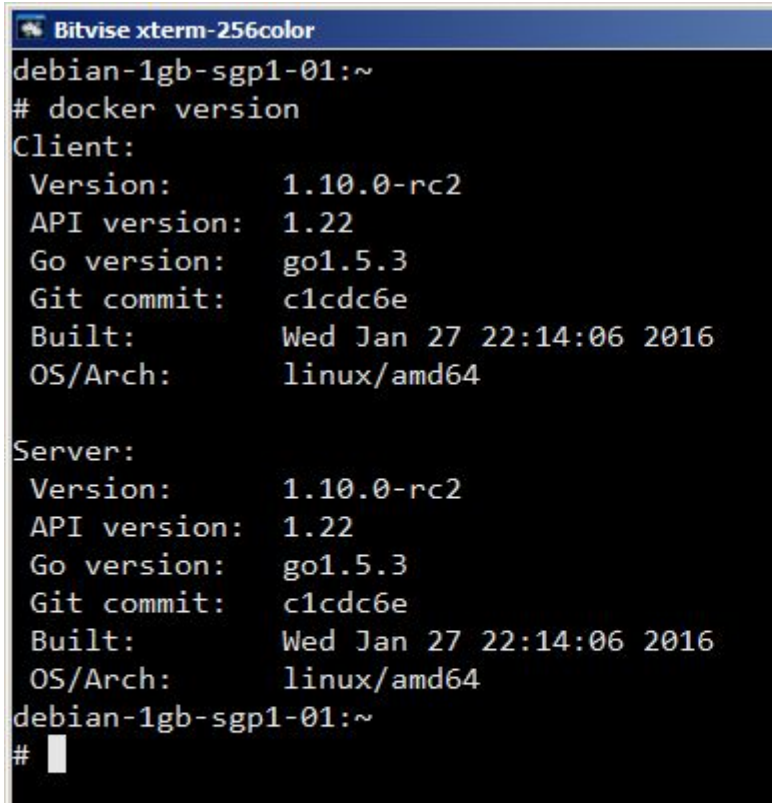
- Experimental Channel

https://experimental.docker.com

Software in Docker Toolbox

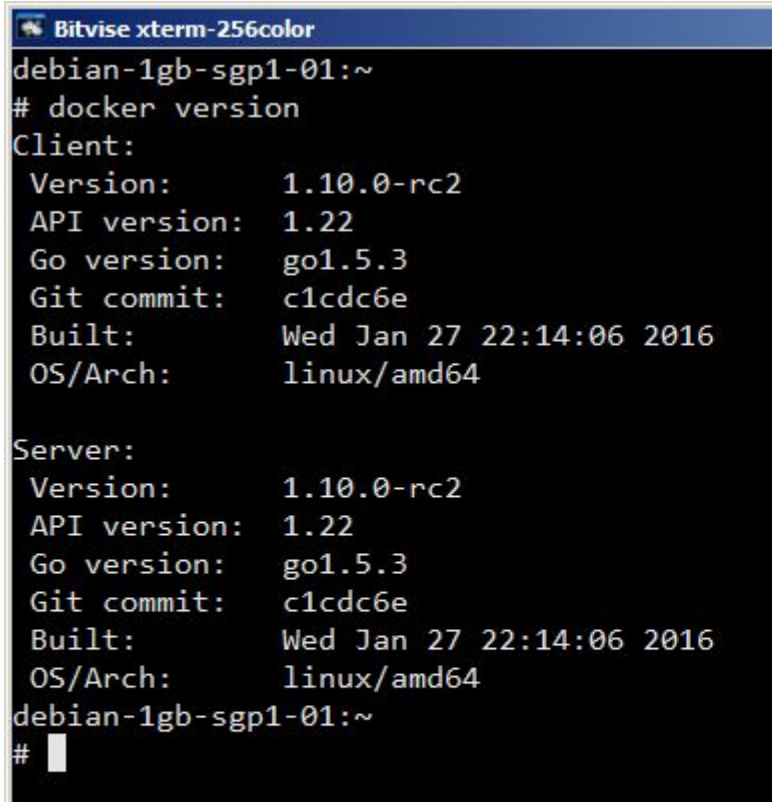
- Docker Machine
- Docker Compose
- Docker Client
- VirtualBox
- Kitematic

Docker - Check Version



Command Line

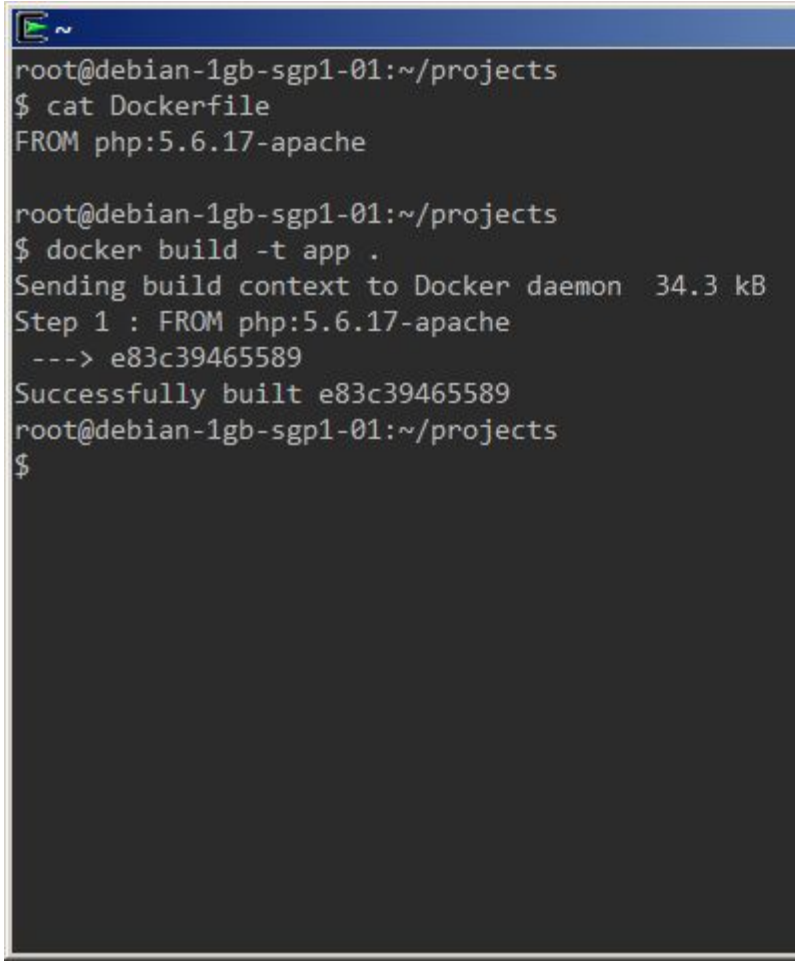
$ docker version



การเขียน Dockerfile

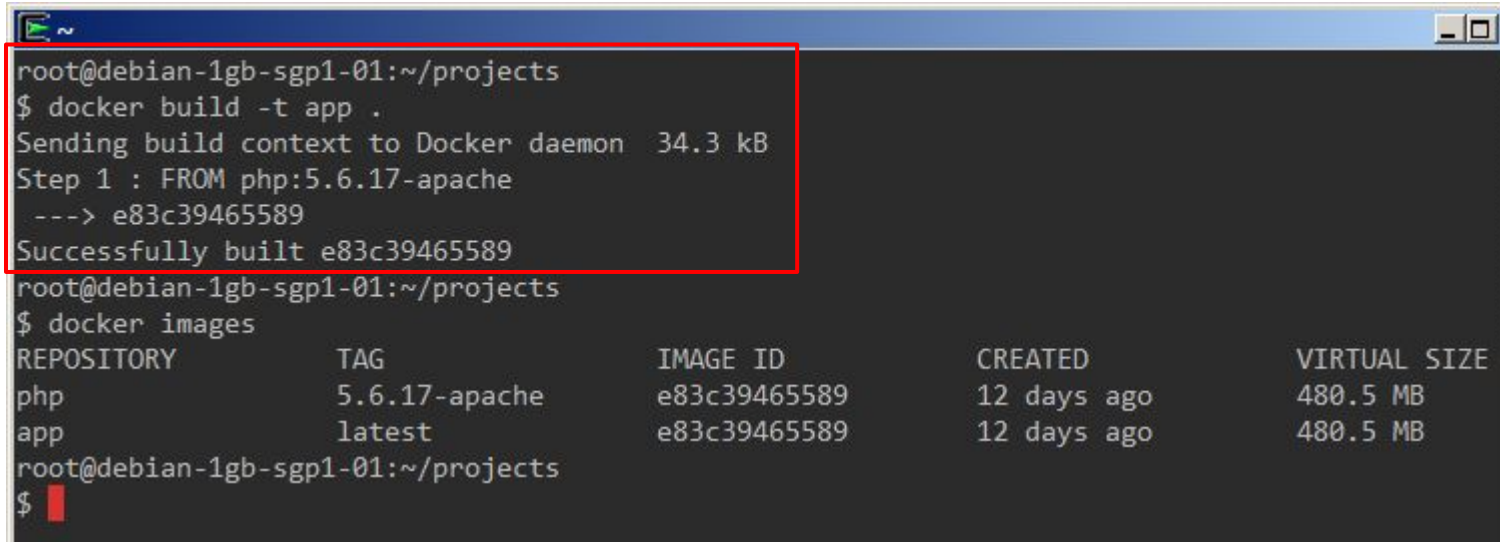
Dockerfile: FROM

FROM php:5.6.17-apache



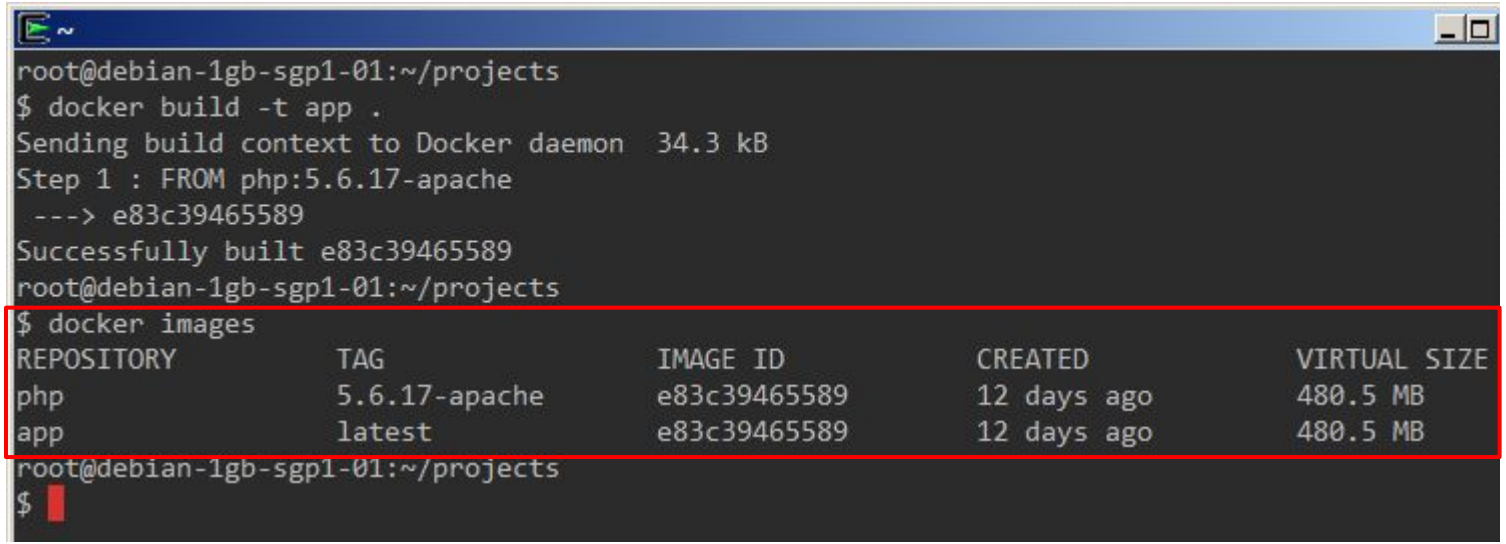
Command “build” & “images”

$ docker build -t app .



Command “build” & “images”

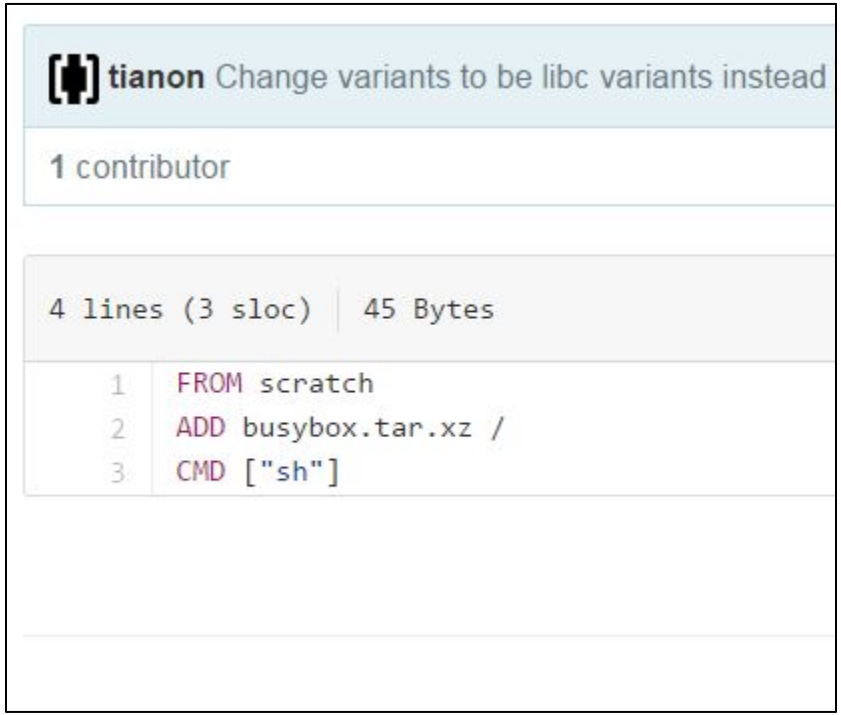
$ docker images



Dockerfile Instructions

- FROM
- MAINTAINER
- ADD
- COPY
- RUN
- ENTRYPOINT
- CMD

Dockerfile Example: busybox glibc



Dockerfile Example: php 5.5

68 lines (58 sloc) | 2.38 KB

Raw | Blame | History

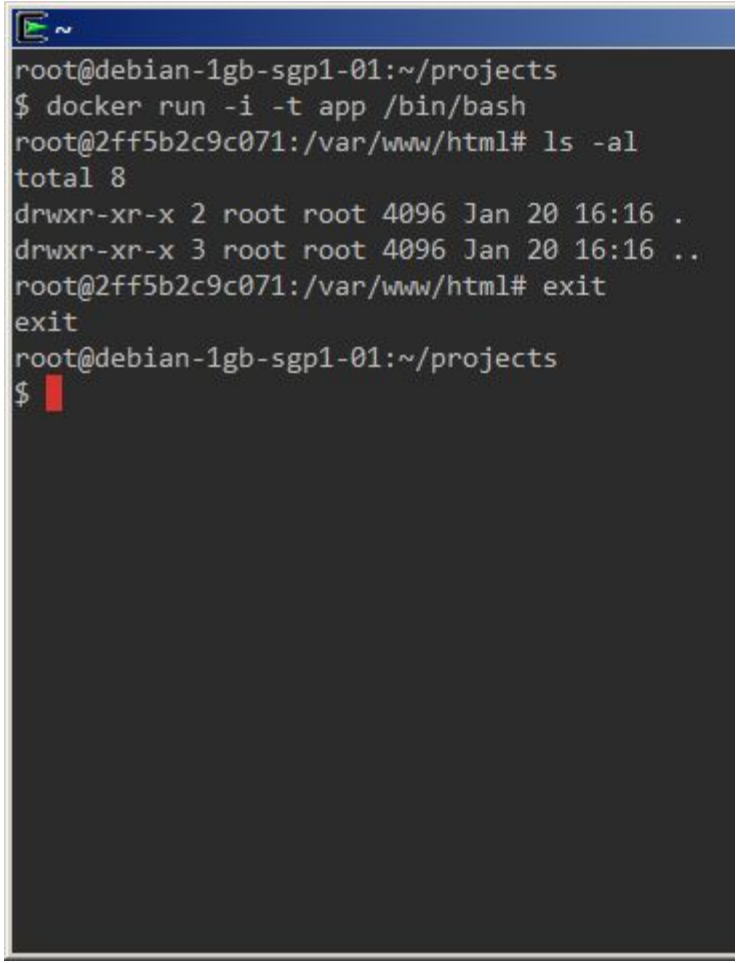
```
1  FROM debian:jessie
2
3  # persistent / runtime deps
4  RUN apt-get update && apt-get install -y ca-certificates curl librecode0 libsqlite3-0 libxml2 --no-install-recommends && rm -r /var/lib/apt
5
6  # phpize deps
7  RUN apt-get update && apt-get install -y autoconf file g++ gcc libc-dev make pkg-config re2c --no-install-recommends && rm -r /var/lib/apt/
8
9  ENV PHP_INI_DIR /usr/local/etc/php
10 RUN mkdir -p $PHP_INI_DIR/conf.d
11
12 ##<autogenerated>##
13 ##</autogenerated>##
14
15 ENV GPG_KEYS 0B96609E270F565C13292B24C13C70B87267B52D 0BD78B5F97500D450838F95DFE857D9A90D90EC1 F38252826ACD957EF380D39F2F7956BC5DA04B5D
16 RUN set -xe \
17         && for key in $GPG_KEYS; do \
18                 gpg --keyserver ha.pool.sks-keyservers.net --recv-keys "$key"; \
19         done
20
21 ENV PHP_VERSION 5.5.31
```

Inside our “app” container

$ docker run -i -t app /bin/bash

ls -al

exit



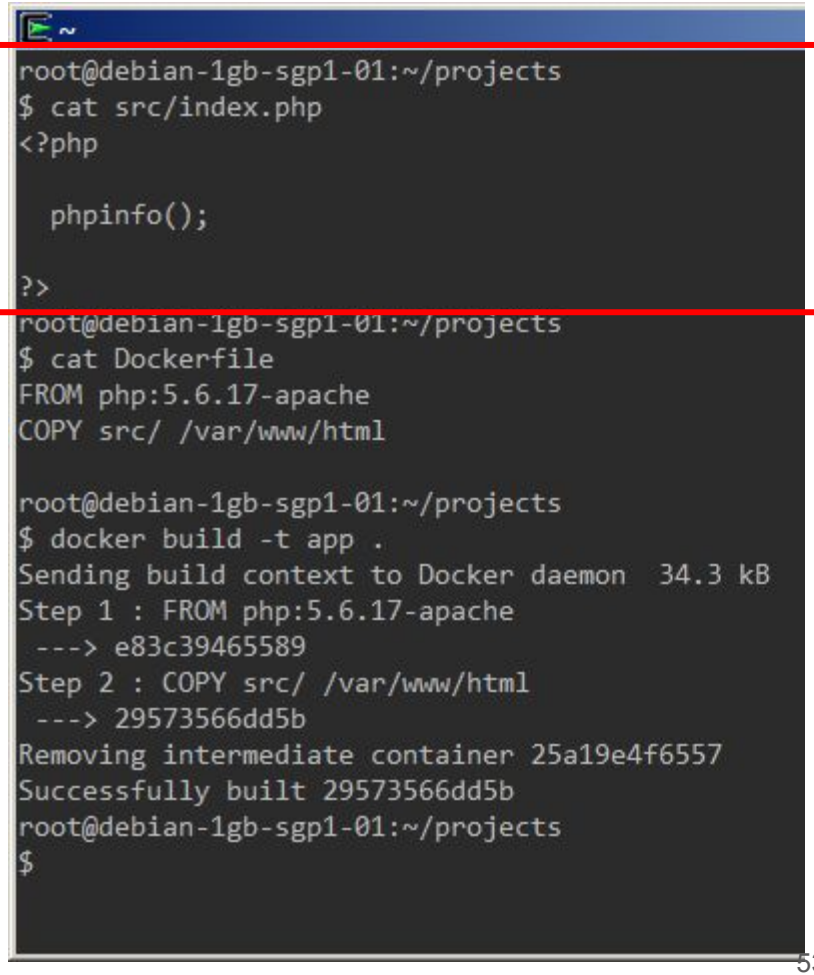
Prepare a program

File: src/index.php

<?php

phpinfo()

?>

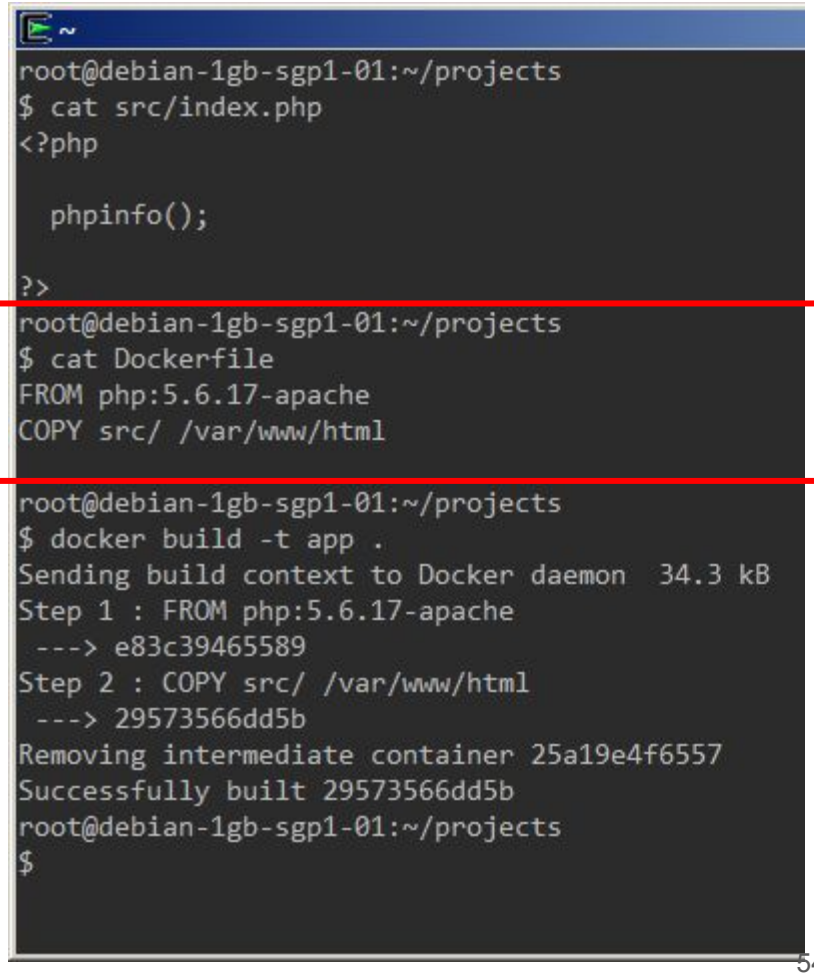


Dockerfile: COPY

File: Dockerfile

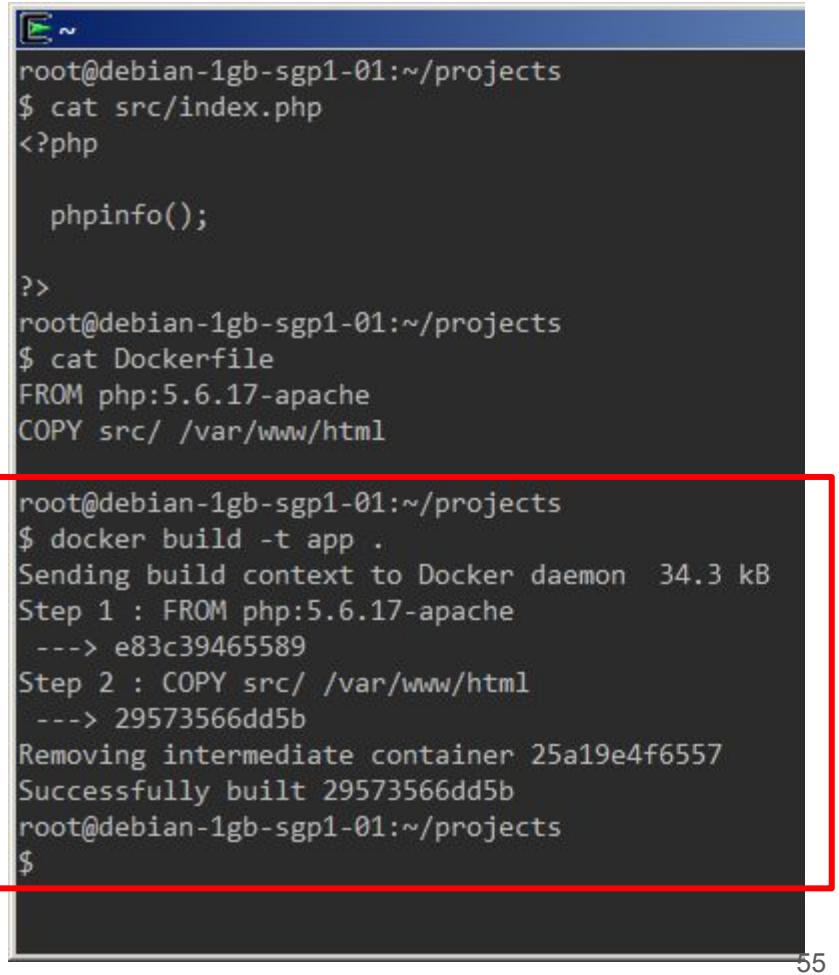
FROM php:5.6.17-apache

COPY src/ /var/www/html



Copy programs and re-build

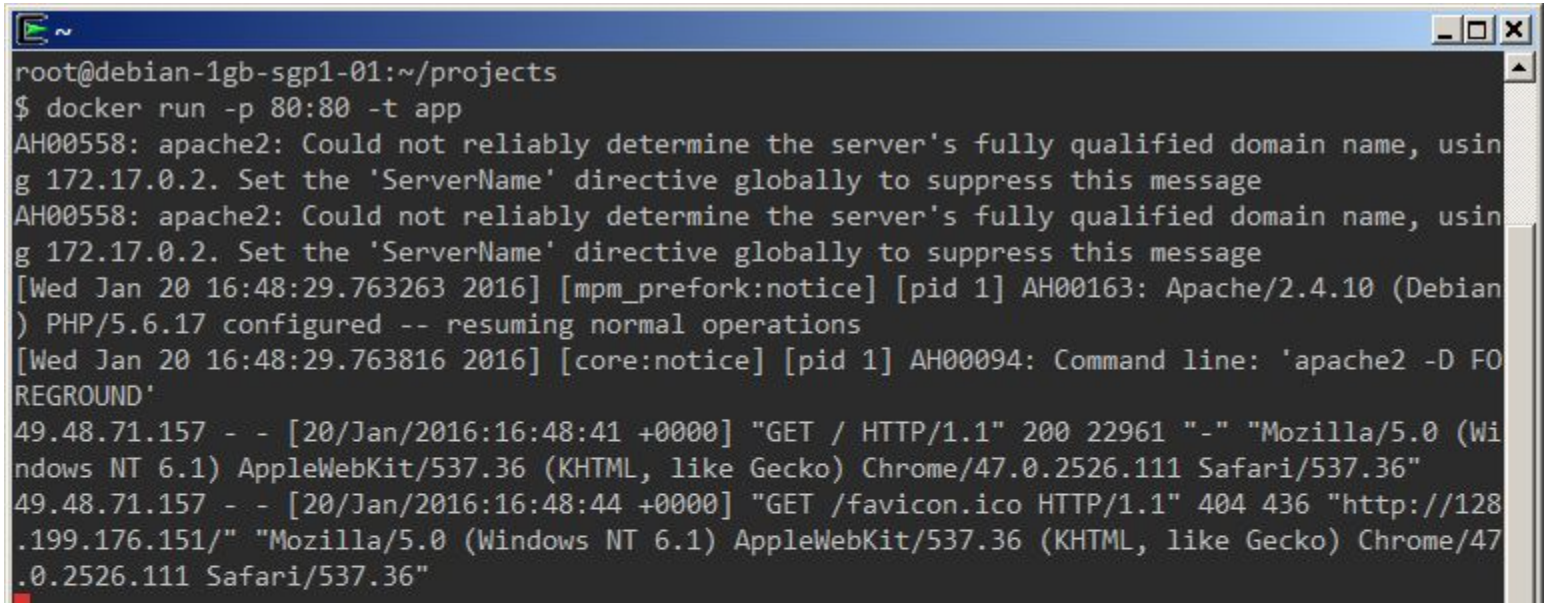
$ docker build -t app .



Docker: run

$ docker run -p 80:80 -t app

Ctrl-C to exit (but the container is still running)

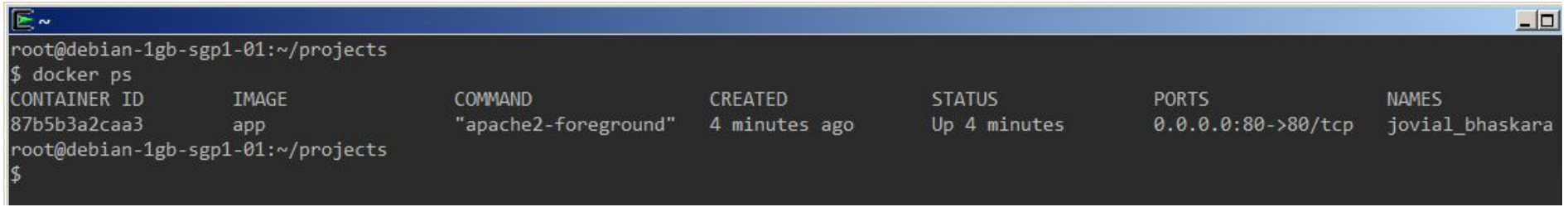


PHP Version 5.6.17

System	Linux 87b5b3a2caa3 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt11-1+deb8u5 (2015-10-09) x86_64
Build Date	Jan 7 2016 22:54:14
Configure Command	'./configure' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--with-apxs2' '--disable-cgi' '--enable-mysqlnd' '--with-curl' '--with-openssl' '--with-readline' '--with-recode' '--with-zlib'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,NTS
PHP Extension Build	API20131226,NTS

Docker: ps

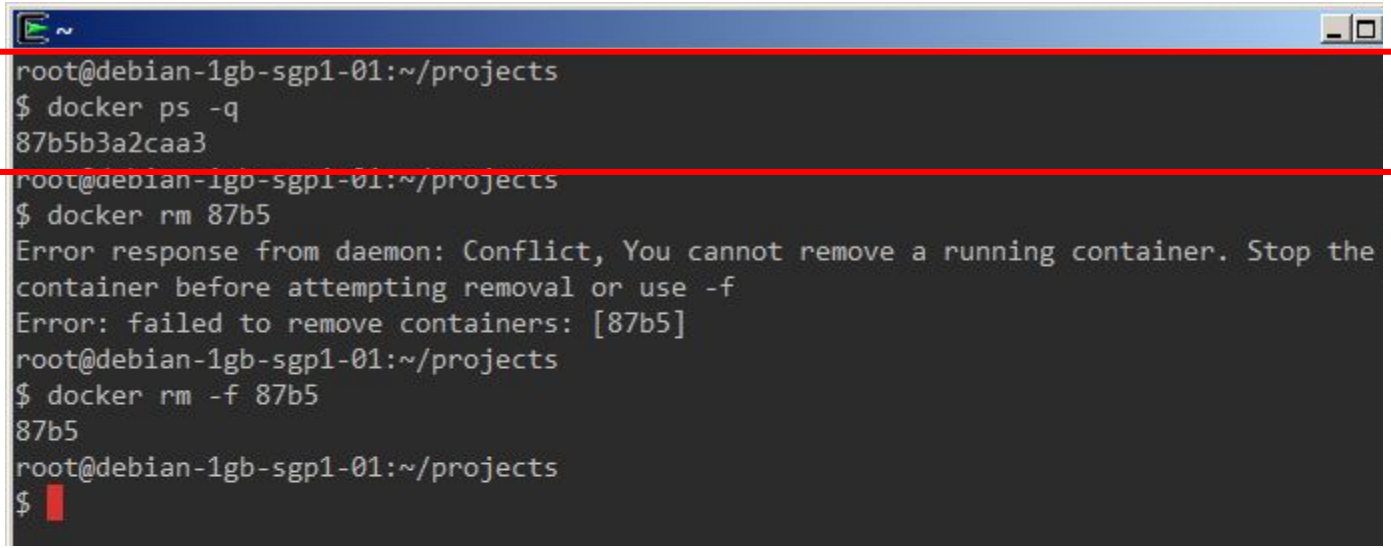
$ docker ps



Docker: ps -q

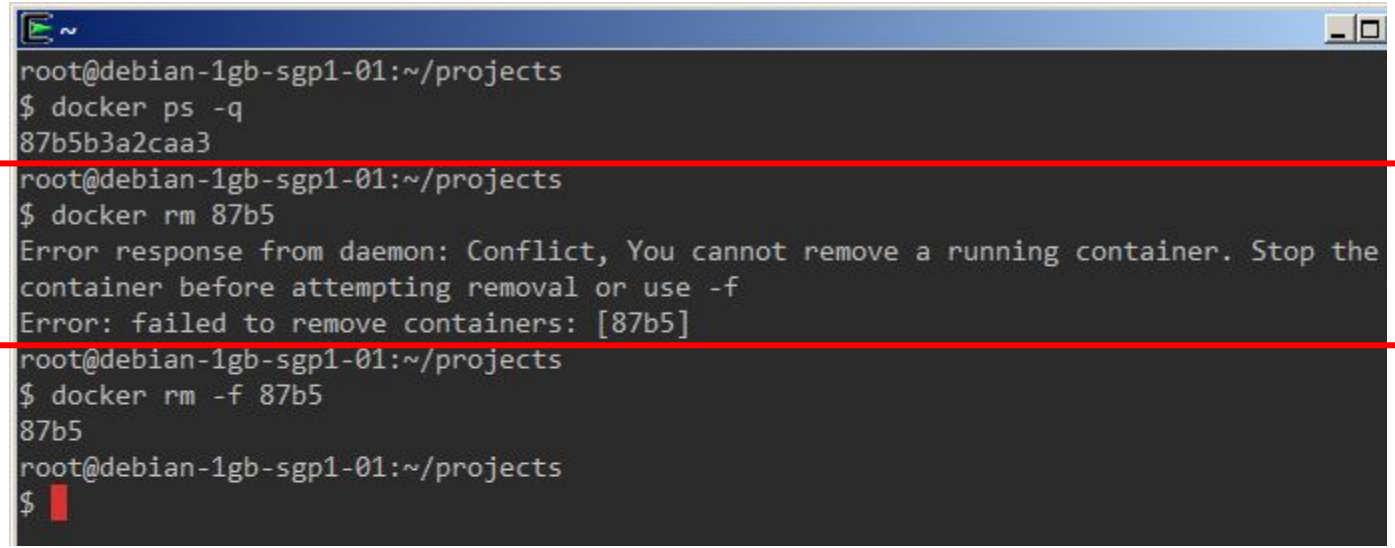
Show only the container's IDs

$ docker ps -q



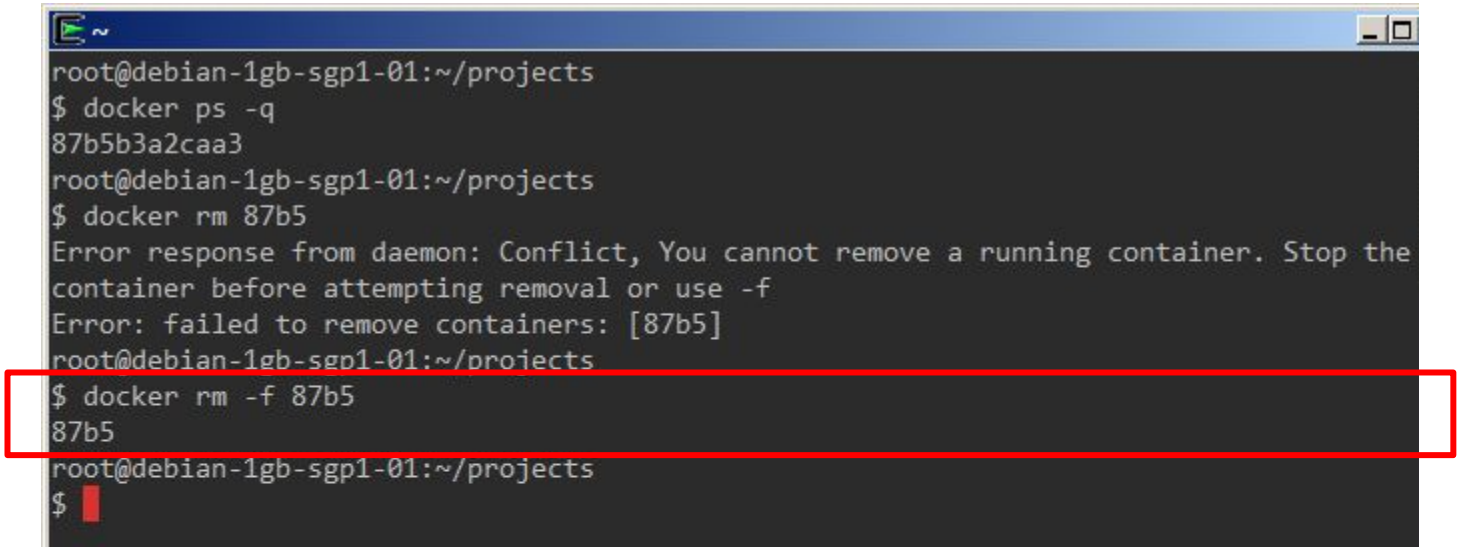
Docker: rm

$ docker rm <container ID>



Docker: rm -f

$ docker rm -f <container ID>



Docker Commands #1

- **attach** Attach to a running container
- **build** Build an image from a Dockerfile
- **commit** Create a new image from a container's changes
- **cp** Copy files/folders between a container and the local filesystem
- **create** Create a new container

Docker Commands #2

- **diff** Inspect changes on a container's filesystem
- **events** Get real time events from the server
- **exec** Run a command in a running container
- **export** Export a container's filesystem as a tar archive
- **history** Show the history of an image

Docker Commands #3

- **images** List images
- **import** Import the contents from a tarball to create a filesystem image
- **info** Display system-wide information
- **inspect** Return low-level information on a container or image
- **kill** Kill a running container

Docker Commands #4

- **load** Load an image from a tar archive or STDIN
- **login** Register or log in to a Docker registry
- **logout** Log out from a Docker registry
- **logs** Fetch the logs of a container
- **network** Manage Docker networks

Docker Commands #5

- **pause** Pause all processes within a container
- **port** List port mappings or a specific mapping for the CONTAINER
- **ps** List containers
- **pull** Pull an image or a repository from a registry
- **push** Push an image or a repository to a registry

Docker Commands #6

- **rename** Rename a container
- **restart** Restart a container
- **rm** Remove one or more containers
- **rmi** Remove one or more images
- **run** Run a command in a new container

Docker Commands #7

- **save** Save an image(s) to a tar archive
- **search** Search the Docker Hub for images
- **start** Start one or more stopped containers
- **stats** Display a live stream of container(s) resource usage statistics
- **stop** Stop a running container

Docker Commands #8

- **tag** Tag an image into a repository
- **top** Display the running processes of a container
- **unpause** Unpause all processes within a container
- **update** Update resources of one or more containers
- **version** Show the Docker version information
- **volume** Manage Docker volumes
- **wait** Block until a container stops, then print its exit code

ลดขนาด Docker Image

Image Size Reduction Techniques

- Squash Image
- Use smaller base image (e.g. Alpine)
- Combine installation commands (RUN in Dockerfile)

Squash Image

Squash Image

```
$ docker save <image id> \
  | docker-squash -from root -t <tag name> \
  | docker load
```

Dockerfile: issue a single RUN command

```
FROM    <base image>
RUN     apt-get install … \
        && apt-get
        && rm …
```

Use Alpine as Base Image

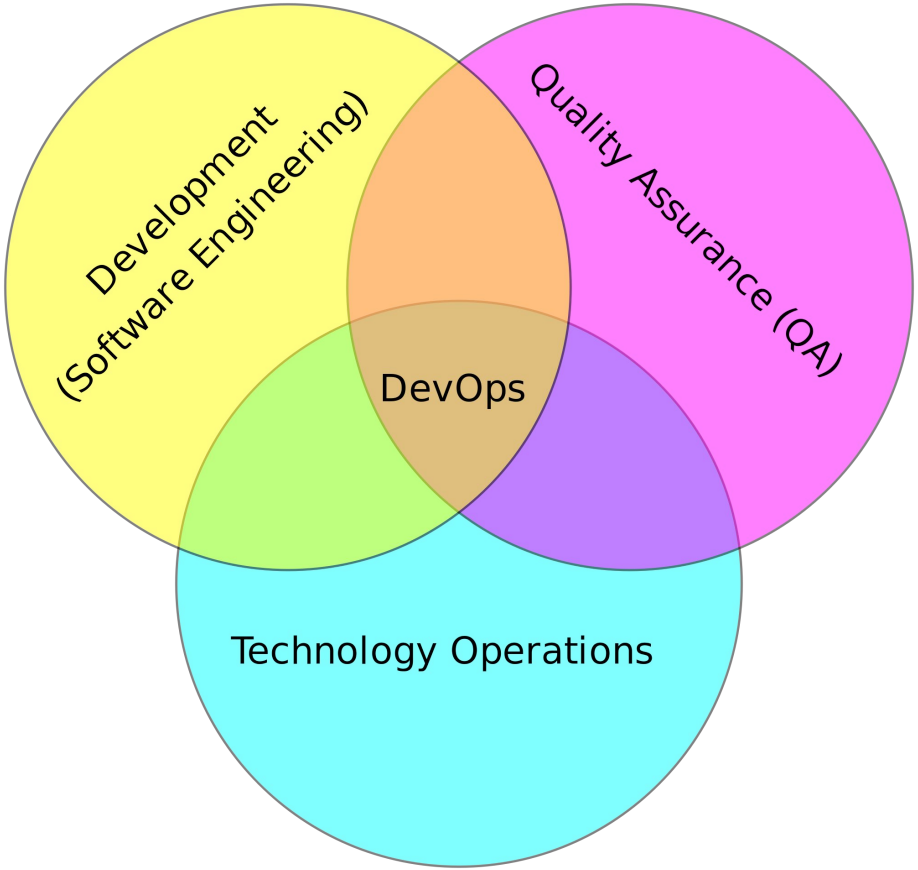
```
FROM    alpine:latest

RUN     <use alpine package manager>
```

เตรียม Image ด้วย Alpine

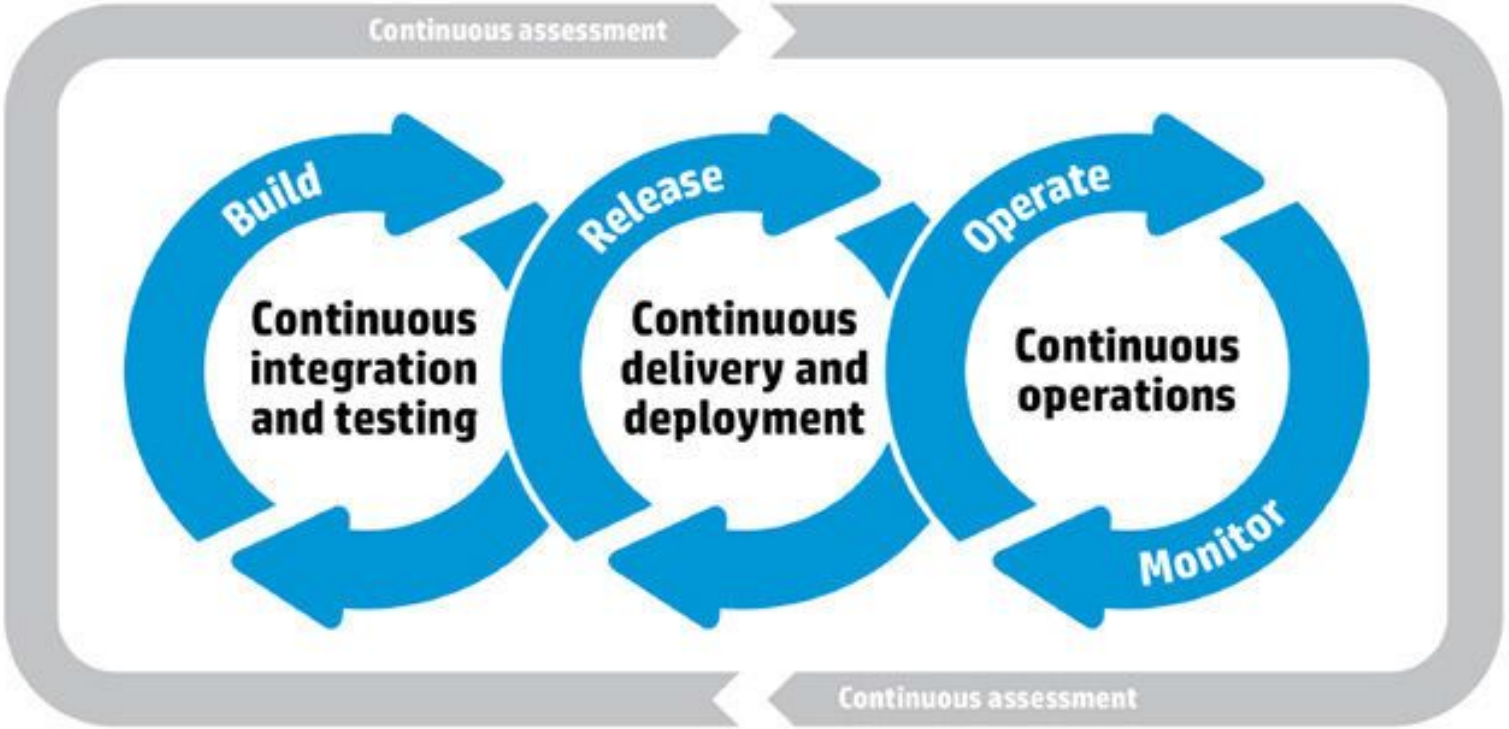
```
FROM    alpine:3.2

RUN     apk update && apk add nginx
```

วางระบบ Build

DevOps


Development
(Software Engineering)
Quality Assurance (QA)
DevOps
Technology Operations

DevOps


Continuous assessment
Build
Continuous integration and testing
Release
Continuous delivery and deployment
Operate
Continuous operations
Monitor
Continuous assessment

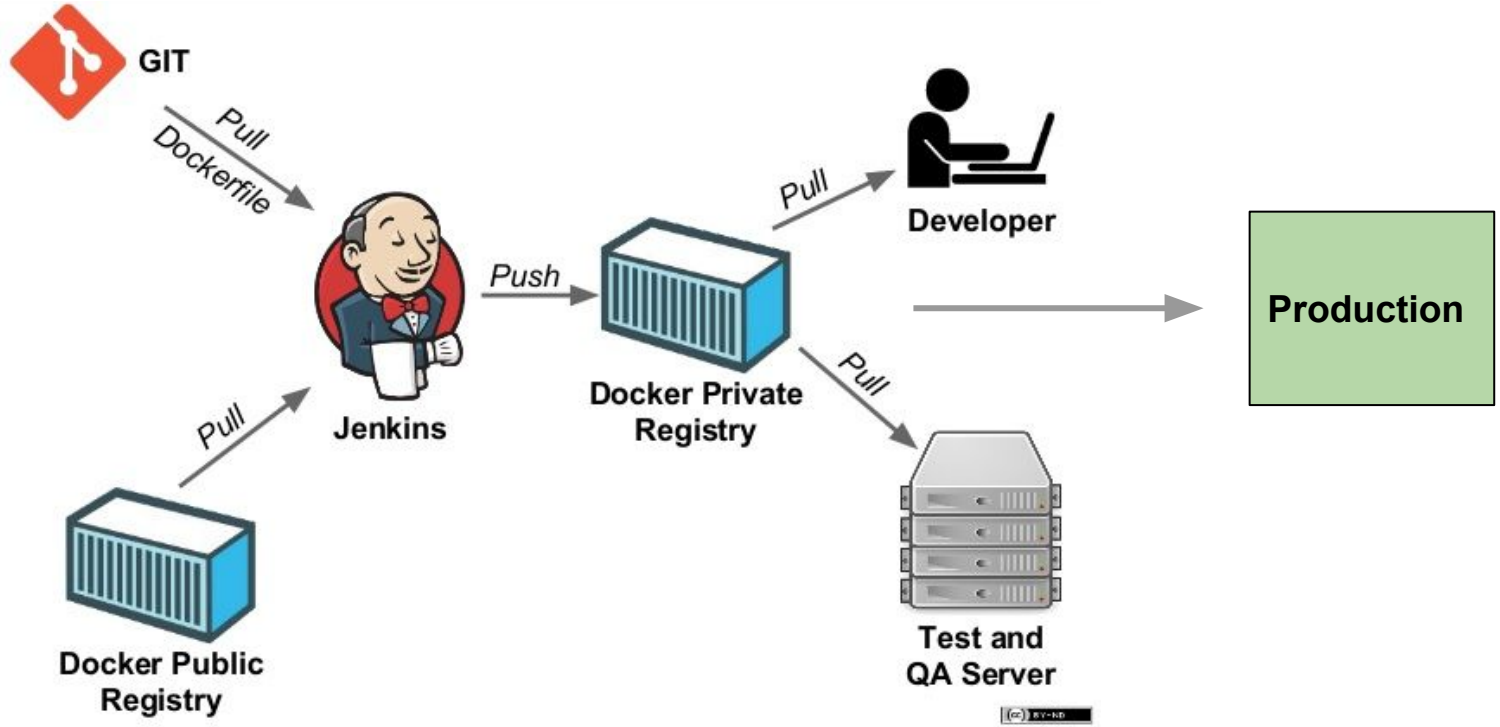
“ก็โปรแกรมมันเวิร์คแล้วในเครื่องหนู”

การเตรียม Continuous Integration ด้วย Docker

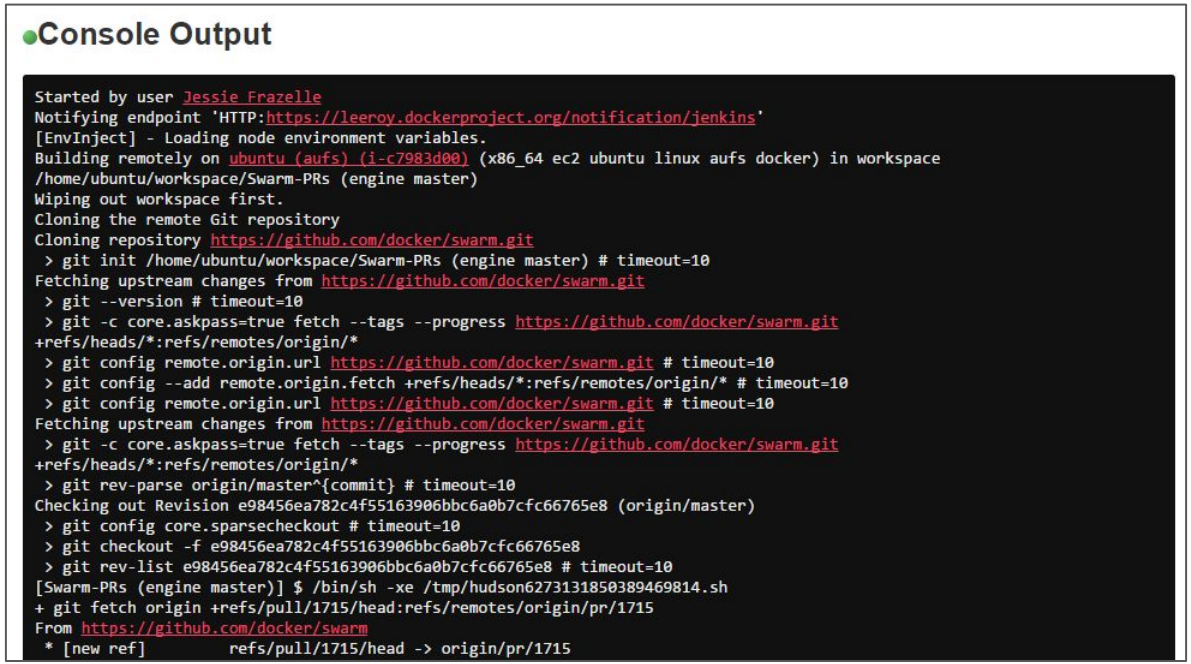
1. เก็บ Docker base images ไว้ใน Docker Registry
2. ดึง Codes จาก Version Control (เช่น Git)
3. ระบบจะ build และทดสอบ Docker images สำหรับแต่ละ Configuration
4. เมื่อ tests ผ่าน แล้ว Docker image จะถูก push ไปไว้ใน Registry
5. บน Production จะทำการ update Docker images ตัวใหม่ไปรันเป็น Container

https://www.docker.com/sites/default/files/UseCase/RA_CI%20with%20Docker_08.25.2015.pdf

Docker Build Pipeline



Jenkins Example



https://jenkins.dockerproject.org/view/Swarm/job/Swarm-PRs%20(engine%20master)/lastSuccessfulBuild/console

Jenkins Image

Jenkins

The Jenkins Continuous Integration and Delivery server.

This is a fully functional Jenkins server, based on the Long Term Support release http://jenkins-ci.org/.

Jenkins

How to use this image

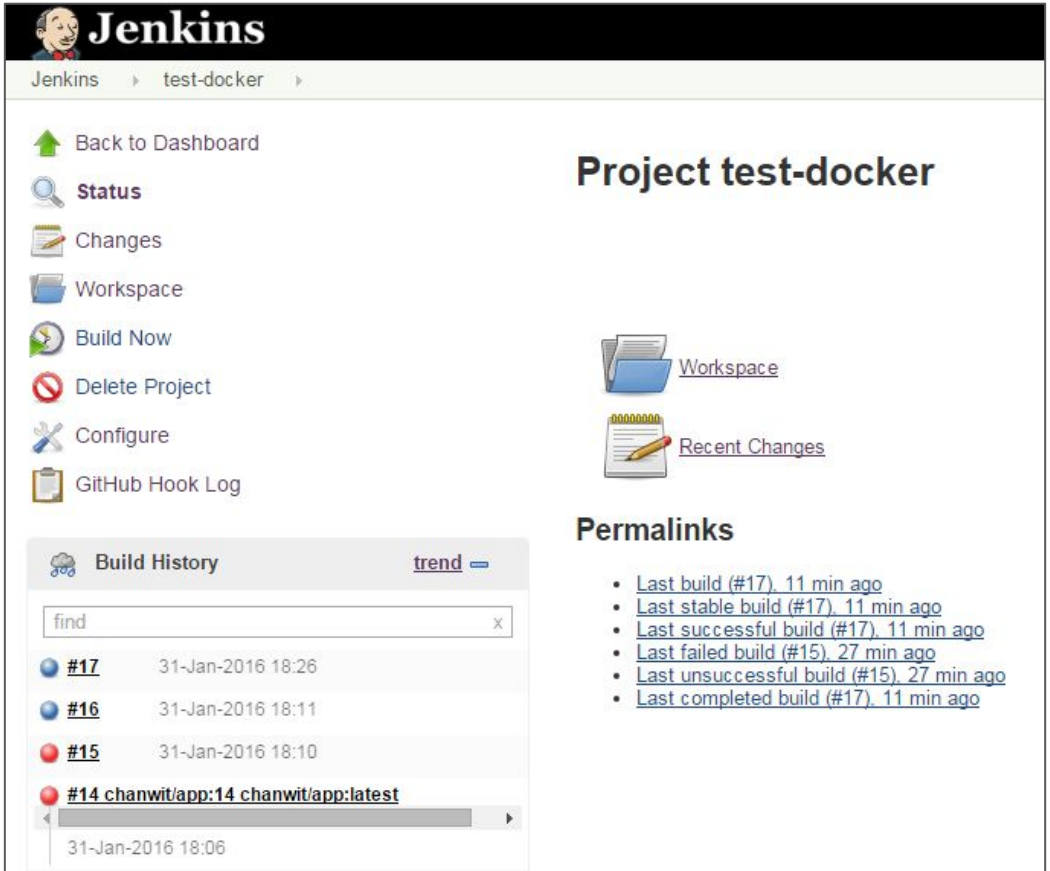
```
docker run -p 8080:8080 -p 50000:50000 jenkins
```

This will store the workspace in /var/jenkins_home. All Jenkins data lives in there - including plugins and configuration. You will probably want to make that a persistent volume (recommended):

```
docker run -p 8080:8080 -p 50000:50000 -v /your/home:/var/jenkins_home jenkins
```

This will store the jenkins data in `/your/home` on the host. Ensure that `/your/home` is accessible by the jenkins user in container (jenkins user - uid 1000) or use `-u some_other_user` parameter with `docker run` .

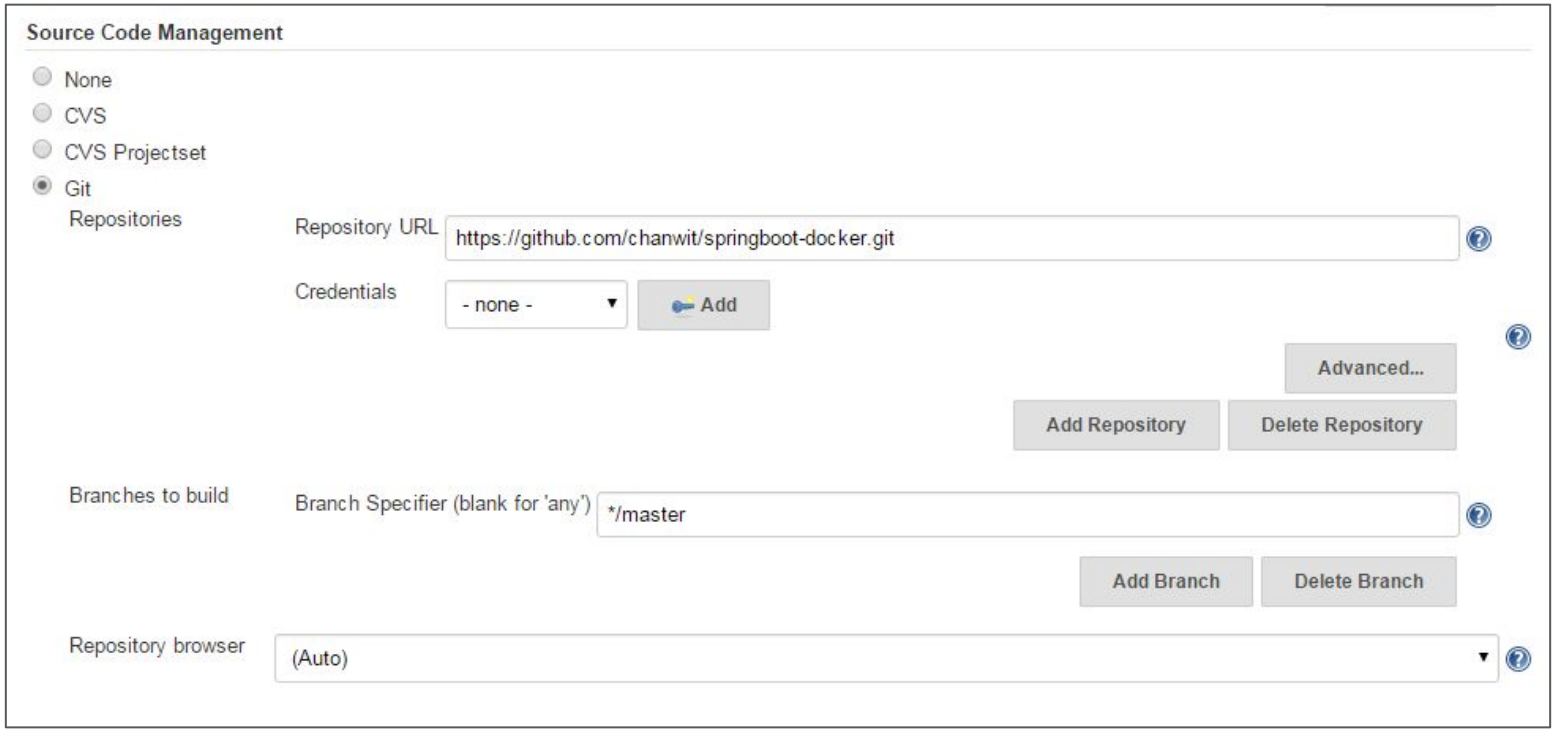
Jenkins “test-docker” Project



Jenkins - Add Plugins

- Docker Plugin
- GitHub Plugin
  - หรือ Plugin ของ Version Control ระบบอื่น ๆ
- Docker Binary inside Jenkins
  - หรือ Set ที่ Docker-common plugin

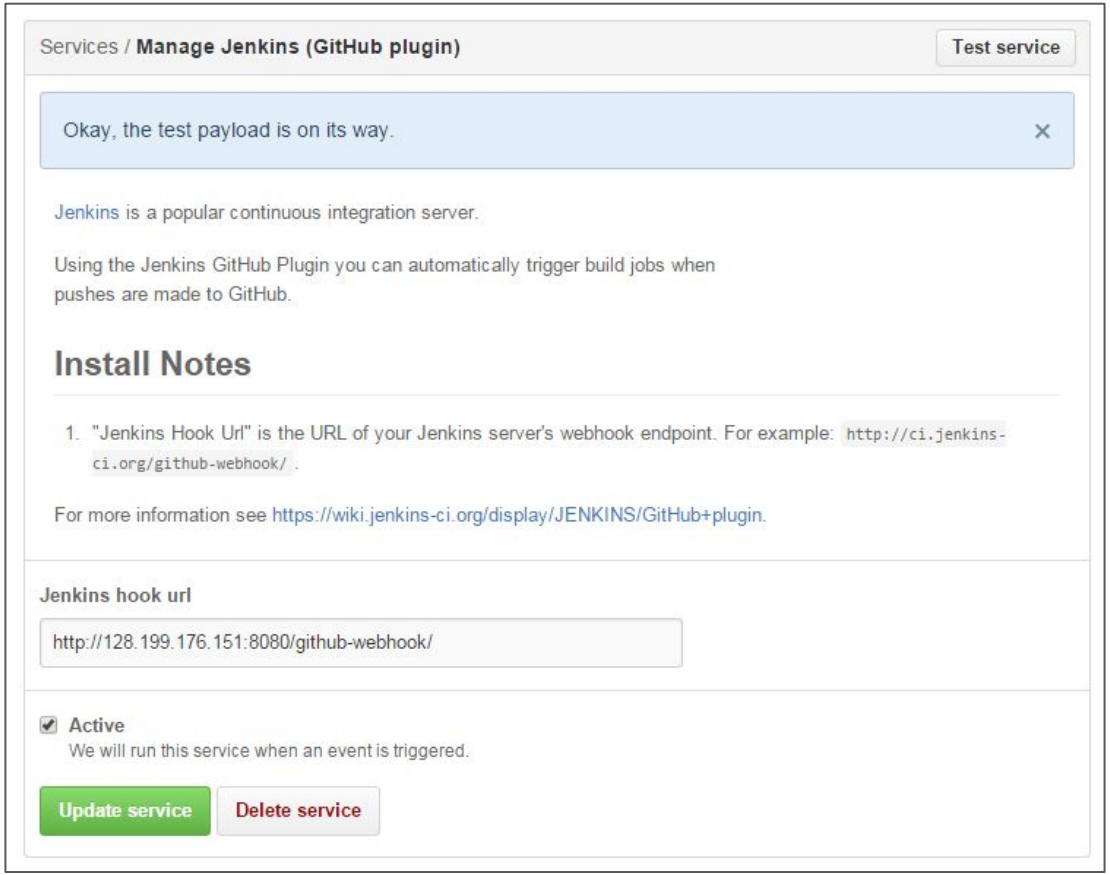
Link from Jenkins to Git Repository



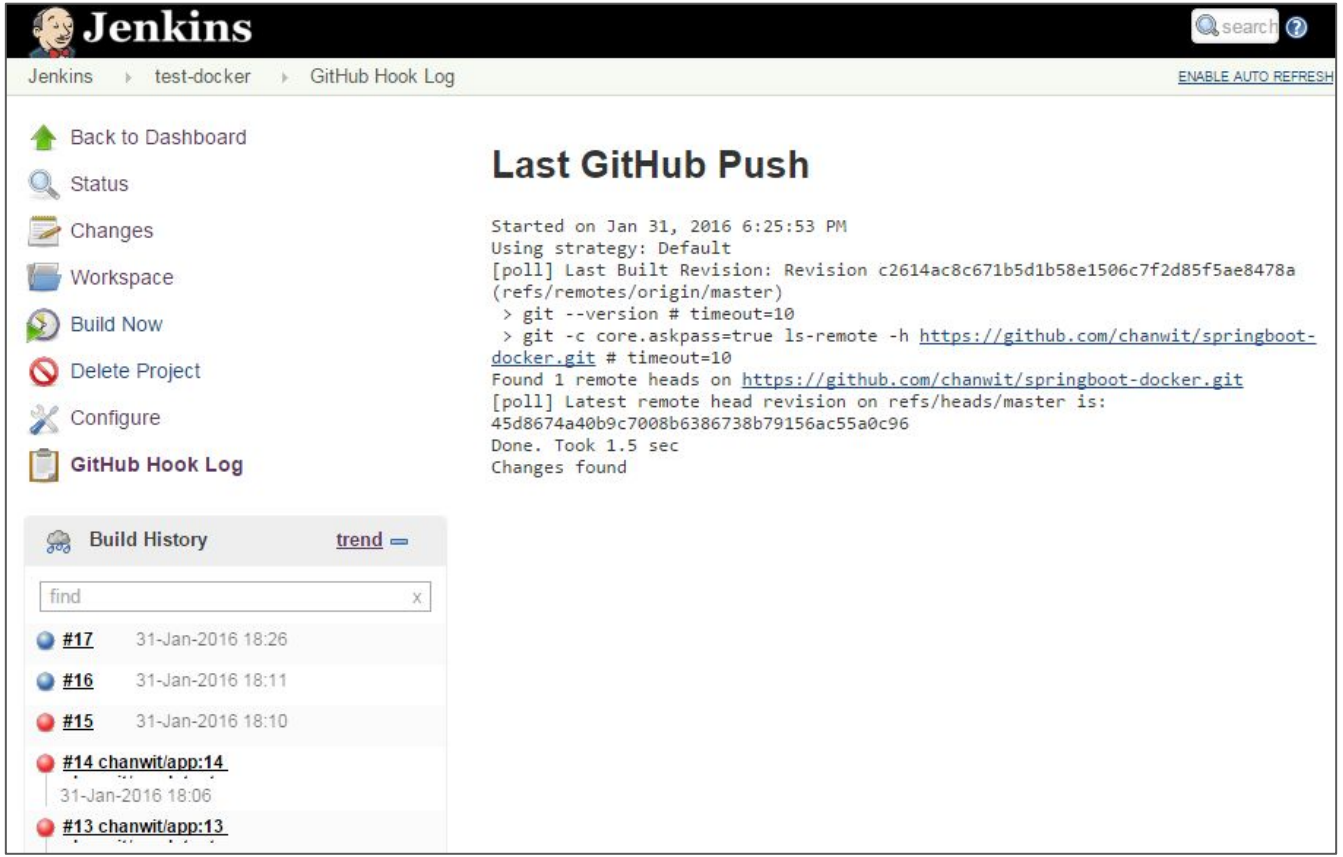
Build Steps


Build Triggers
Build after other projects are built
Build periodically
Build when a change is pushed to GitHub
Poll SCM
Build
Execute shell
Command
docker -H tcp://128.199.176.151:2375 build -t chanwit/app:$BUILD_NUMBER .
docker -H tcp://128.199.176.151:2375 push chanwit/app:$BUILD_NUMBER
See the list of available environment variables
Delete

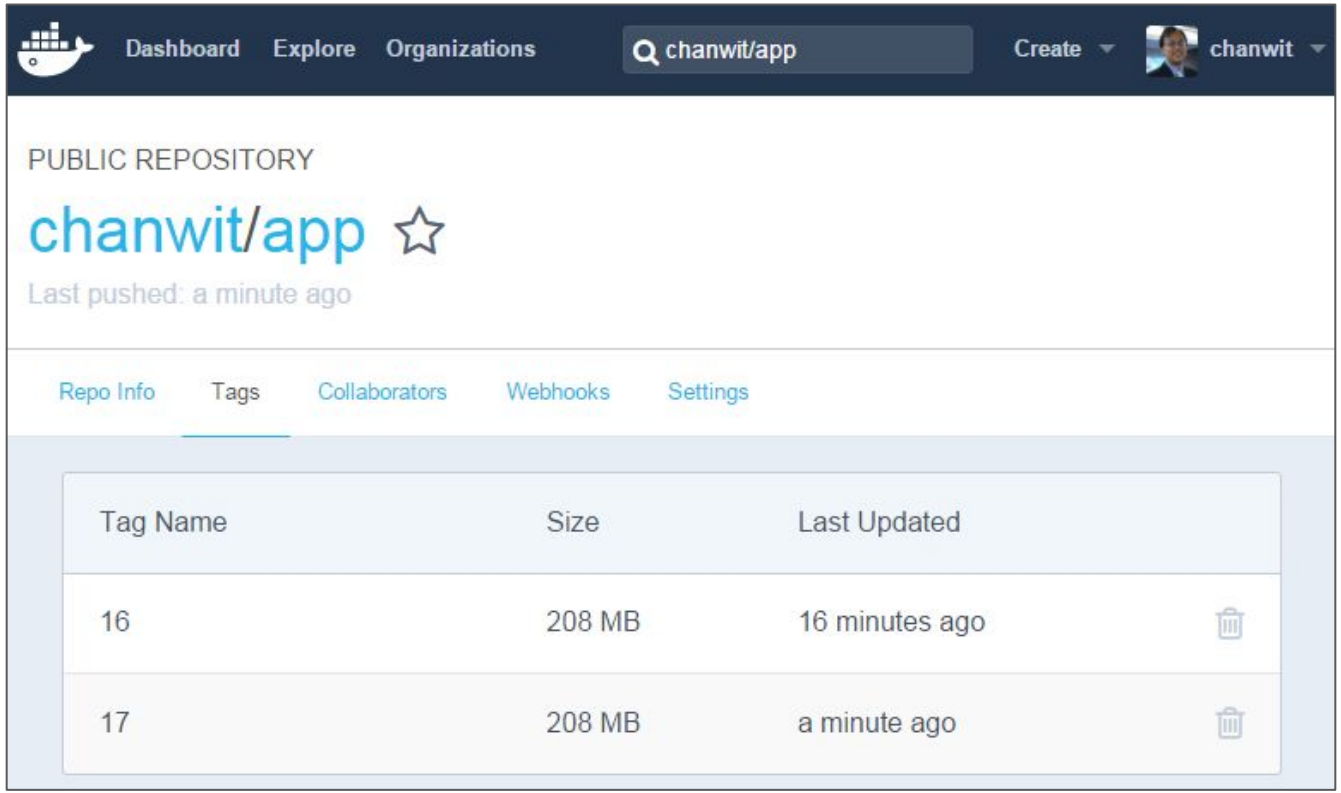
GitHub Hook


Services / Manage Jenkins (GitHub plugin)
Test service
Okay, the test payload is on its way.
Jenkins is a popular continuous integration server.
Using the Jenkins GitHub Plugin you can automatically trigger build jobs when pushes are made to GitHub.
Install Notes
1. "Jenkins Hook Url" is the URL of your Jenkins server's webhook endpoint. For example: http://ci.jenkins-ci.org/github-webhook/ .
For more information see https://wiki.jenkins-ci.org/display/JENKINS/GitHub+plugin.
Jenkins hook url
http://128.199.176.151:8080/github-webhook/
Active
We will run this service when an event is triggered.
Update service
Delete service

Pull from Git Repository



Ready to Deploy



ใช้ Docker สำหรับพัฒนาโปรแกรม

Onbuild Image

- เหมือน Abstract Class ใน OOP

เช่น

- ONBUILD COPY src/ /app/src
  - เพิ่มไฟล์จาก **src/** เข้าไปใน “**image** ลูก” ตอนสั่ง **build**

Java Microservices และ Docker

Java - Spring Boot

http://containertutorials.com/docker-compose/spring-boot-app.html

- Maven Build
- Gradle Build

Java - Spring Boot

Spring Boot Onbuild

Dockerfile

Raw

```
FROM frolvlad/alpine-oraclejdk8

ENV SRC_DIR /app/src
ENV BIN_DIR /app/bin
ENV SPRINGBOOT_VERSION 1.2.3.RELEASE

RUN apk update && apk add bash curl unzip
RUN curl -s get.sdkman.io | bash
RUN bash -c "source $HOME/.sdkman/bin/sdkman-init.sh && sdk install gradle && sdk install springboot $SPRINGBOOT_VERSION"

RUN mkdir -p $SRC_DIR
WORKDIR $SRC_DIR

EXPOSE 8080

CMD ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app/bin/app.jar"]

ONBUILD RUN mkdir -p $BIN_DIR
ONBUILD ADD . $SRC_DIR
ONBUILD RUN bash -c "source $HOME/.bashrc && spring jar $BIN_DIR/app.jar $SRC_DIR"
```

Java - Tomcat

- A Small Tomcat Image

https://github.com/jeanblanchard/docker-tomcat/blob/master/Dockerfile

Node.JS

Node.JS onbuild Image

How to use this image

Create a `Dockerfile` in your Node.js app project

```
FROM node:4-onbuild
# replace this with your application's default port
EXPOSE 8888
```

You can then build and run the Docker image:

```
$ docker build -t my-nodejs-app .
$ docker run -it --rm --name my-running-app my-nodejs-app
```

Notes

The image assumes that your application has a file named `package.json` listing its dependencies and defining its start script.

Ruby on Rails

Ruby on Rails

How to use this image

Create a `Dockerfile` in your Rails app project

```
FROM rails:onbuild
```

Put this file in the root of your app, next to the `Gemfile`.

This image includes multiple `ONBUILD` triggers which should cover most applications. The build will `COPY . /usr/src/app`, `RUN bundle install`, `EXPOSE 3000`, and set the default command to `rails server`.

You can then build and run the Docker image:

```
$ docker build -t my-rails-app .
$ docker run --name some-rails-app -d my-rails-app
```

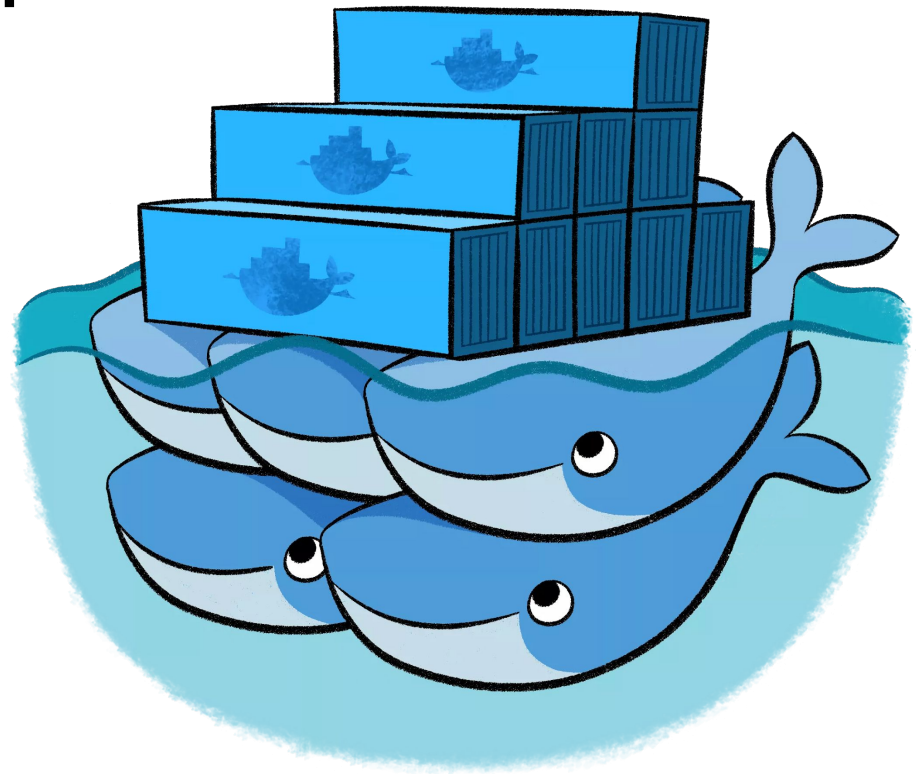
You can test it by visiting `http://container-ip:3000` in a browser or, if you need access outside the host, on port 8080:

```
$ docker run --name some-rails-app -p 8080:3000 -d my-rails-app
```

You can then go to `http://localhost:8080` or `http://host-ip:8080` in a browser.

สร้าง Cluster ด้วย Docker Swarm

Docker Swarm



Concept ของการ Clustering

- ก่อน 1.12
- ฟอร์ม Docker Engine หลาย ๆ ตัวรวมกันเป็น Virtual Engine ตัวเดียว
- ผ่านระบบคล้าย Proxy == Docker Swarm
- API ของ Swarm และ Engine ใช้ชุดเดียวกัน
- ทีม Docker Swarm จะพยายาม keep ให้ compatibility เป็น 100% เสมอ

Cluster Components

- 1 Discovery Service
- 1 Master
- N Nodes

HA Cluster Components

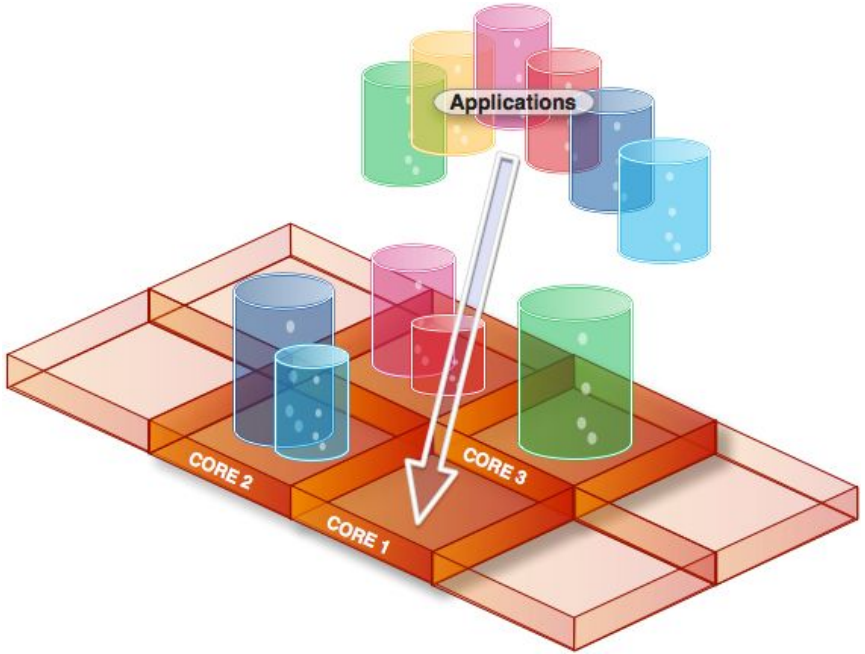
- 3 Discovery Service
- 3 Master
- N Nodes

Concept ของการ Clustering: Discovery

- ต้องมีตัวกลางสำหรับเก็บข้อมูล Node
- ตัวกลางเรียกว่า Cluster Store (KV Store)
- Concept ใน Swarm เรียกตัวกลางว่า Discovery Service
  - Etcd
  - Consul
  - ZooKeeper
  - File
  - Node

Concept ของการ Clustering: Strategy

- ตัว Cluster มี Strategy ในการเลือก Node ให้ Container
- ตัวปกติคือ Spread
  - กระจายให้มากที่สุด == Performance
- การคำนวณใช้ 2 แกน
  - CPU ของ Node
  - Memory ของ Node
- Binpack
  - อัดแน่นสุด
- Random
  - สุ่ม



Concept ของการ Clustering: Filters

- Health
  - กรอง node ที่ตายทิ้ง
- Port
  - กรอง node ที่ port ไม่ว่างทิ้ง
- Dependency
- Affinity
- Constraint

```
24
25  func init() {
26          filters = []Filter{
27                  &HealthFilter{},
28                  &PortFilter{},
29                  &DependencyFilter{},
30                  &AffinityFilter{},
31                  &ConstraintFilter{},
32          }
33  }
```

Concept ของการ Clustering: Filters

- Health
- Port
- Dependency
  - วาง container ที่โยงกันไว้ใน ีnode เดียวกัน
- Affinity
- Constraint

```
24
25  func init() {
26          filters = []Filter{
27                  &HealthFilter{},
28                  &PortFilter{},
29                  &DependencyFilter{},
30                  &AffinityFilter{},
31                  &ConstraintFilter{},
32          }
33  }
```

Concept ของการ Clustering: Filters

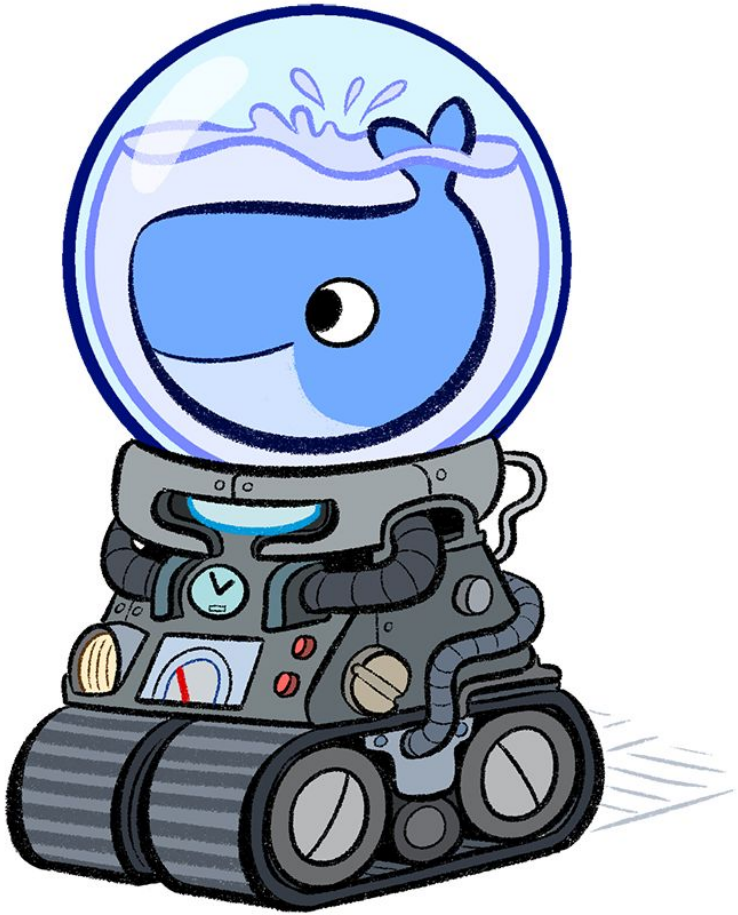
- Health
- Port
- Dependency
- Affinity
  - เลือก node ตาม container หรือ image บน node นั้น ๆ
- Constraint
  - เลือก node ตาม label ของ node

```
24
25  func init() {
26          filters = []Filter{
27                  &HealthFilter{},
28                  &PortFilter{},
29                  &DependencyFilter{},
30                  &AffinityFilter{},
31                  &ConstraintFilter{},
32          }
33  }
```

Provision Cluster via Docker Machine

- ใช้ Docker Machine ช่วยในการเชื่อมต่อ cluster มายังโลกภายนอก

Docker Machine



Docker Machine

- Tool สำหรับทำการ provision เครื่องให้พร้อมใช้งาน Docker
- ถ้าเตรียมเครื่องอย่างระมัดระวังจะสามารถย้าย Container ไปรันข้าม Cloud Provider ได้อย่างสะดวก
- ปกติ Image จะเป็น Ubuntu
- แนะนำ Debian 8 (Jessie) Image
  - Kernel 3.16
  - ปัญหาน้อย
  - Test แล้ว Provision ได้ง่ายทั้งบน DigitalOcean และ OpenStack
  - ปู Virtual Network ง่าย
  - มี Kernel Module ค่อนข้างครบ

Docker Machine Drivers

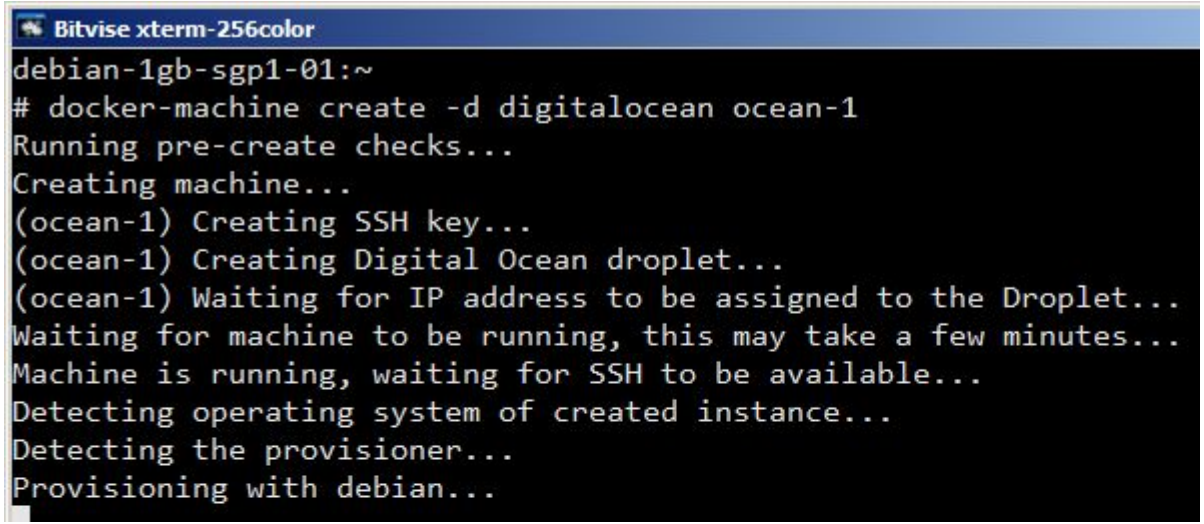
- amazonec2
- azure
- digitalocean
- exoscale
- generic
- google
- hyperv
- openstack
- rackspace
- softlayer
- virtualbox
- vmwarefusion
- vmwarevcloudair
- vmwarevsphere

Docker Machine: create

```
$ docker-machine create \
      -driver virtualbox \
      box2
```

Debian Jessie Machine

```
$ export DIGITALOCEAN_IMAGE=debian-8-x64
$ docker-machine create –driver digitalocean \
 –digitalocean-image=debian-8-x64 \
 ocean-1
```



Docker Machine: ls - List

$ docker-machine ls

```
NAME   ACTIVE   URL          STATE     URL                         SWARM   DOCKER        ERRORS
os-1   -        openstack    Running   tcp://xx.xx.xx.xx:2376              v1.10.0-rc2
os-2   -        openstack    Running   tcp://xx.xx.xx.xx:2376              v1.10.0-rc2
```

$ docker-machine ls -f "{{.Name}}"

```
os-1
os-2
```

Docker Machine: rm - Remove

- Manual confirmation

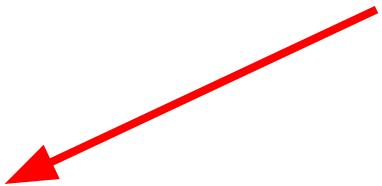
$ docker-machine rm os-1

- Auto confirmation

$ docker-machine rm -y os-1

- Force remove

$ docker-machine rm -f os-1



ระวังเครื่องบนคลาวด์
อาจจะยังค้างอยู่

Docker Engine Options

- ส่งให้ **Docker Machine**
- เพื่อทำการ **customize Docker Engine** ระหว่างการ **provision**

- เช่น อยากกำหนดป้าย **region**
- **–engine-label region=thai**
- **–engine-label region=china**
- **–engine-label region=usa**

Docker Engine Storage Drivers

- AUFS
- DeviceMapper (Thin Provision)
- BTRFS
- Overlay
  - บางครั้งต้อง **modprobe** ก่อน
- ใช้ `--engine-storage-driver` ใน Docker Machine

Docker Engine Logging Drivers

Configure logging drivers

The container can have a different logging driver than the Docker daemon. Use the `--log-driver=VALUE` with the `docker run` command to configure the container's logging driver. The following options are supported:

`none`	**Disables any logging for the container. `docker logs` won't be available with this driver.**
`json-file`	Default logging driver for Docker. Writes JSON messages to file.
`syslog`	Syslog logging driver for Docker. Writes log messages to syslog.
`journald`	Journald logging driver for Docker. Writes log messages to `journald`.
`gelf`	Graylog Extended Log Format (GELF) logging driver for Docker. Writes log messages to a GELF endpoint likeGraylog or Logstash.
`fluentd`	Fluentd logging driver for Docker. Writes log messages to `fluentd` (forward input).
`awslogs`	Amazon CloudWatch Logs logging driver for Docker. Writes log messages to Amazon CloudWatch Logs.

Gelf Logging Driver

gelf options

The GELF logging driver supports the following options:

```
--log-opt gelf-address=udp://host:port
--log-opt tag="database"
--log-opt labels=label1,label2
--log-opt env=env1,env2
```

The `gelf-address` option specifies the remote GELF server address that the driver connects to. Currently, only `udp` is supported as the transport and you must specify a `port` value. The following example shows how to connect the `gelf` driver to a GELF remote server at `192.168.0.42` on port `12201`

```
$ docker run --log-driver=gelf --log-opt gelf-address=udp://192.168.0.42:12201
```

Discovery in Docker Engine

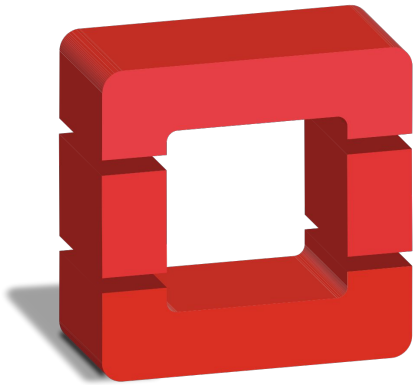
- เมื่อก่อนเป็น code ใน Docker Swarm ตอนนี้ค่อย ๆ ย้ายไป Docker Engine

- 3 options สำหรับ setup overlay network
- –engine-opt
  - –cluster-store
  - –cluster-advertise
  - –cluster-store-opt

Discovery in Docker Engine

- ตั้ง Cluster Store
  - ใช้ Consul
- เตรียม Node Advertising to Cluster Store
  - -cluster-advertise eth0:2376

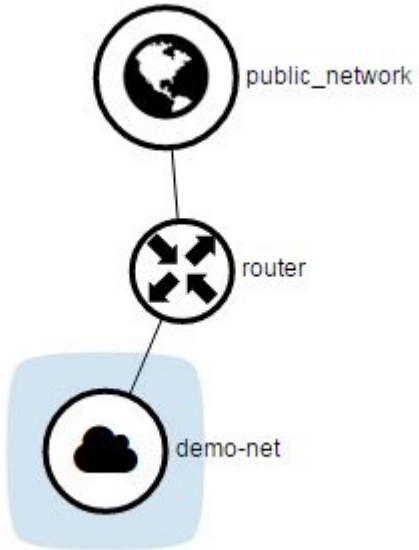
Provision on OpenStack



openstack®

CLOUD SOFTWARE

OpenStack Network



OpenStack RC File

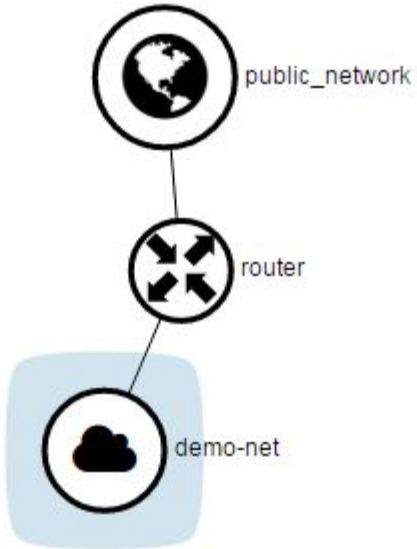
ไฟล์ตั้งค่าพารามิเตอร์ของ OpenStack

โหลดได้จาก Horizon Dashboard

$ source ./openrc.sh (แล้วแต่ว่าโหลดลงมาแล้วชื่ออะไร)

OpenStack Provision Parameters

- –openstack-flavor-name
  - “m1.small”
- –openstack-image-name
  - เช่น “Debian-8.2-x86_64”
    (ดูจากรายการ Image ใน Horizon)
- –openstack-ssh-user
  - root
  - centos
  - debian
- –openstack-net-name
  - private network name ที่ต่อผ่าน Virtual Router
- –openstack-floatingip-pool
  - เช่น public_network



Provision on DigitalOcean

DigitalOcean Provision Parameters

- –digitalocean-access-token
  - secret
- –digitalocean-region
  - nyc3
  - sgp1
- –digitalocean-ssh-user
  - root
  - centos
  - debian

Creating a Cluster

- 1 Discovery Service
  - เราจะใช้ consul
- 1 Node สำหรับ Swarm Master
  - รันตัว Manager (เป็น node ด้วย)
- 2 Node สำหรับ Engine

Provision a Cluster

Hands On

Docker Machine: env - Environment

ใช้ต่อเซ็ตการเชื่อมต่อให้คำสั่ง docker & docker-compose ผ่าน Environment Variables

- เชื่อมต่อปกติ

```
$ eval $(docker-machine config node-1)
$ docker ps
```

- เชื่อมต่อแบบคลัสเตอร์

```
$ eval $(docker-machine env –swarm node-master)
$ docker ps
```

Docker Machine: config - Configuration

ใช้ต่อคำสั่ง docker ถ้าต้องการเลี่ยงการเซ็ต Environment Variables

- เชื่อมต่อปกติ

$ docker $(docker-machine config node-1) ps

- เชื่อมต่อแบบคลัสเตอร์

$ docker $(docker-machine config --swarm node-master) ps

Docker Machine: ip - Show IP Address

$ docker-machine ip node-1

Docker Machine: ssh - Secured Shell

$ docker-machine ssh node-1

Docker Machine: scp - Copy files across machines

- คัดลอกไฟล์ข้ามจากเครื่องนึงไปอีกเครื่องนึง
  - โดยไม่ต้องย้ายมาเครื่องกลางก่อน

```
$ docker-machine scp -r ocean-master:/etc/ceph ocean-1:/etc/ceph
```

Using our Swarm Cluster

Hands on

Ex: รัน Container ตาม Zone

$ docker run -d -e constraint:**region**==thai nginx

ป้าย **region** เป็นสิ่งที่กำหนดขึ้นเองตอน **provision**

Ex: รัน Container ตาม Node

$ docker run -d -e constraint:node!=master nginx

Ex: สร้าง Container และรันบน Node ที่สร้าง

$ docker build –build-arg constraint:node==node-1 -t app .

รัน container บนโหนดที่มี image ชื่อ app

$ docker run -d -e affinity:image==app -t app

Ex: Soft Constraint

ใช้ตัว tild (~) เพื่อบอกว่า Constraint เป็นแบบ Soft

ถ้าเป็น Soft Constraint แล้วเงื่อนไขไม่ตรงก็จะยัง run container

$ docker run -d -e constraint:region==~europe -t app

เราไม่ได้กำหนด region เป็น europe ไว้ แต่ app ก็จะรันบน node ใด ึnode หนึ่ง

เตรียม Docker Network

Linux vxlan - L2 Network Virtualization

vxlan ทำงานบน L2 (เป็น L2 packet ที่ encapsulated ด้วย L3 ของ network จริง)

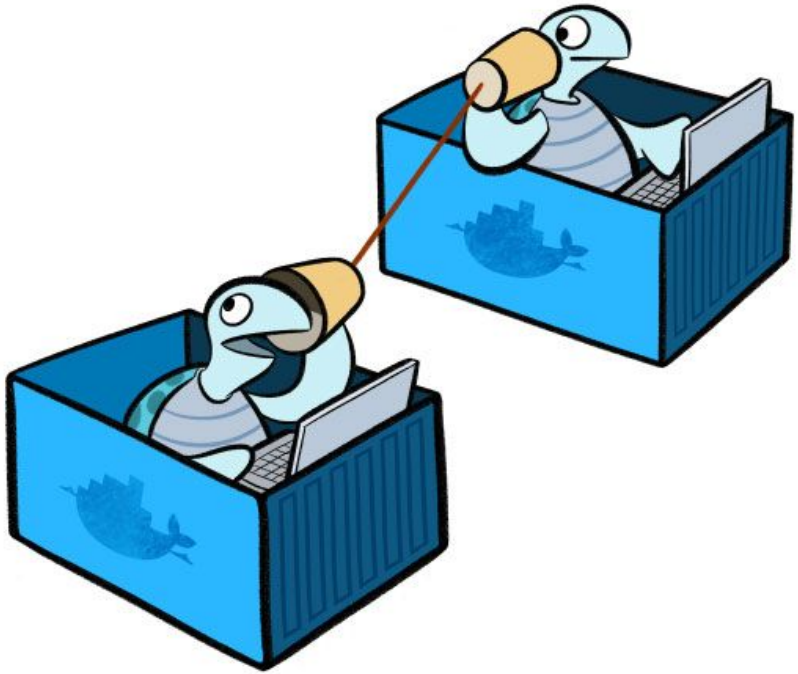
libnetwork's overlay driver ทำงานบน L3

(Calico driver ทำงานบน L3)

โดยหลักการแล้วสามารถใช้ service ตั้งแต่ L4 ขึ้นไป ได้ทุกแบบ

Network Types

- Bridge
- Host
- Custom
  - เช่น overlay
  - หรือ Calico (3rd party)

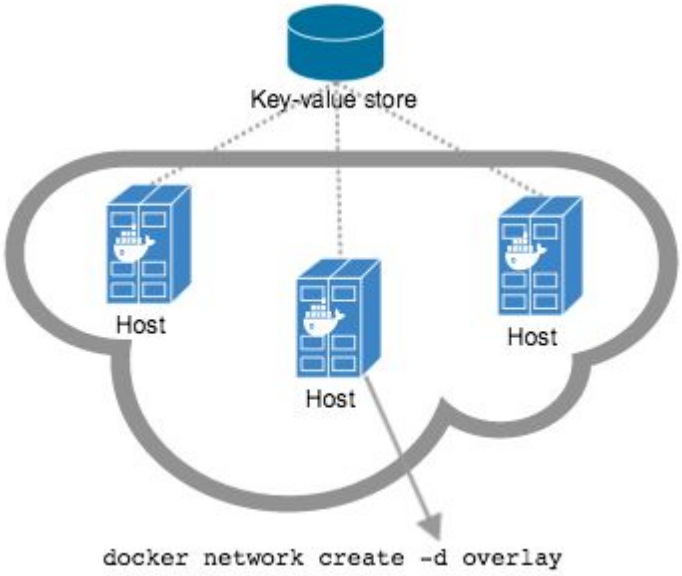


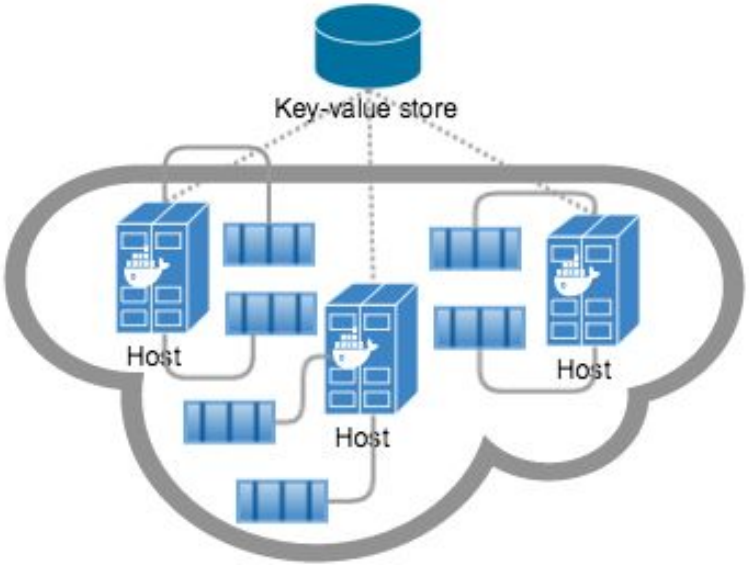
Network Create: overlay

$ docker network create –driver overlay <name>

ถ้า config ถูกต้องจะเห็น network ID เดียวกันทั้ง cluster

Docker Networking





Container with 2 Networks

$ docker create ...

$ docker network connect <network> <container id>

$ docker start <container id>

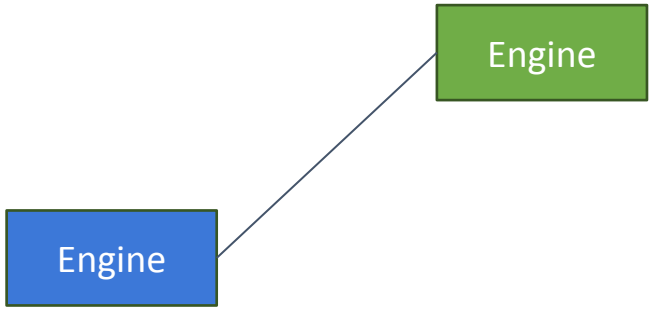
Docker 1.12 Swarm mode

- แตกต่างจาก Docker Swarm ในเชิง concept
- Rewrite from Scratch
- Concepts of
  - Services
  - Tasks

Swarm Mode

- `$ docker swarm init`

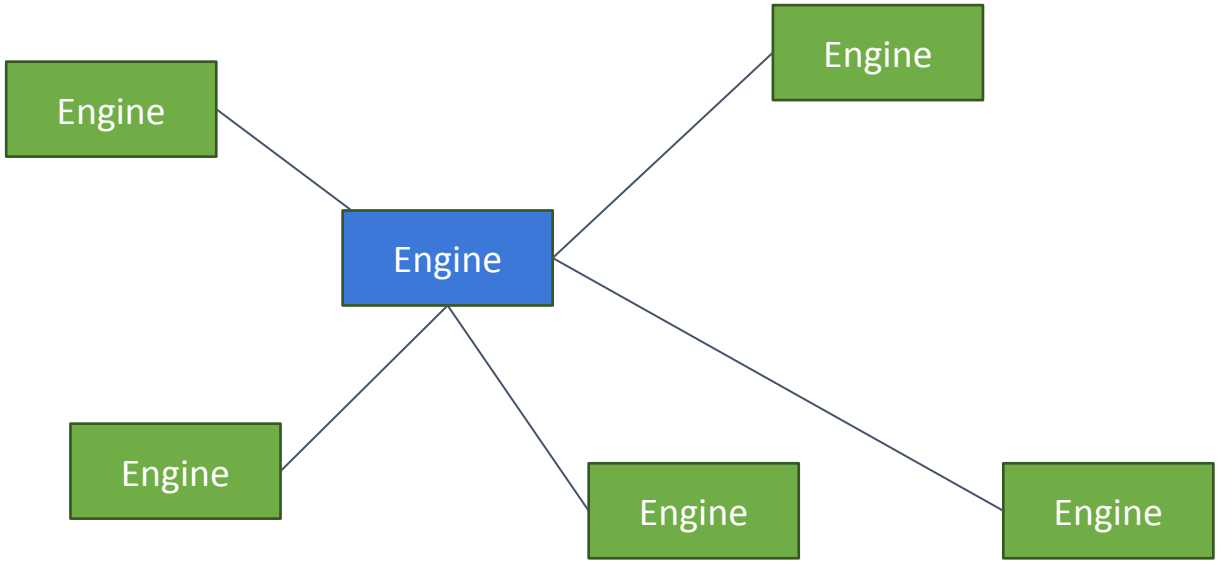
Swarm Mode




```
$ docker swarm init
```

```
$ docker swarm join <IP of manager>:2377
```

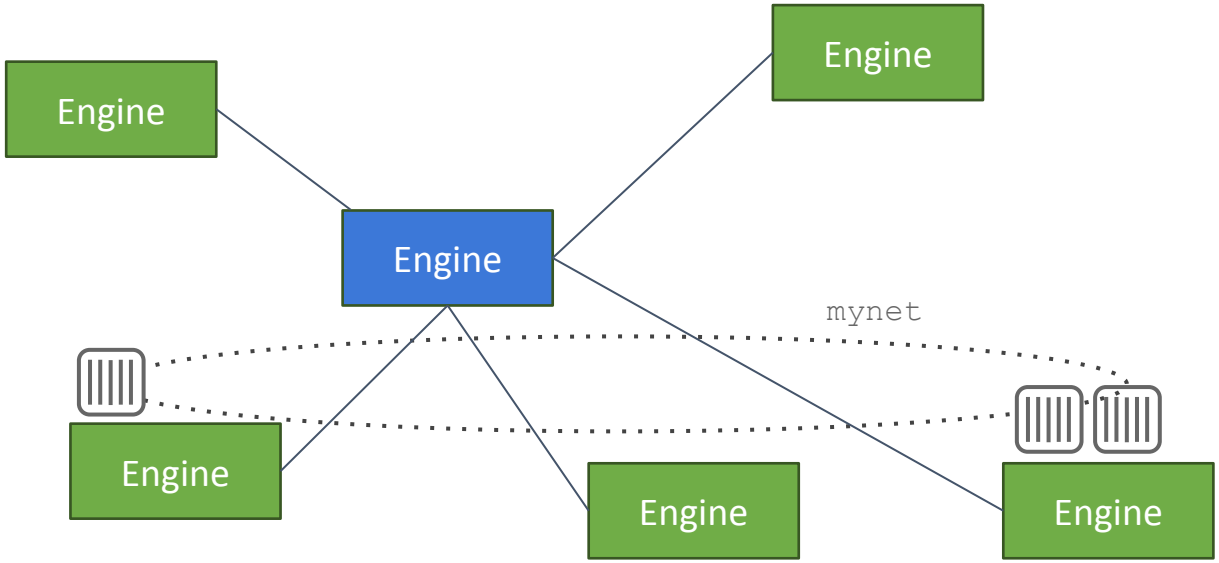
Swarm Mode




```
$ docker swarm init
```

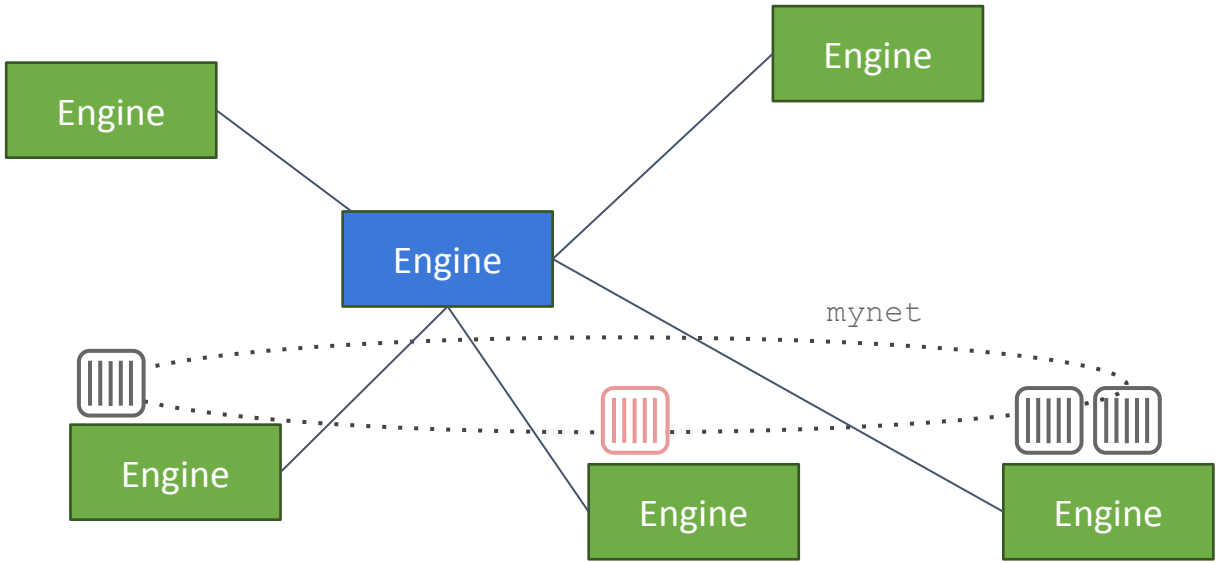
```
$ docker swarm join <IP of manager>:2377
```

สร้าง Services




```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 80:80/tcp frontend_image:latest
```

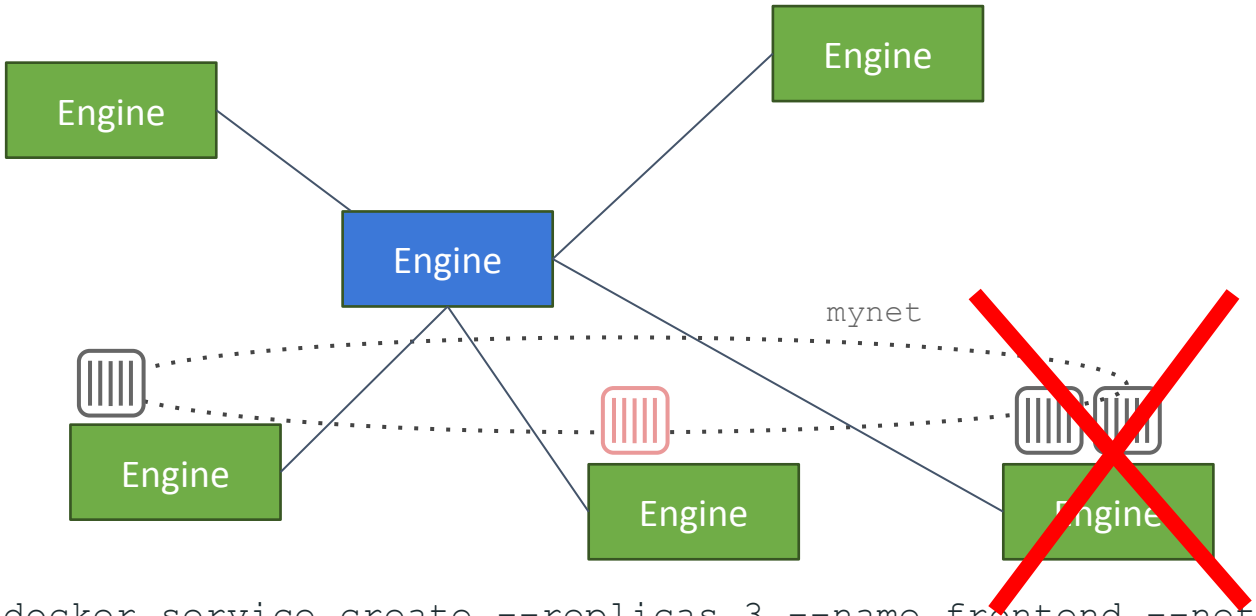
ถ้า Node ล่ม




```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 80:80/tcp frontend_image:latest
```

```
$ docker service create --name redis --network mynet redis:latest
```

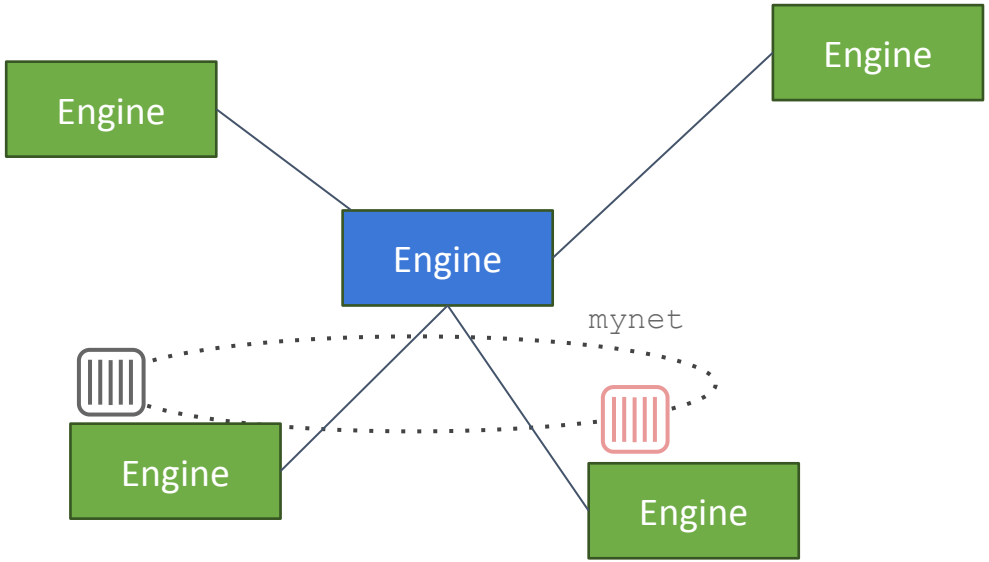
ถ้า Node ล่ม




```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 80:80/tcp frontend_image:latest
```

```
$ docker service create --name redis --network mynet redis:latest
```

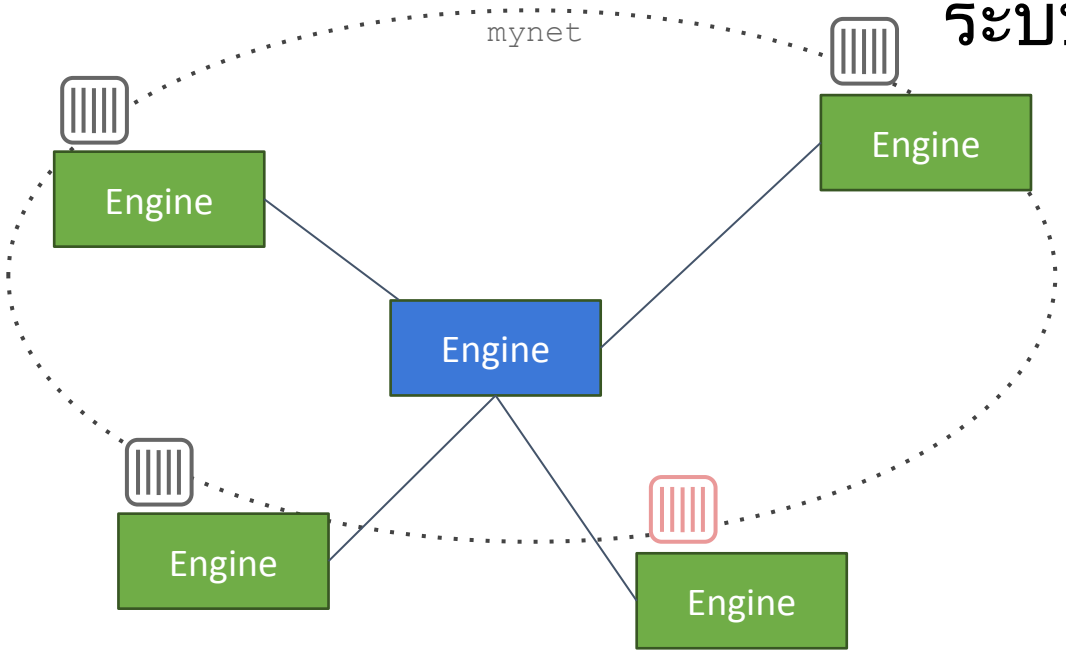
สถานะที่ต้องการ ≠ สถานะจริง




```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 80:80/tcp frontend_image:latest
```

```
$ docker service create --name redis --network mynet redis:latest
```

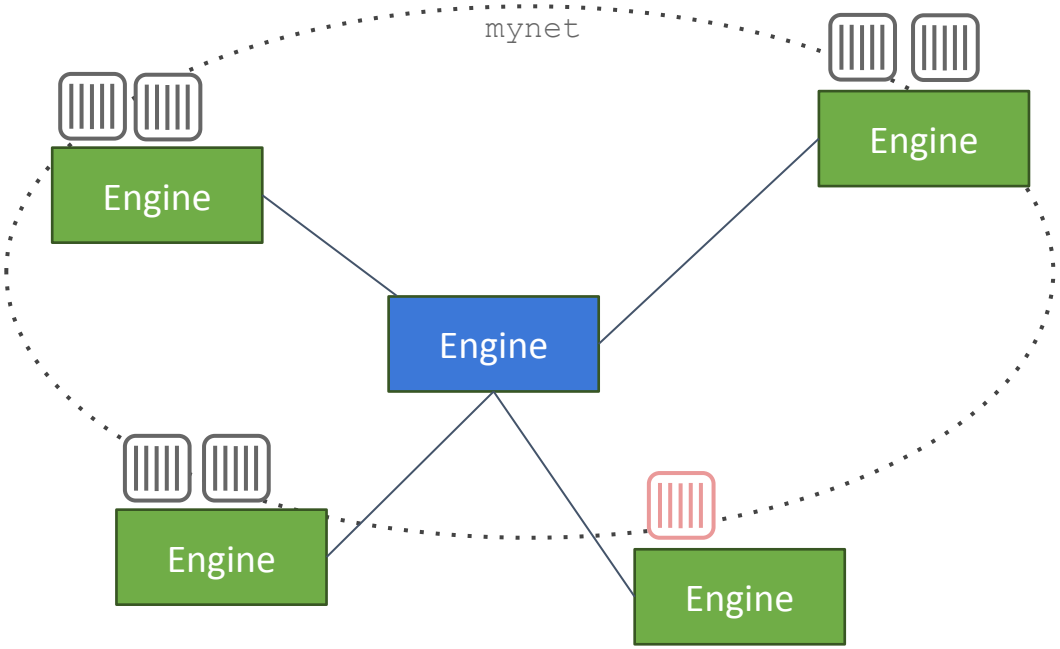
ระบบจะพยายามปรับกลับ




```
$ docker service create --replicas 3 --name frontend --network mynet
 --publish 80:80/tcp frontend_image:latest
```

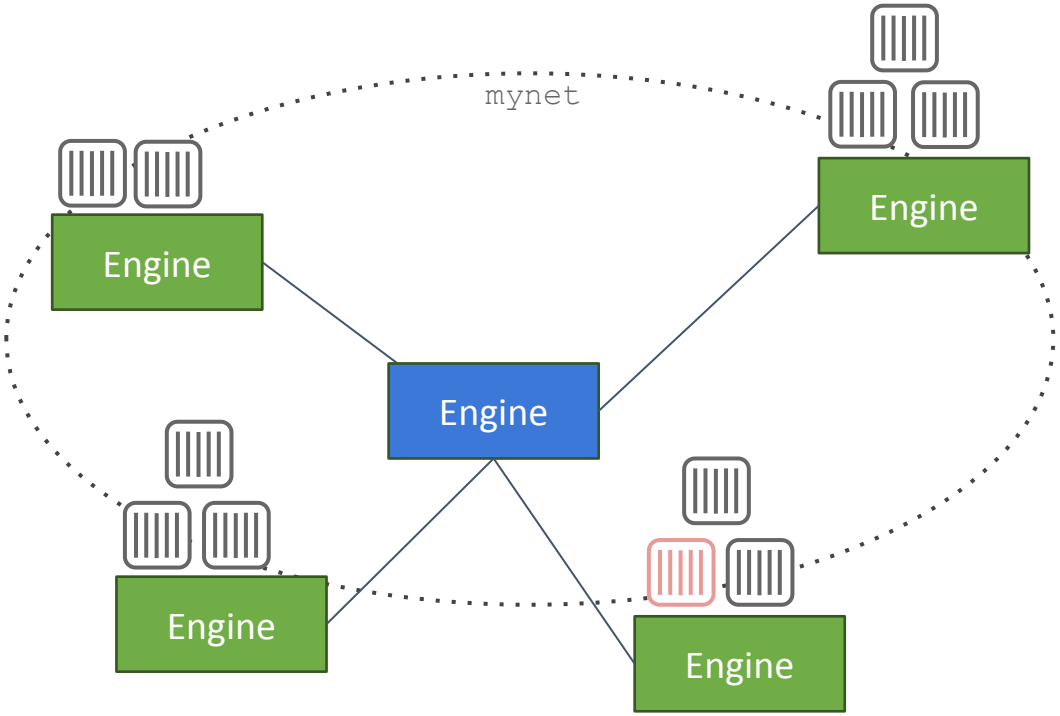
```
$ docker service create --name redis --network mynet redis:latest
```

การ Scale



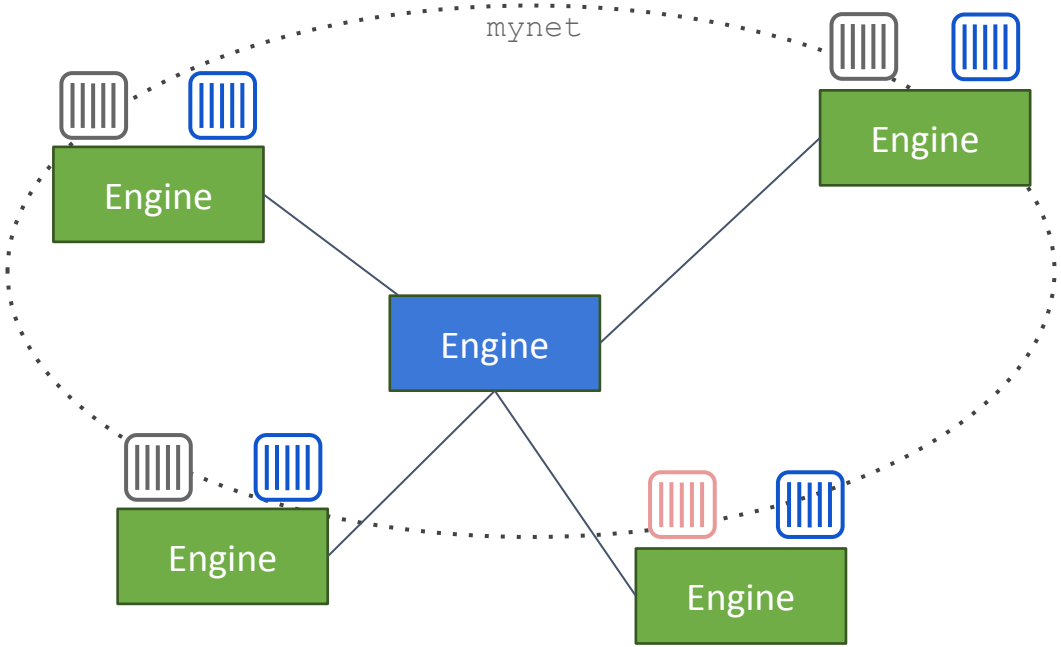

```
$ docker service scale frontend=6
```

การ Scale



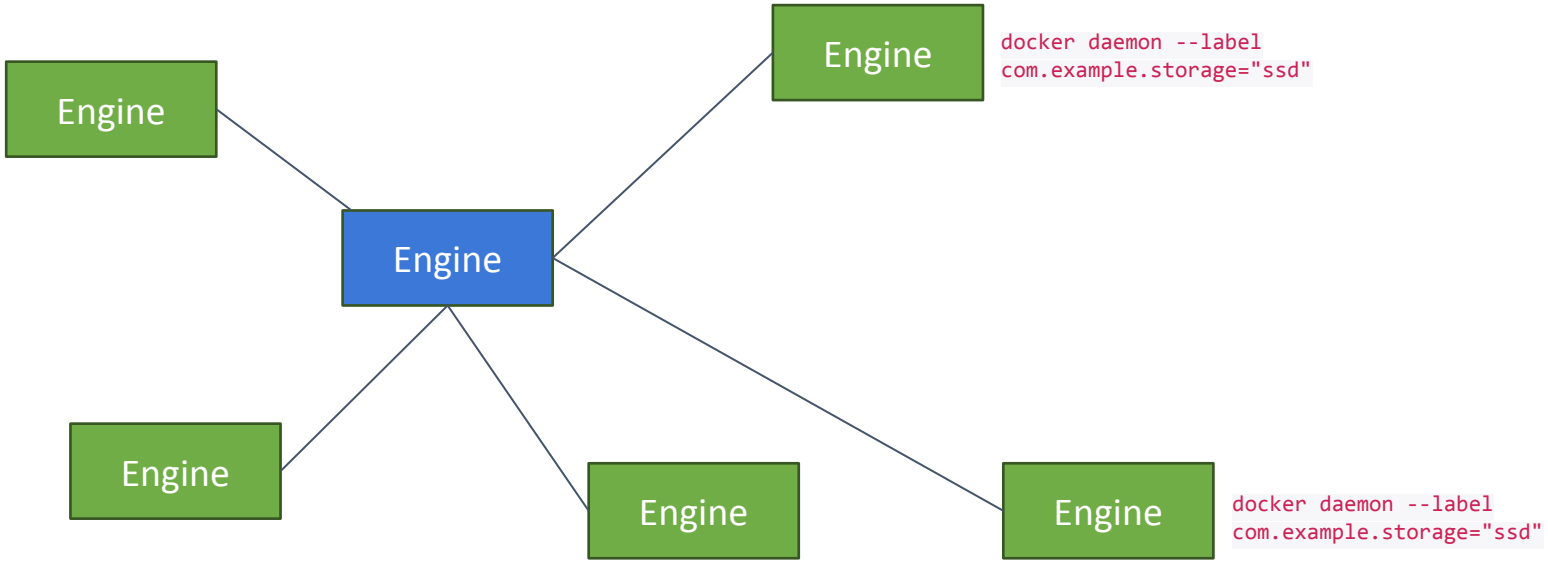

```
$ docker service scale frontend=10
```

Global Service

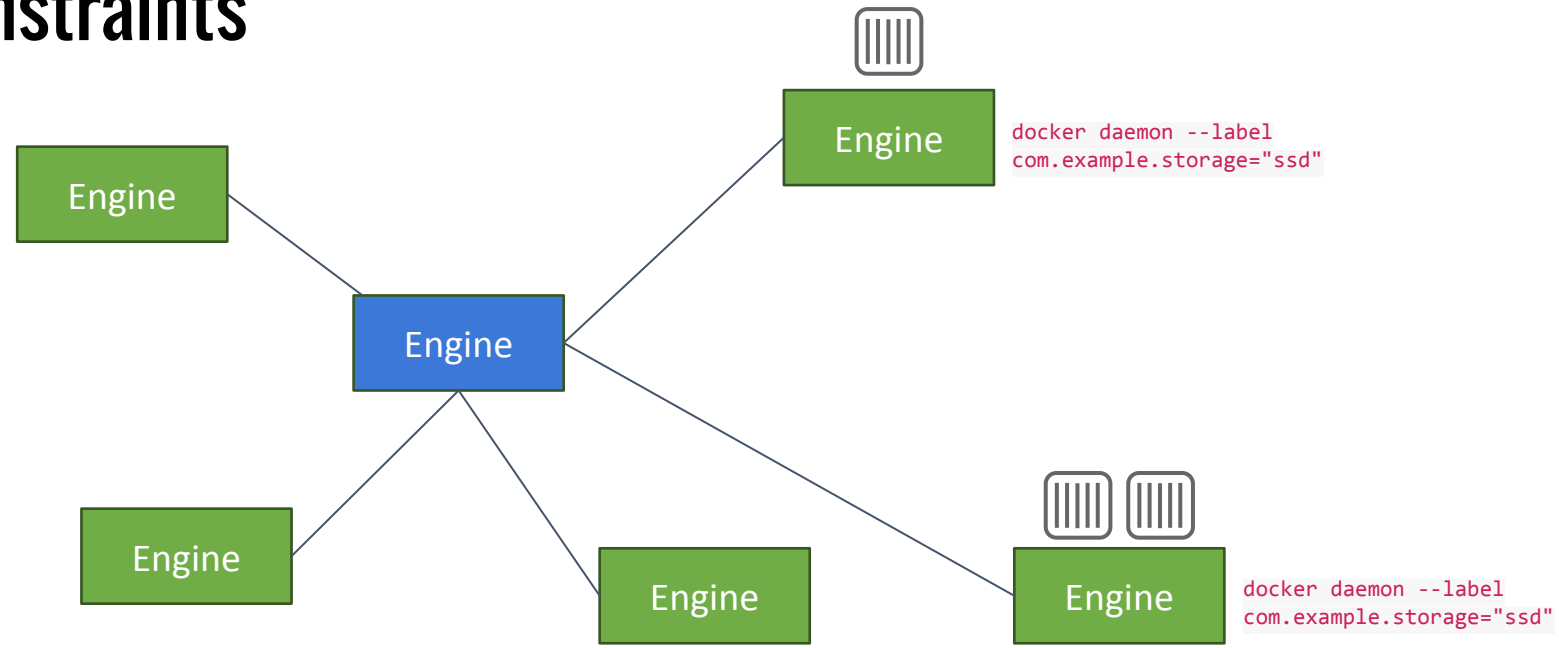



```
$ docker service create --mode=global --name prometheus
prom/prometheus
```

Constraints

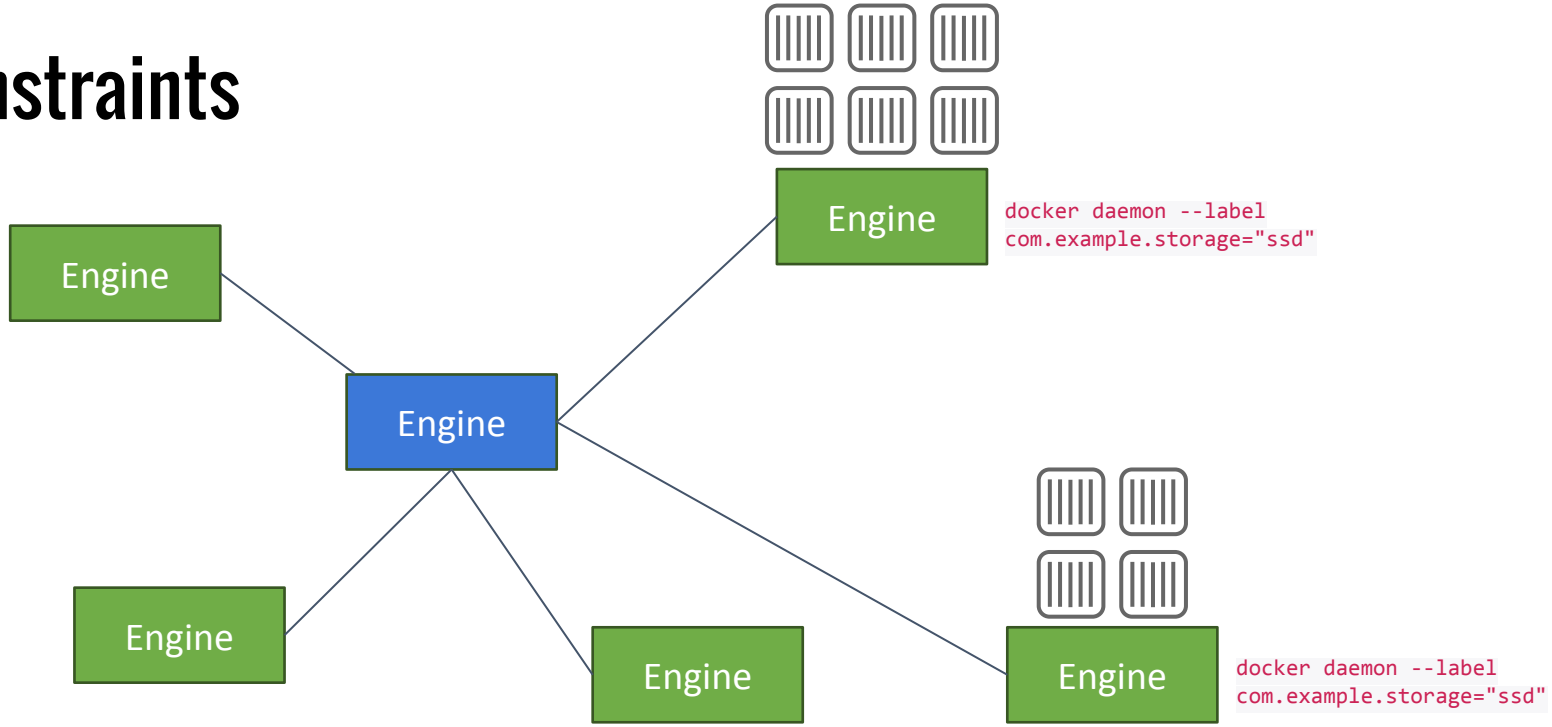


Constraints



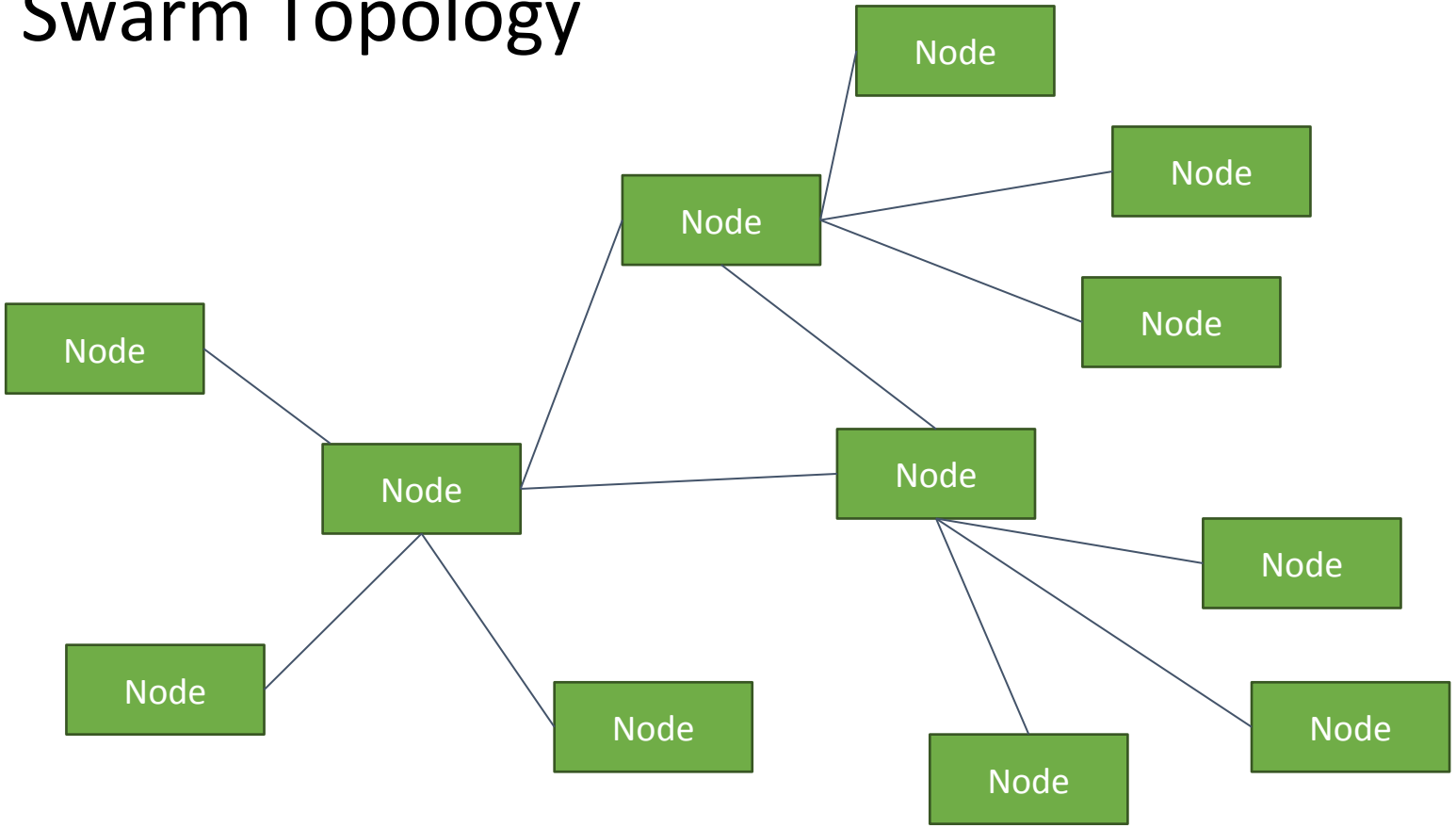

```
$ docker service create --replicas 3 --name frontend --network mynet
 --publish 80:80/tcp --constraint com.example.storage="ssd"
frontend_image:latest
```

Constraints

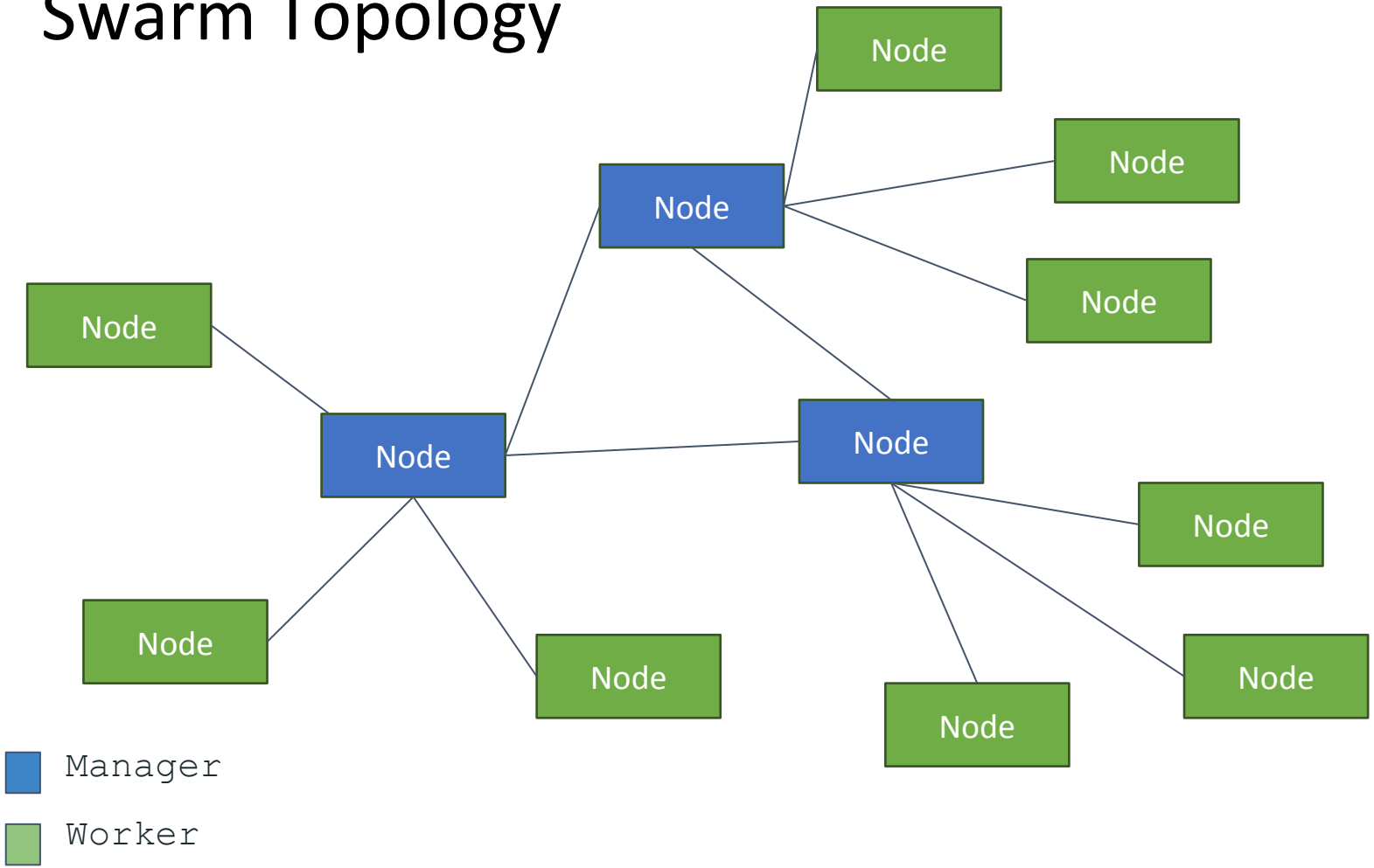



```
$ docker service create --replicas 3 --name frontend --network mynet
 --publish 80:80/tcp --constraint com.example.storage="ssd"
frontend_image:latest
$ docker service scale frontend=10
```

Swarm Topology

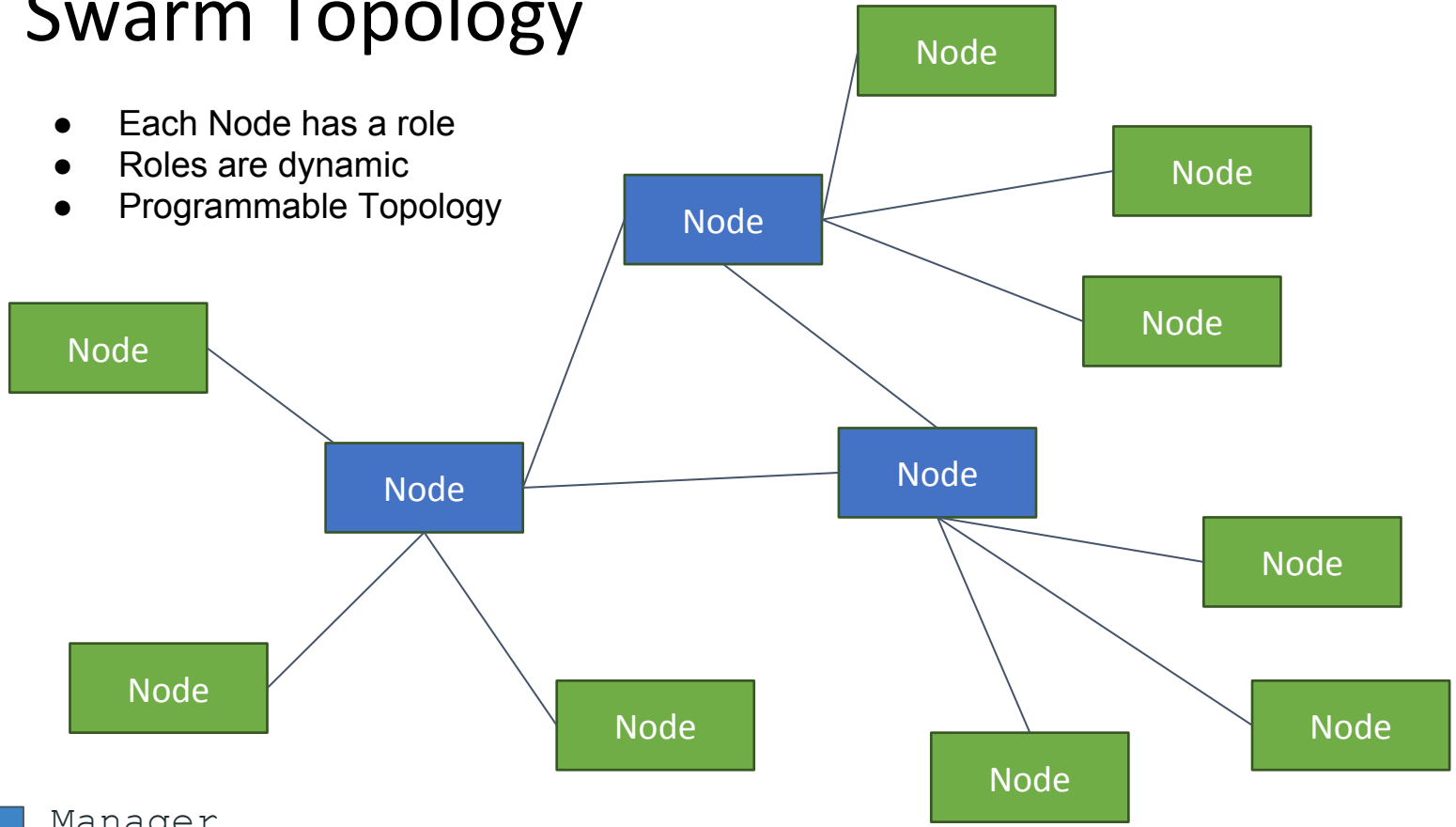


Swarm Topology

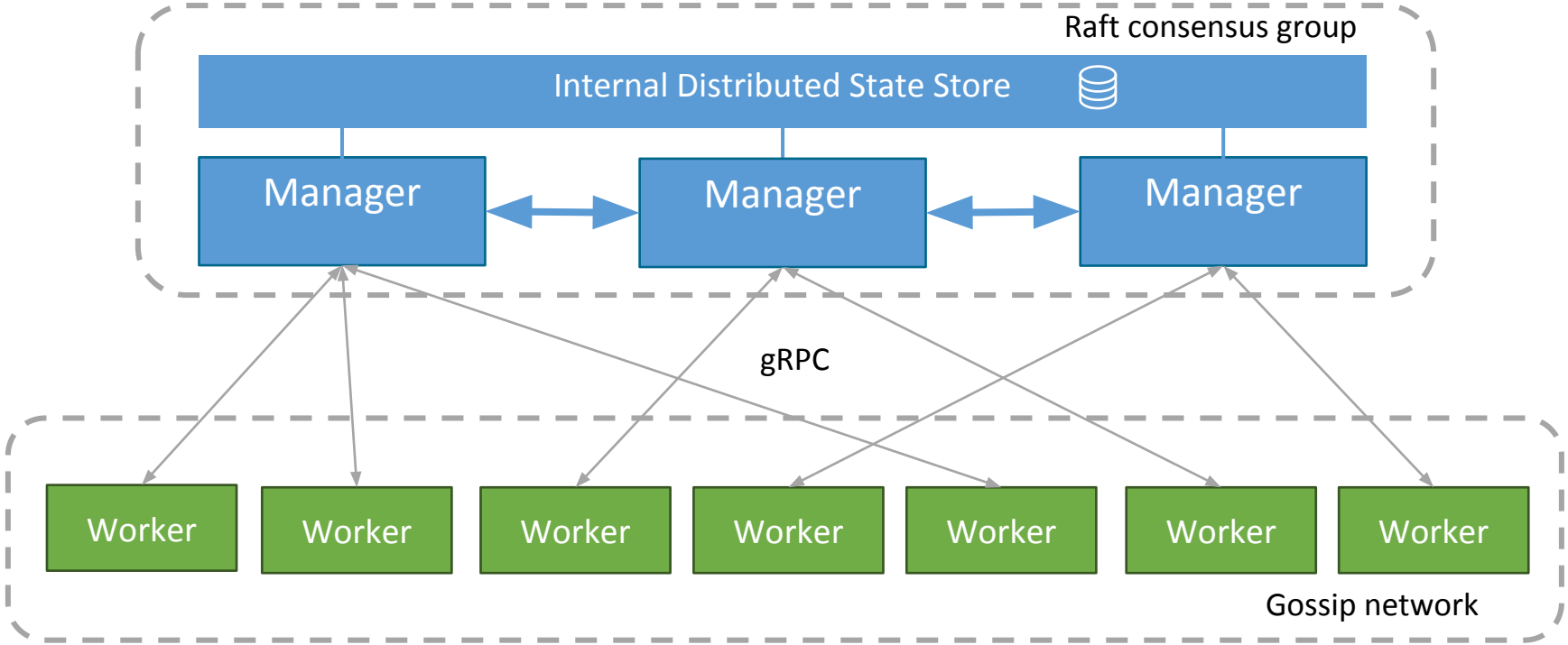


Swarm Topology

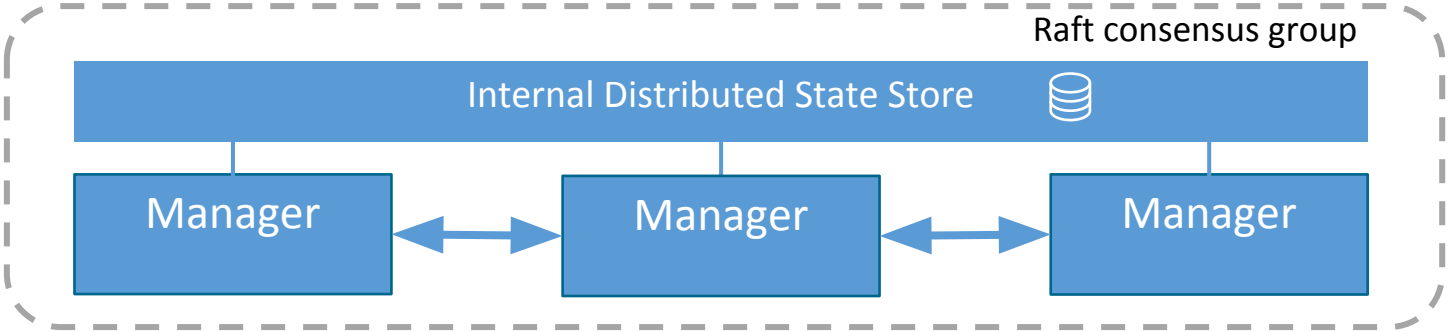
- Each Node has a role
- Roles are dynamic
- Programmable Topology



Docker Swarm Communication Internals

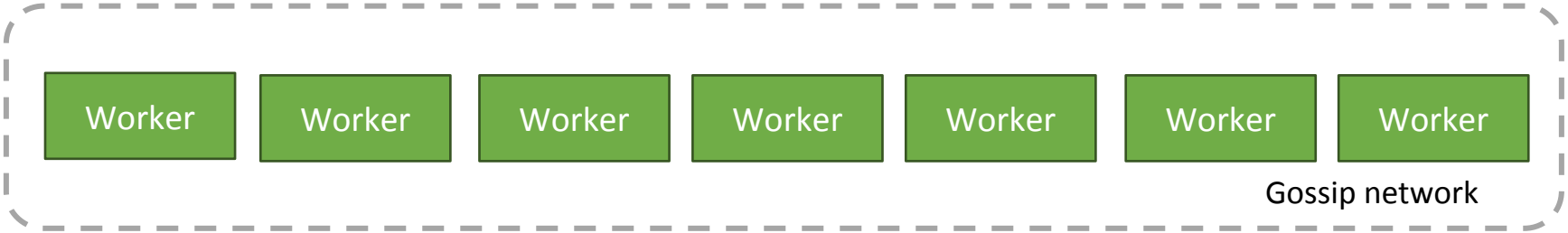


Quorum Layer



- Strongly consistent: Holds desired state
- Simple to operate
- Blazing fast (in-memory reads, domain specific indexing, ...)
- Secure

Worker-to-Worker Gossip

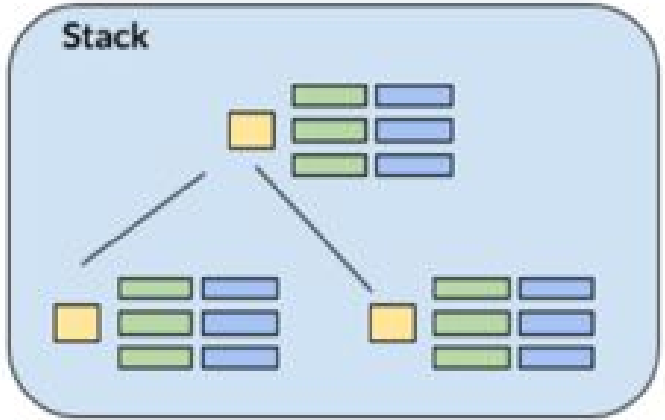


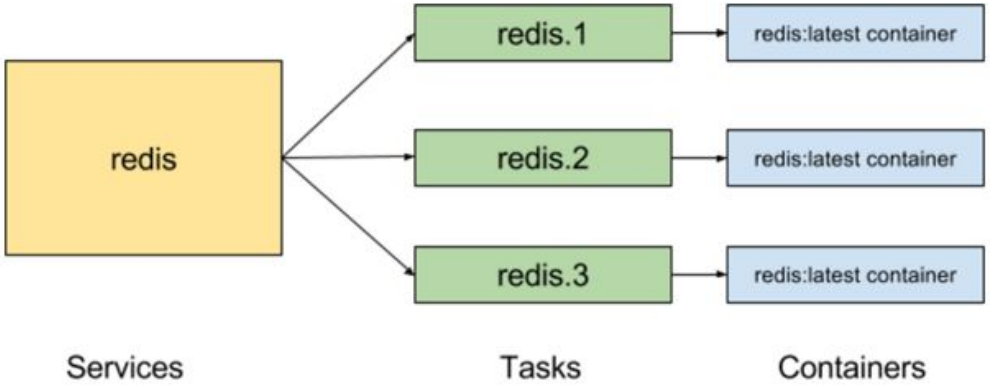
- Eventually consistent: Routing mesh, load balancing rules, ...
- High volume, p2p network between workers
- Secure: Symmetric encryption with key rotation in Raft

Services and Tasks

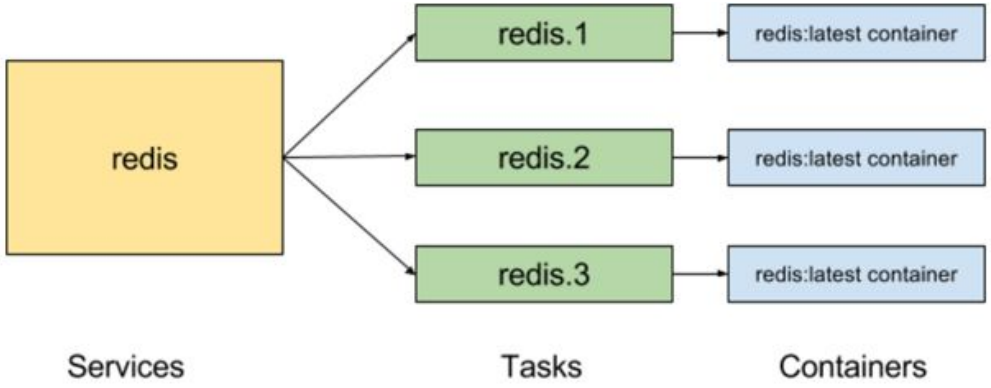
คอนเซ็ปใหม่ใน Docker 1.12+

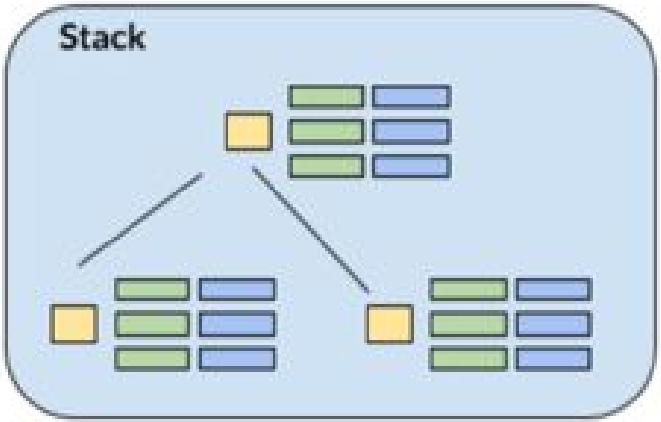
- docker service create
- 1 service มี N tasks



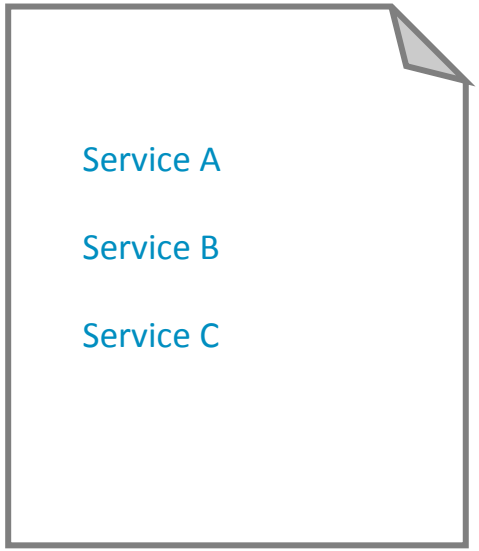


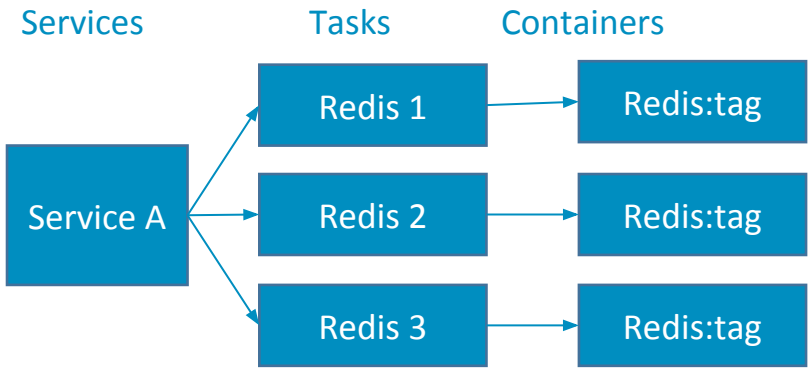
Services are grouped into stacks





Distributed Application Bundle (.dab) declares a stack

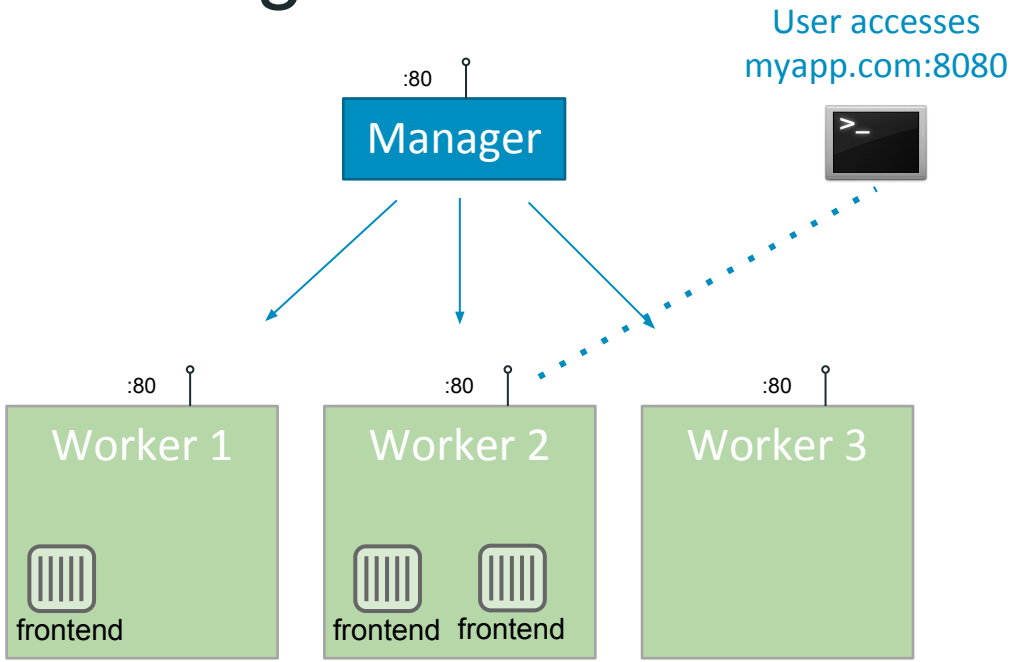




Swarm mode orchestration is optional

- You don’t have to use it
- 1.12 is fully backwards compatible
- Will not break existing deployments and scripts

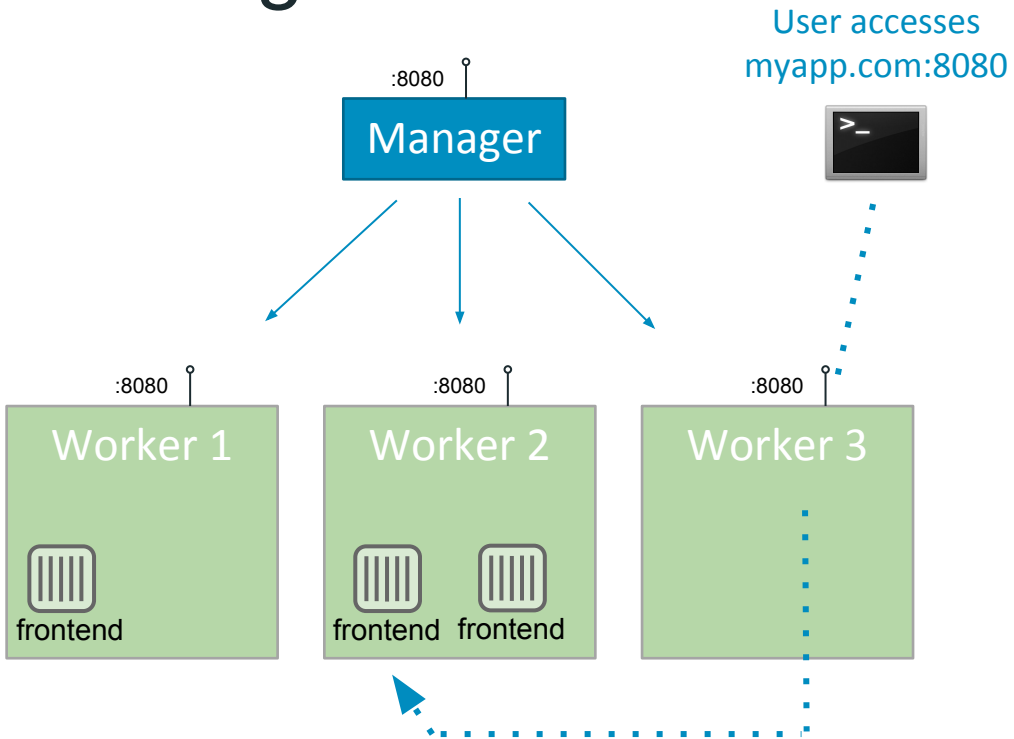
Routing Mesh



- Operator reserves a swarm-wide ingress port (80) for `myapp`
- Every node listens on 80
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 80:80/tcp frontend_image:latest
```

Routing Mesh: Published Ports



- Operator reserves a swarm-wide ingress port (8080) for `myapp`
- Every node listens on 8080
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 8080:80/tcp frontend_image:latest
```

Security out of the box

- Cryptographic Node Identity
  - Workload segregation (think PCI)
- There is no “insecure mode”:
  - TLS mutual auth
  - TLS encryption
  - Certificate rotation

Container Health Check in `Dockerfile`

```
HEALTHCHECK --interval=5m --timeout=3s
  --retries 3
  CMD curl -f http://localhost/ || exit 1
```

Checks every 5 minutes that web server can return index page within 3 seconds.

Three consecutive failures puts container in an unhealthy state.

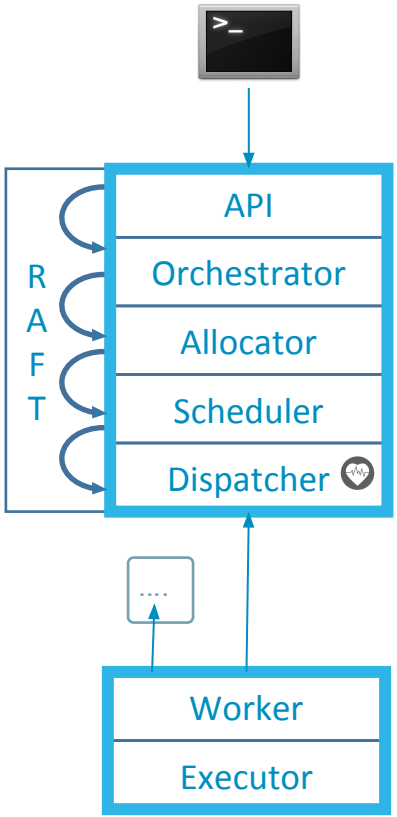
New Plugin Subcommands (Experimental)

```
docker plugin install tiborvass/no-remove

docker plugin enable no-remove

docker plugin disable no-remove
```

Plugin Permissions Model

```
$ docker plugin install tiborvass/no-remove
Plugin "mikegoelzer/myplugin:latest"
requested the following privileges:
 - Networking: host
 - Mounting host path: /data
Do you grant the above permissions? [y/N]
```

Node Breakdown



docker service create

Manager Node

Accepts command from client and creates service object

Reconciliation loop for service objects and creates tasks

Allocates IP addresses to tasks

Assigns nodes to tasks

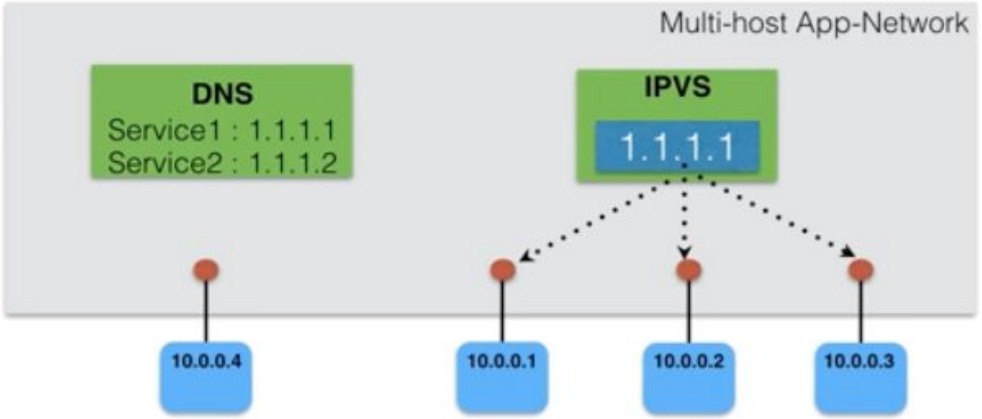
Checks in on workers

Worker Node

Connects to dispatcher to check on assigned tasks

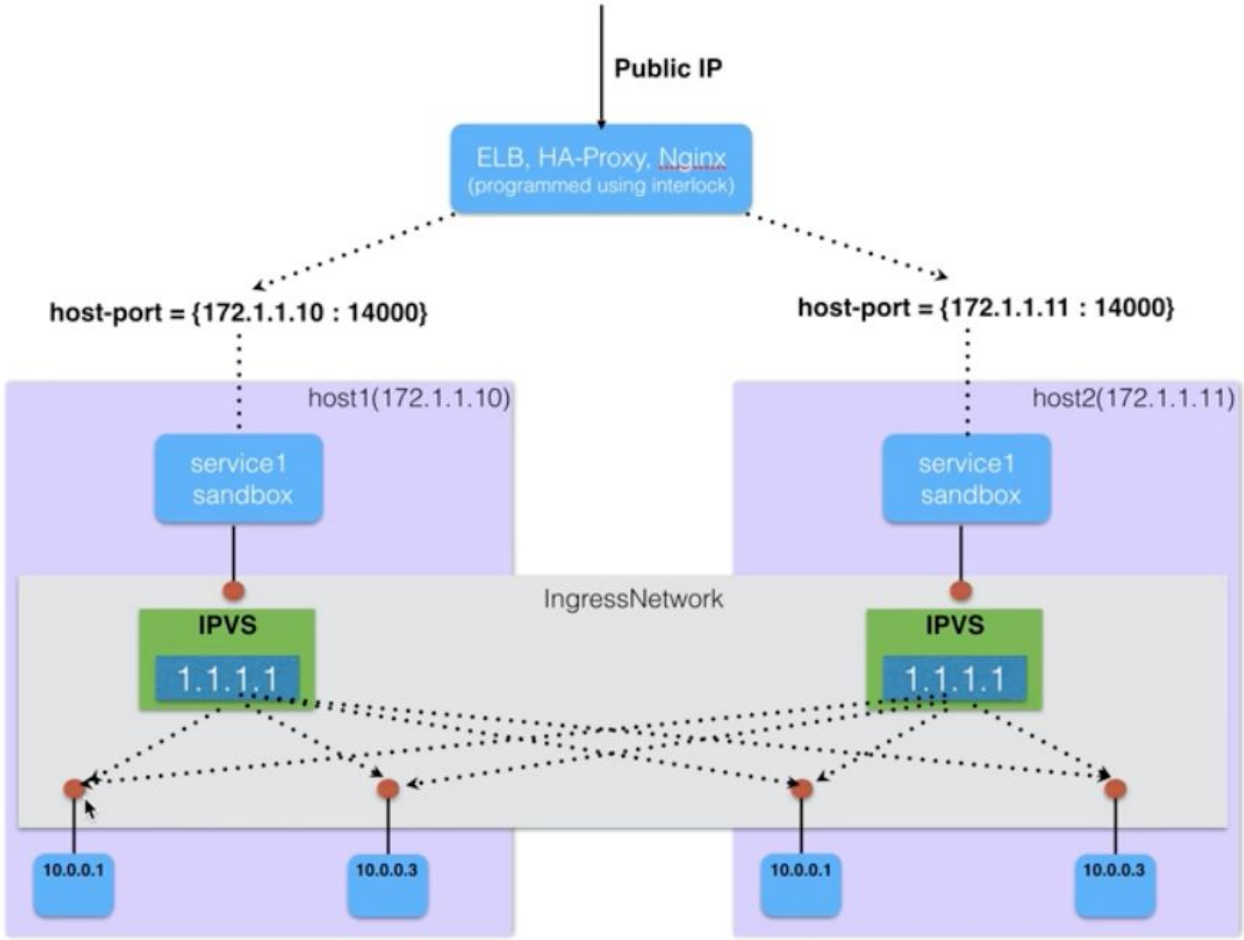
Executes the tasks assigned to worker node

Internal Load-Balancer

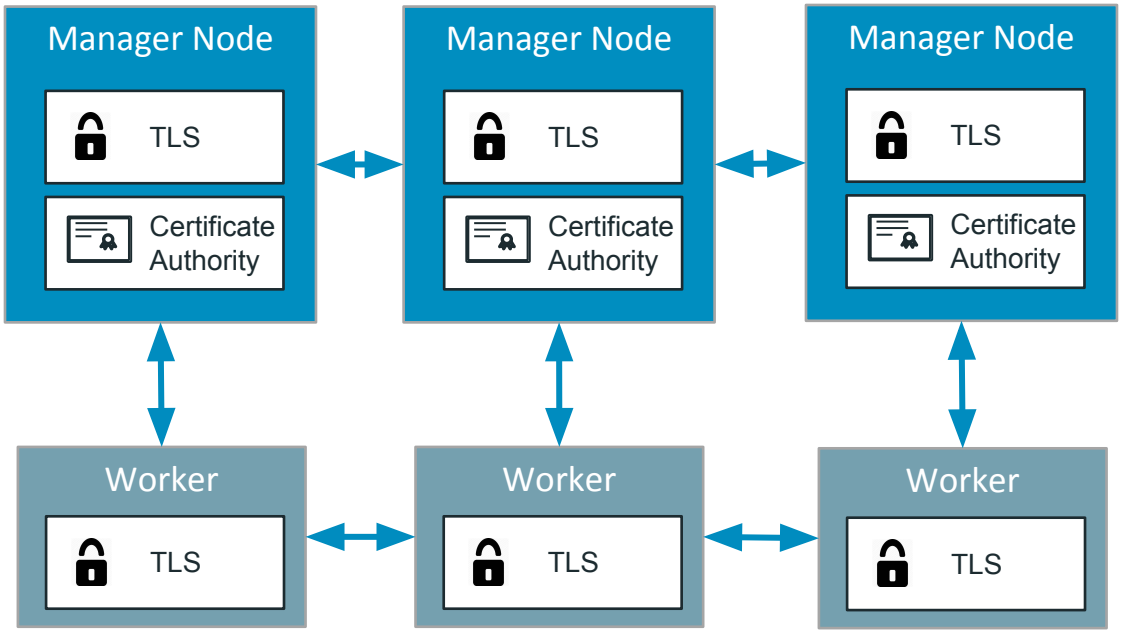


- **Load-balancer is designed as an integral part of CNM**
  - Works on top of CNM constructs (network, endpoint, sandbox, SD)
  - Every Service gets a Virtual-IP
  - Built-in SD resolves Service-Name -> VIP
  - Service VIP -> Container IP load balancing achieved using IPVS

Ingress Load-Balancer



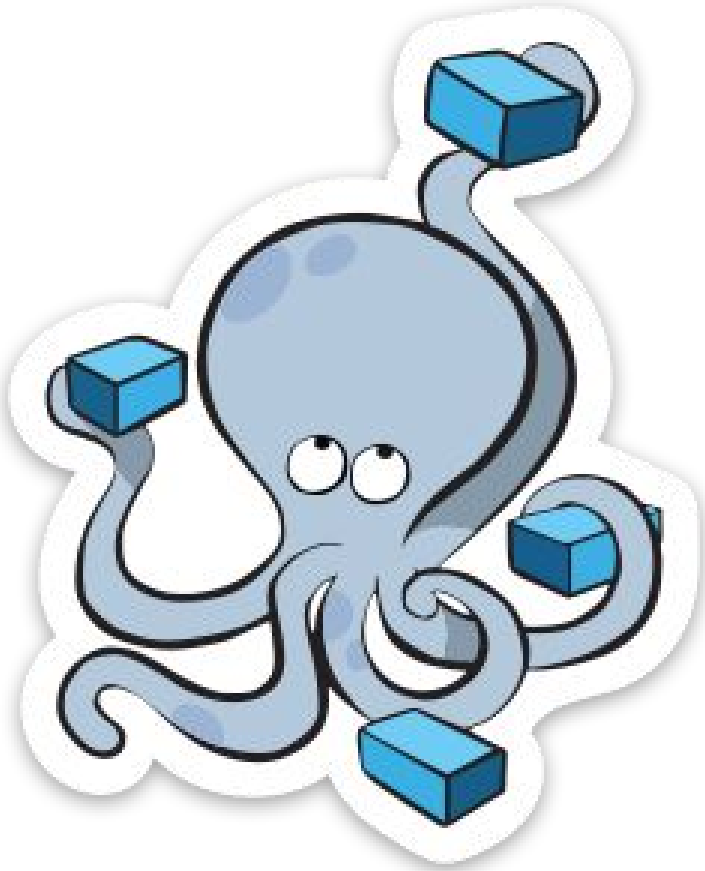
Secure by default with end to end encryption



- Cryptographic node identity
- Automatic encryption and mutual auth (TLS)
- Automatic cert rotation
- External CA integration

Docker Compose บน Docker Network

Docker Compose



Docker Compose

- จัดเอา **container** มาโยงกันเป็นกลุ่ม
- เพื่ออธิบาย **application**
- เช่น

app ประกอบไปด้วย **front** โยงหา **web** และมี **db** เป็นตัวเก็บข้อมูล

- การ **scale** ควรทำในระดับ **app**
    - เป็น **best practice** จาก **concept** ของ **Pod** ใน **Kubernetes**

Docker Compose

```
// เตรียมแอพพลิเคชัน “app”

version: 2


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

- จะได้ Network ชื่อ “app_default”
- web และ db จะเข้าไปอยู่ด้วยกันใน Network นี้
- ยกตัวอย่าง เช่น web สามารถโยงหา Dabase ได้โดยอ้างถึง postgres://db:5432

Docker Compose

// เตรียมแอพพลิเคชัน “app”

```
version: 2
services:
  proxy:
    build: ./proxy
    networks:
      - front
  web:
    build: ./web
    networks:
      - front
      - back
  db:
    image: postgres
    networks:
      - back
```

```
networks:
  front:
    driver: overlay
  back:
    driver: calico-ipam-driver
    driver_opts:
      foo: "1"
      bar: "2"
```

Docker Compose - default network

```
version: 2

services:
  web:
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres

networks:
  default:
    driver: overlay
```

- ระบุ default Network ให้ app

Docker Compose - external network

```
version: 2

services:
  web:
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres

networks:
  default:
    external:
      name: multihost
```

- สร้าง Network “multihost” ไว้แล้ว
- service ทั้งหมดใน app นี้ใช้ “multihost” เป็น default Network

Up and Run

เตรียมไฟล์ docker-compose.yml

$ cd app

$ docker-compose up -d

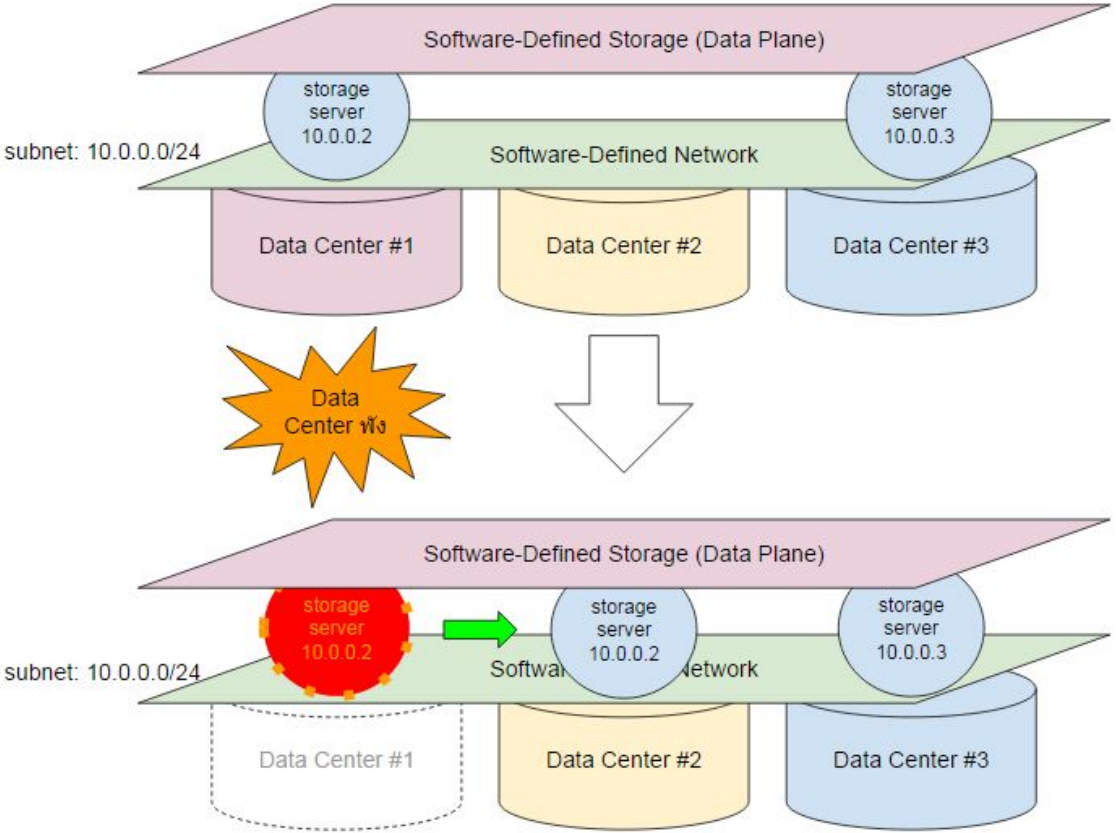
Docker Compose
Hands On

Discovery Service

File: compose.yml

```
consul:
  image: "progrium/consul:latest"
  container_name: "consul"
  hostname: "consul"
  ports:
    - "8400:8400"
    - "8500:8500"
    - "53:53"
  command: "-server -bootstrap-expect 1 -ui-dir /ui"
```

Towards Software-Defined Infrastructure



Question ?