Linux Containers and Dockers

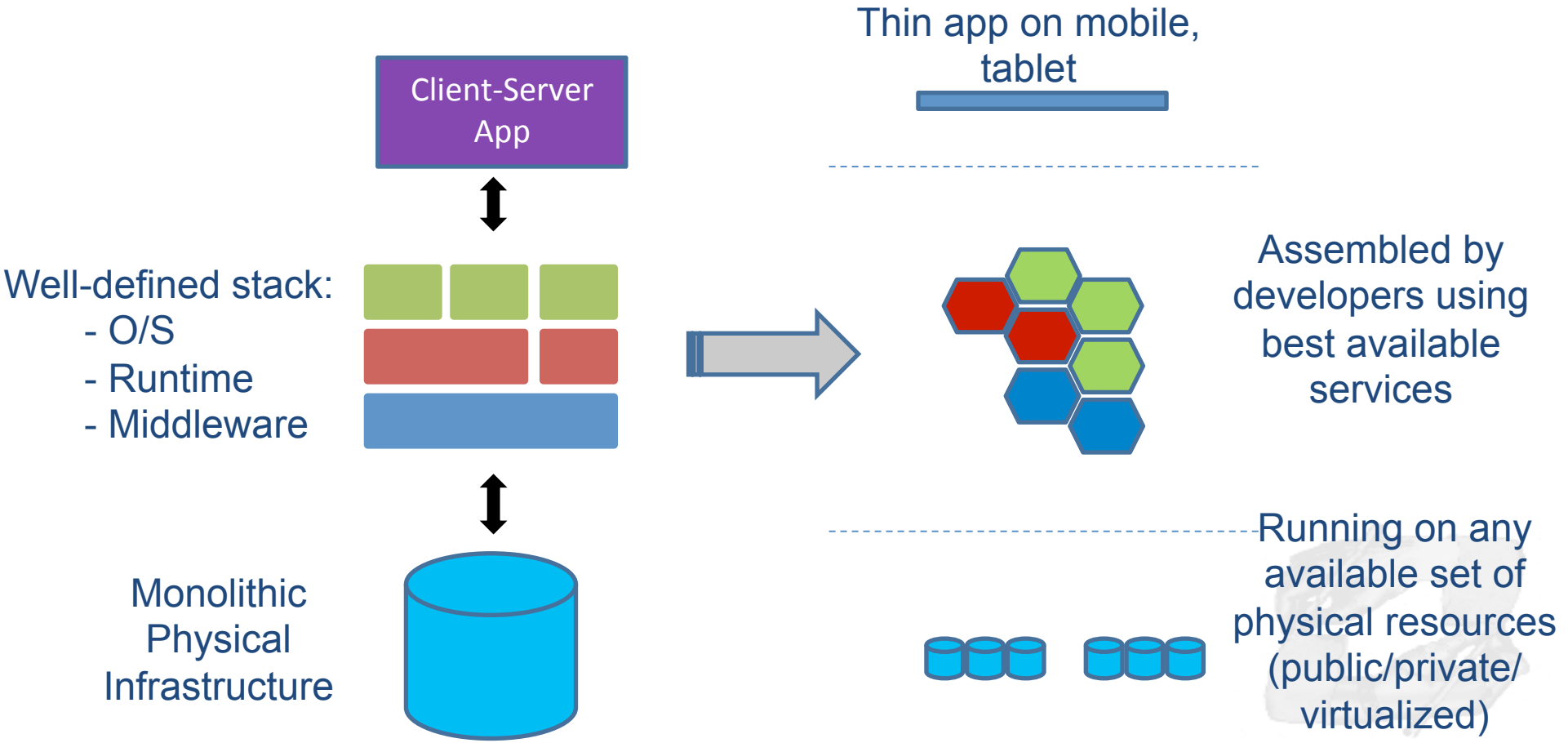
When, Pros and Cons

Dr. Fabio Fumarola

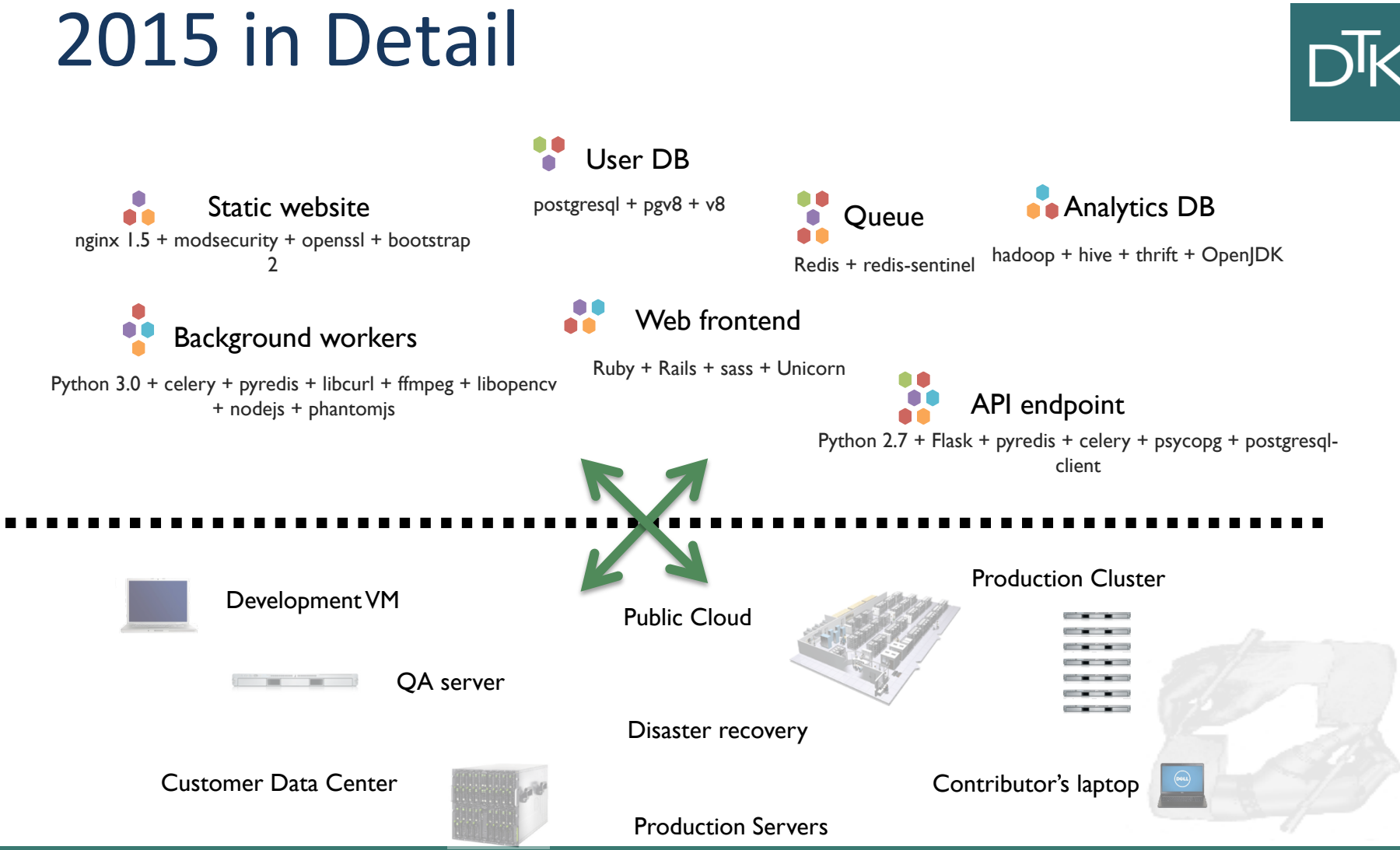
Contents

- The Evolution of IT
- The Solutions: Virtual Machines vs Vagrant vs Docker
- Differences
- Examples: Vagrant, Boot2Docker, Docker, Docker Hub
- Orchestrate Docker
- Mesosphere
- CoreOS

From 1995 to 2015



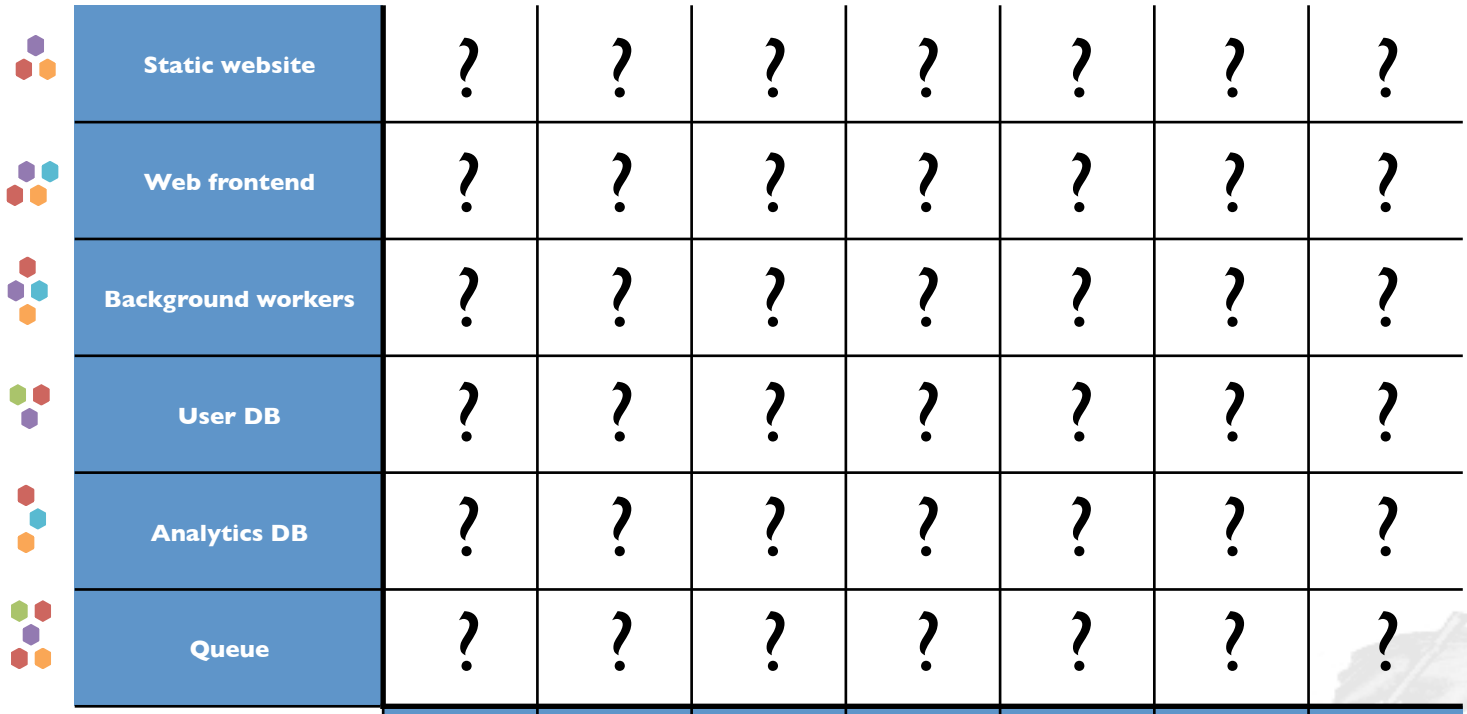
2015 in Detail



Challenges

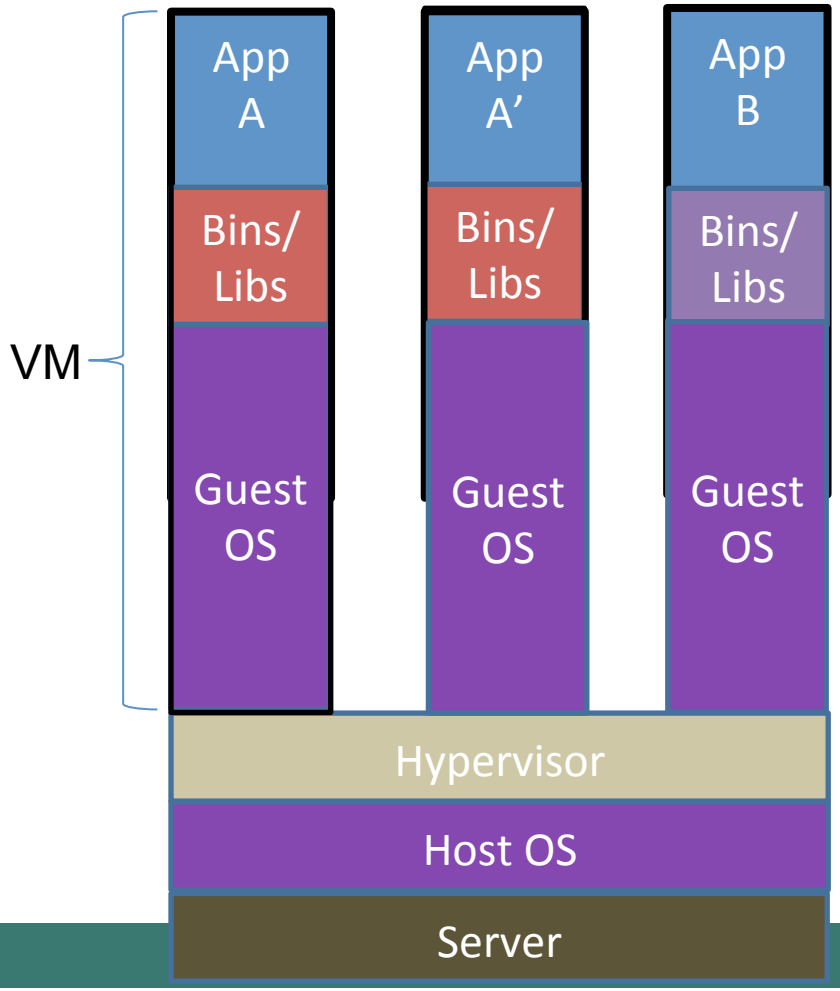
- How to ensure that services interact consistently?
- How to avoid to setup N different configurations and dependencies for each service?
- How to migrate and scale quickly ensuring compatibility?
- How to replicate my VM and services quickly?

How to deal with different confs?



1. VIRTUAL MACHINES

Virtual Machines



- Run on top of an Hypervisor

Pros

- fully virtualized OS
- Totally isolated

Cons

- Needs to take a snapshot of the entire VM to replicate
- Uses a lot of space
- Slow to move around

Hypervisors Trend

2011

- XEN: Default choice given Rackspace and Amazon use
- KVM: Bleeding edge users

2012

- KVM: Emerges as the lead
- XEN: Loses momentum

Hipervisors Trend

2013

- KVM: Maintains lead (around 90%+ for Mirantis)
- Vmware: Emerges as a surprising second choice
- Containers (LXC, Parallels, Docker): Web Hosting and SAS focused
- Xen and HyperV: Infrequent requests (XenServer.org)

2014 – 2015

- ???

2. VAGRANT

Vagrant

- Open source VM manager released in 2010
- It allows you to script and package VMs config and the provisioning setup via a VagrantFile
- It is designed to run on top of almost any VM tool: VirtualBox, VMVare, AWS, OpenStack[1]
- It can be used together with provisioning tools such as shell scripts, Chef and Puppet.

1. https://github.com/cloudbau/vagrant-openstack-plugin

Vagrant: idea

Use a VagrantFile to install

1. an operating system
2. Required libraries and software

and finally run programs and processes of your final application

```
box      = 'precise32'
url      = 'http://files.vagrantup.com/precise32.box'
hostname = 'myprecisebox'
domain   = 'example.com'
ip       = '192.168.0.42'
ram      = '256'

Vagrant::Config.run do |config|
  config.vm.box = box
  config.vm.box_url = url
  config.vm.host_name = hostname + '.' + domain
  config.vm.network :hostonly, ip

  config.vm.customize [
    'modifyvm', :id,
    '--name', hostname,
    '--memory', ram
  ]
end
```

Vagrant: Feature

- Command-Line Interface
- Vagrant Share
- VagrantFile
- Boxes
- Provisioning
- Networking
- Synced Folders
- Multi-Machine
- Providers
- Plugins

https://www.vagrantup.com/downloads

Vagrant: Demo

- It allows us to interact with Vagrant
- It offers the following commands: box, connect, destroy, halt, init, login, package a vm, rdp, …

https://docs.vagrantup.com/v2/cli/index.html

Vagrant Example

1. Download and install VirtualBox and Vagrant

```
$ mkdir vagrant_first_vm && cd vagrant_first_vm
$ vagrant init
```

2. This will place a VagrantFile in the directory
3. Install a Box

```
$ vagrant box add ubuntu/trusty64
```

4. Using a Box -> https://vagrantcloud.com/

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
end
```

Vagran: Start

1. Start the box

```
$ vagrant up
```

2. Login into the vm

```
$ vagrant ssh
```

3. You can destroy the vm by

```
$ vagrant destroy
```

Vagrant: Synced Folders

- By default, it shares your project directory to the /vagrant directory on the guest machine.

```
$ vagrant up
$ vagrant ssh
$ ls /vagrant
--Vagrantfile
```

- If you create a file on your guest os the file will be on the vagrant vm.

```
$ touch pippo.txt
$vagrant ssh
$ls /vagrant/
```

Vagrant: Provisioning

- Let's install Apache via a boostrap.sh file

```
#!/usr/bin/env bash

apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /vagrant /var/www
```

- If you create a file on your gues os the file will be on the vagrant vm. (vagrant reload --provision)

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.provision :shell, path: "bootstrap.sh"
end
```

Vagrant: Networking

- **Port Forwarding**: llows you to specify ports on the guest machine to share via a port on the host machine

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, host: 4567, guest: 80
end
```

- By running vagrant reload or vagrant up we can see on http://127.0.0.1:4567 our apache
- It supports also bridge configurations and other configurations (https://docs.vagrantup.com/v2/networking/)

Vagrant: Share and Provider

- It is possible to share Vagrant box via vagrant cloud (but?)

Providers

- By default Vagrant is configured with VirtualBox but you can change the provider

```
$ vagrant up --provider=vmware_fusion
$ vagrant up --provider=aws
```

- How?

```
$ vagrant plugin install vagrant-aws
```

Vagrant: AWS Vagrantfile

```
Vagrant.configure("2") do |config|
# config.vm.box = "sean"

        config.vm.provider :aws do |aws, override|
        aws.access_key_id = "AAAAIIIIYYYY4444AAAA”
        aws.secret_access_key =
"c344441LooLLU322223526IabcdeQL12E34At3mm”
        aws.keypair_name = "iheavy"

        aws.ami = "ami-7747d01e"

        override.ssh.username = "ubuntu"
        override.ssh.private_key_path = "/var/root/iheavy_aws/pk-
XHHHHHMMMAABPEDEFGHOAOJH1QBH5324.pem"
        end
end
```

3. DOCKER

Quick Survey

- How many people have heard of Docker before this Seminar?
- How many people have tried Docker ?
- How many people are using Docker in production ?

What is Docker?

"With Docker, developers can build any app in any language using any toolchain. “Dockerized” apps are completely portable and can run anywhere - colleagues’ OS X and Windows laptops, QA servers running Ubuntu in the cloud, and production data center VMs running Red Hat.”

Docker.io

Docker in simple words

- It is a technology that allow you running applications inside containers (not VM)
- This assures that libraries and package needed by the application you run are always the same.
- This means you can make a container for Memcache and another for Redis and they will work the same in any OS (also in Vagrant).

Why Docker?

- Fast delivery of your applications
- Deploy and scale more easily
- Get higher density and run more workload
- Faster deployment makes for easier management

How does docker work?

- LinuX Containers (LXC)
- Control Groups & Namespaces (CGroups)
- AUFS
- Client – Server with an HTTP API

LXC- Linux Containers

- It is a user-space interface for the Linux kernel containment features
- Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.
- Currently LXC can apply the following kernel features to contain processes:
  - Kernel namespaces (ipc, uts, mount, pid, network and user)
  - Apparmor and SELinux profiles
  - Seccomp policies
  - Chroots (using pivot_root)
  - Kernel capabilities & Control groups (cgroups)

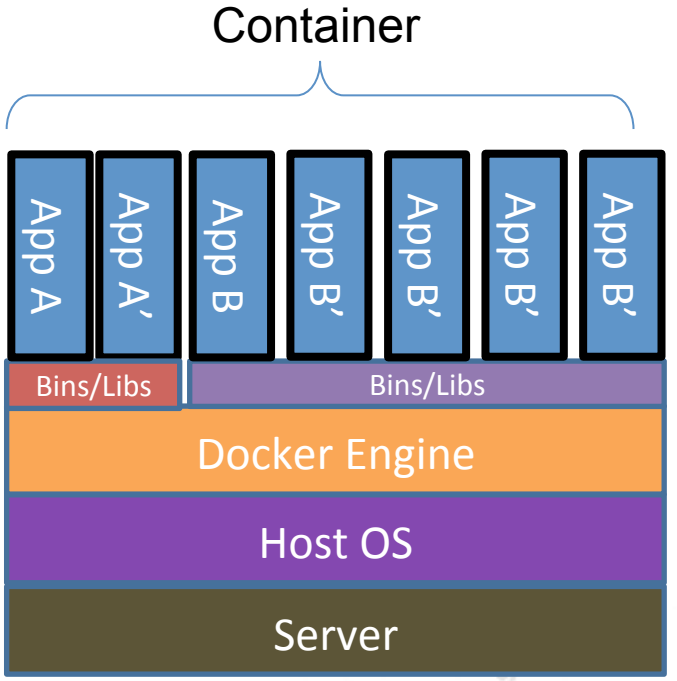
Cgroups

- Control groups is a Linux kernel feature to limit, account and isolate resource usage (CPU, memory, disk I/O, etc) of process groups.
- Features:
  - Resource limitation: limit CPU, memory…
  - Prioritization: assign more CPU etc to some groups.
  - Accounting: to measure the resource usage.
  - Control: freezing groups or check-pointing and restarting.

LCX based Containers

- It allows us to run a Linux system within another Linux system.
- A container is a group of processes on a Linux box, put together is an isolated environment.

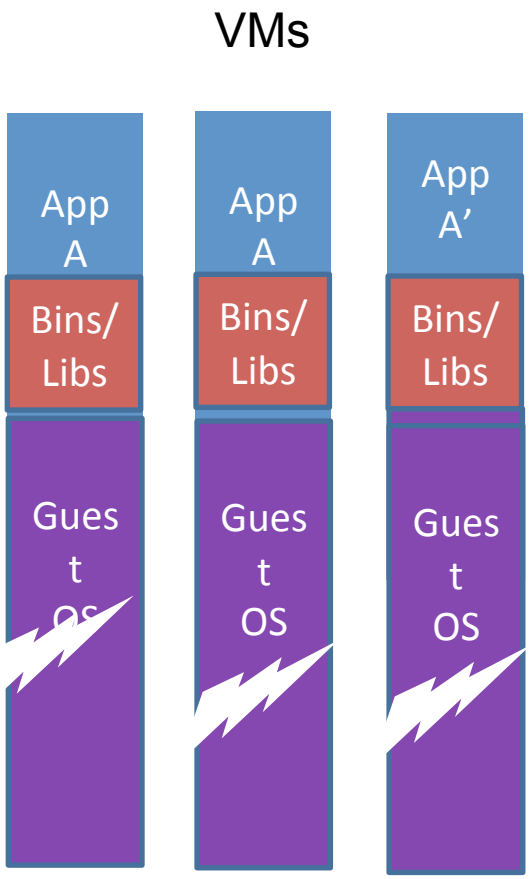
  - From the inside it looks like a VM
  - From the outside, it looks like normal processes

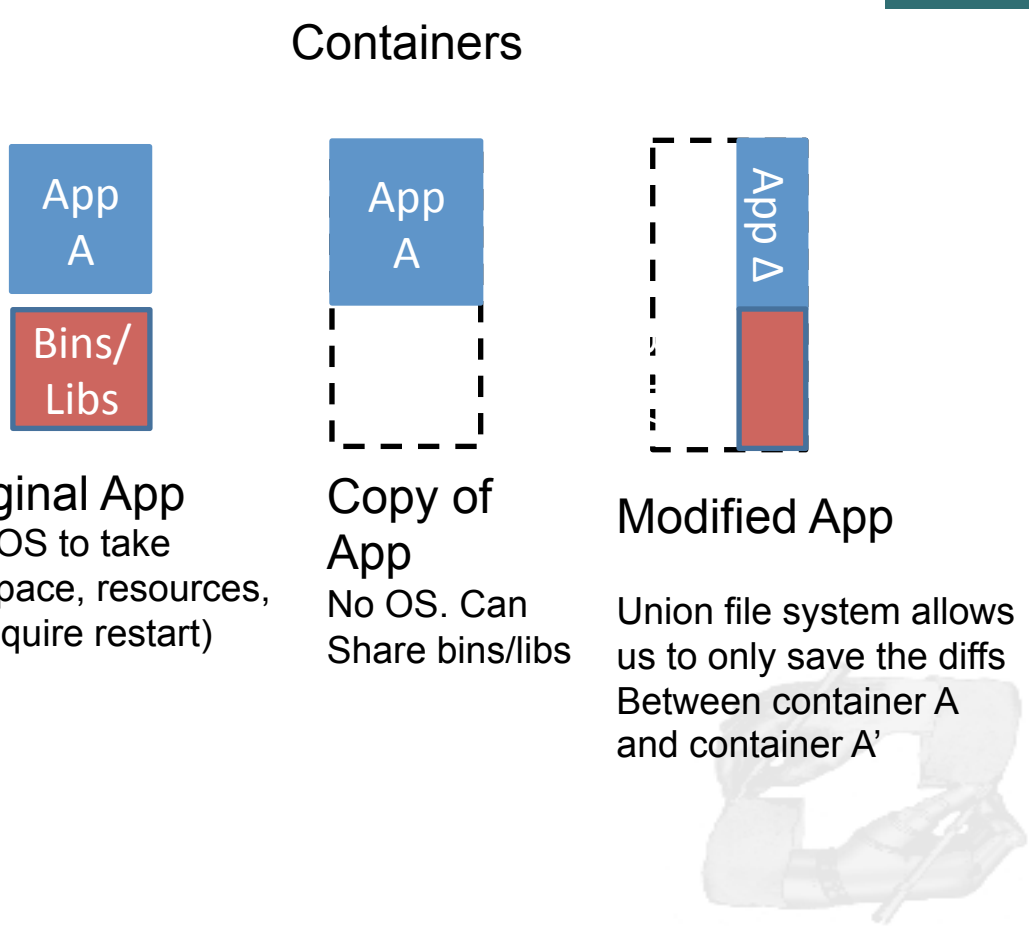


Docker Features

- VE (Virtual Environments) based on LXC
- Portable deployment across machines
- Versioning: docker include git-like capabilities for tracking versions of a container
- Component reuse: it allows building or stacking already created packages. You can create ‘base images’ and then running more machine based on the image.
- Shared libraries: there is a public repository with several images (https://registry.hub.docker.com/)

Why are Docker Containers lightweight?





Prerequisites

- I use Oh My Zsh[1] with the Docker plugin[2] for autocompletion of docker commands
- Linux at least with kernel 3.8 but 3.10.x is recommended
  - $ uname –r
- MacOS or Windows via Boot2Docker[3] or via Vagrant

1. https://github.com/robbyrussell/oh-my-zsh
2. https://github.com/robbyrussell/oh-my-zsh/wiki/Plugins#docker
3. http://boot2docker.io/

Docker Installation Ubuntu

- AUFS support

  $ sudo apt-get update

  $ sudo apt-get intall linux-image-extra-`uname –r`

- Add docker repo

  $ sudo sh –c “curl https://get.docker.io/gpg | apt-key add -”

  $ sudo sh –c “echo deb http://get.docker.io/ubuntu docker \

  main > /etc/apt/sources.list.d/docker.list”

- Install

  $ sudo apt-get update

  $ sudo apt-get install lxc-docker

Docker install Vagrant

- Create the folders

  $ mkdir ~/boot2docker

  $ cd ~/boot2docker

- Init the vagrant box

  $ vagrant init yungsang/boot2docker

  $ vagrant up; export DOCKER_HOST=tcp://localhost:2375

- Check docker

  $ docker version

* NOTE: the YungSang boot2docker opens up port forwarding to the network, so is not safe on public wifi.

Docker Installation Vagrant

- Clone the docker repository

  $ git clone https://github.com/dotcloud/docker.git

- Startup the vagrant image

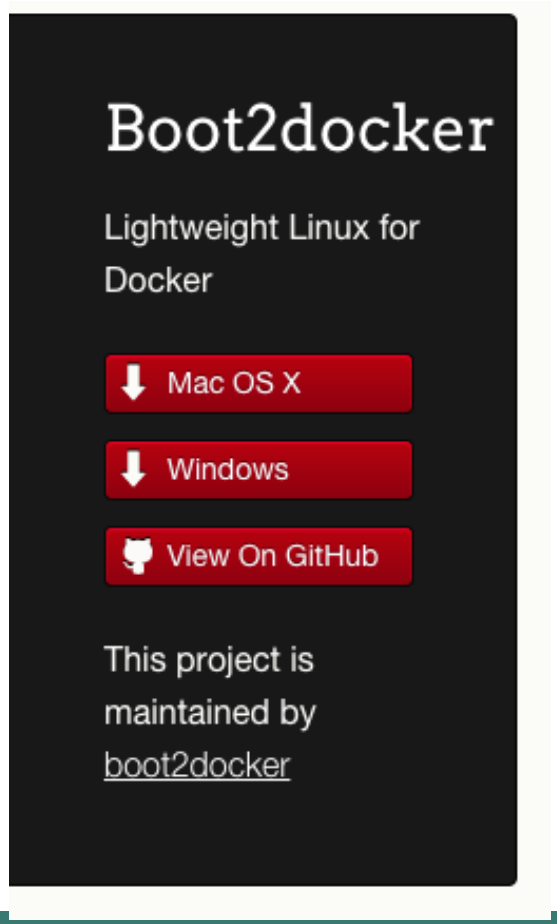
  $ vagrant up

- SSH into the image

  $ vagrant ssh

- Docker client works normally

Docker install boot2docker



http://boot2docker.io/

BASE COMMANDS

Docker: hello world

- Get one base image from https://registry.hub.docker.com

  $ sudo docker pull centos

- List images on your system

  $ sudo docker images

- Check the images
  - $ sudo docker images
- Run your first container
  - $ sudo docker run centos:latest echo “hello world”

An Interactive Container

- Run bash in your container
  - $ sudo docker run -t -i centos /bin/bash
- The -t flag assigns a pseudo-tty or terminal inside our new container
- The -i flag allows us to make an interactive connection by grabbing the standard in (STDIN) of the container
- We also specified a command for the container

A Daemonized Hello world

- Run a sh script
  - sudo docker run -d centos:6 /bin/sh –c ‘while true; do echo hello world; sleep 1; done’
- The -d flag tells Docker to run the container and put it in the background, to daemonize it.
- To list the docker containers running
  - $ docker ps
- To get the logs of the container
  - $ sudo docker logs container_id
- To stop the container:
  - $ sudo docker stop container_id

A web container with docker

- To run a Python Flask application
  - $ sudo docker run -d -P training/webapp python app.py
- The -P flag is new and tells Docker to map any required network ports inside our container to our host.
- To view our application with the port mapping
  - $ sudo docker ps –l
- We can see that the default flask port 5000 is exposed to 49155
  - $ sudo docker run -d -p 5000:5000 training/webapp python app.py
- Check the url to continue the guide
  - https://docs.docker.com/userguide/usingdocker/

Working with docker images

- To find images go to
  - https://hub.docker.com/
- To pull an image
  - $ sudo docker pull training/sinatra
- Updating and committing an image
  - $ sudo docker run -t -i training/sinatra /bin/bash
  - # gem install json
  - $ sudo docker commit -m="Added json gem" -a="Kate Smith" \
    0b2616b0e5a8 ouruser/sinatra:v2
  - $ sudo docker images

Create an image from a Dockerfile


```
FROM library/centos:centos6
MAINTAINER fabio fumarola fabiofumarola@gmail.com
RUN yum install -y curl which tar sudo openssh-server openssh-clients rsync
# passwordless ssh
RUN ssh-keygen -q -N "" -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -q -N "" -t rsa -f /etc/ssh/ssh_host_rsa_key
RUN ssh-keygen -q -N "" -t rsa -f /root/.ssh/id_rsa
RUN cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Build and run an image

- $docker build –t fabio/centos:ssh .
- $docker run –i –t fabio/centos:ssh /bin/bash
- Or
- $docker run –d fabio/centos:ssh /bin/bash
- Check the following commands:
  - $ docker top
  - $ docker logs
  - $ docker inspect

Other Commands

- Docker cp: copy a file from container to host
- Docker diff: print container changes
- Docker top: display running processes in a container
- Docker rm /rmi: delete container/image
- Docker wait: wait until container stop and print exit code

More on: http://docs.docker.io/en/latest/commandline/cli

Docker vs Vagrant?

- Less memory for Dockers w.r.t VMs
- With a VM you get more isolation, but is much heavier. Indeed you can run 1000 of Dockers in a machine but not thousand of VMs with Xen.
- A VM requires minutes to start a Docker seconds

There are pros and cons for each type.

- If you want full isolation with guaranteed resources a full VM is the way to go.
- If you want hundred of isolate processes into a reasonably sized host then Docker might be the best solution

http://blog.docker.com/2015/02/orchestrating-docker-with-machine-swarm-and-compose/

ORCHESTRATE DOCKER WITH MACHINE, SWARM AND COMPOSE

Motivation

- Docker Engine works well for packaging applications making much easier to
  - build,
  - deploy
  - and move between providers.
- But, to deploy complex application consisting of multiple services we need to resort to shell scripts.

Motivation

- This isn't ideal
- We'd like to have a more controllable method to distribute applications in the cloud.
- We need that our distributed application is:
  - **Portable across environments**: run seamlessly in testing, staging and production
  - **Portable across providers**: move the applications between different cloud providers
  - **Composable**: split up an application in multiple services

How to Orchestrate Dockers

There are three new tools that can be used to orchestrate docker containers:

- Machine,
- Swarm,
- Compose.

Docker Machine

- It lets easily deploy docker engines on your computer, on cloud providers and in a data center.
- It supports the following providers:
  - Amazon EC2
  - Microsoft Azure
  - Microsoft Hyper-V
  - DigitalOcean
  - Google Compute Engine
  - OpenStack
  - Rackspace
  - SoftLayer
  - VirtualBox
  - VMware Fusion
  - VMware vCloud Air
  - VMware vSphere

Docker Machine

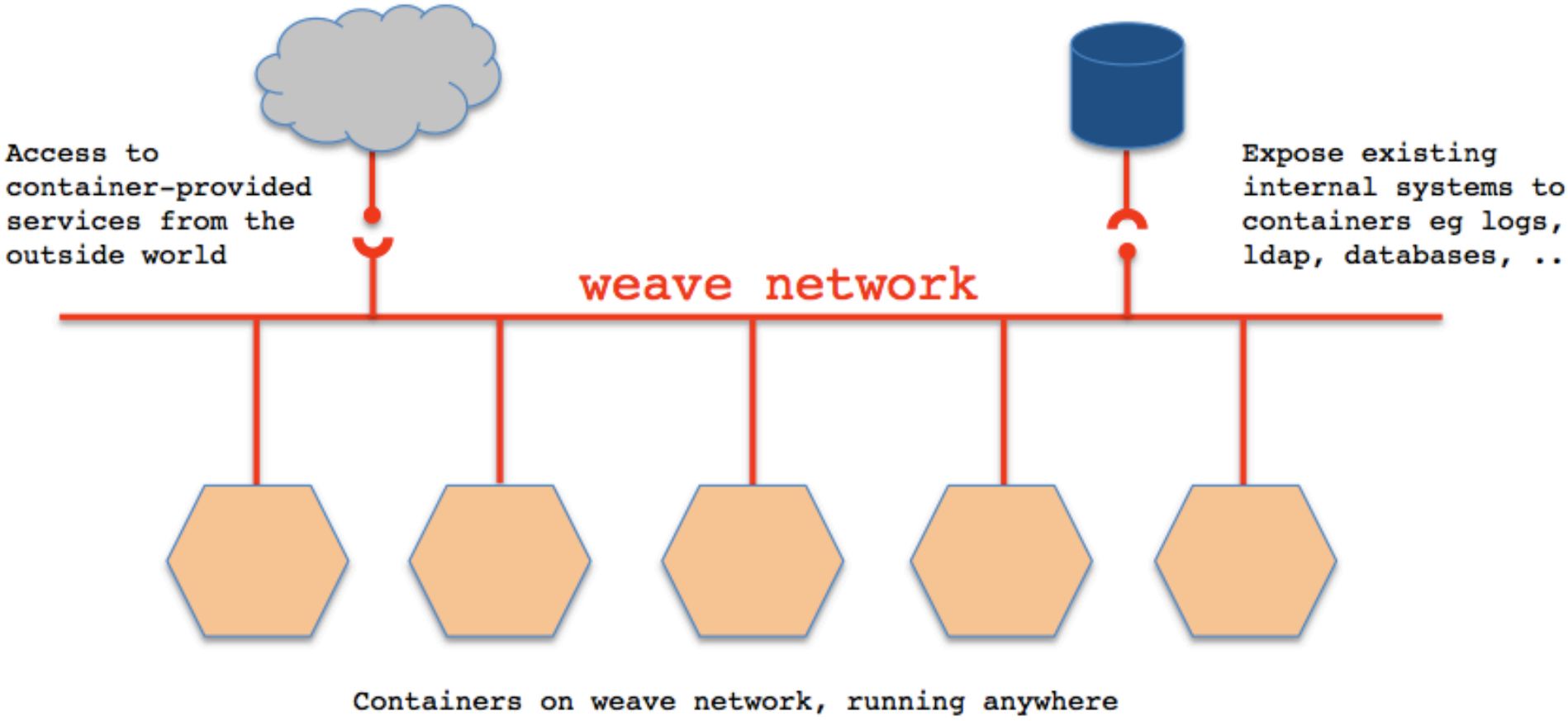
- It is supported on Windows, OSX, and Linux.
  - Windows - x86_64
  - OSX - x86_64
  - Linux - x86_64
  - Windows - i386
  - OSX - i386
  - Linux - i386
- At the lab we will explore how to use it.

Swarm and Weave

- Swarm allows us to connect together several docker containers deployed on different sub-networks.
- This happens when you need to deploy dockers in several machines and you want to achieve resilience through load balancing.
- It pools together several Docker Engines into a single virtual host.

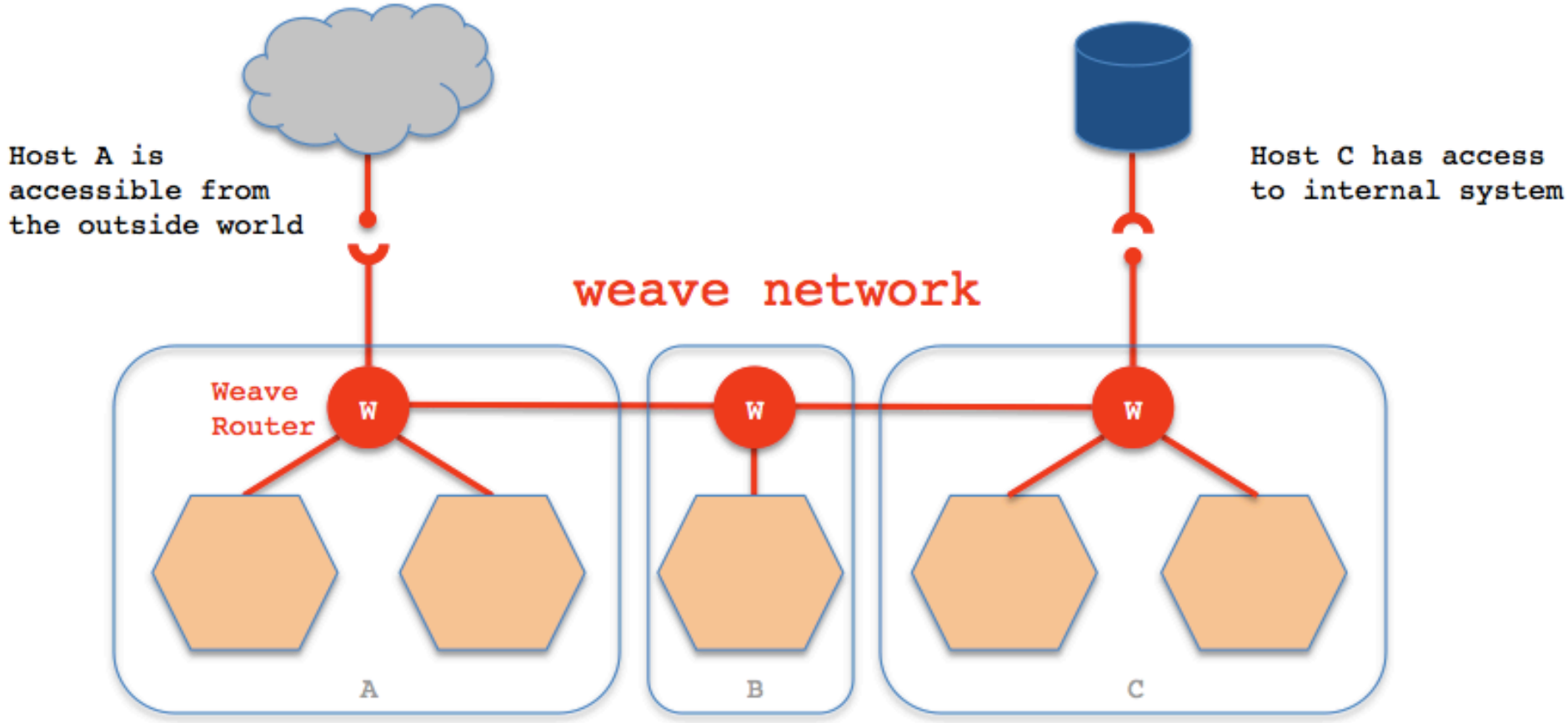
http://blog.docker.com/2015/02/scaling-docker-with-swarm/

Swarm and Weave



https://github.com/zettio/weave

Swarm and Weave



Hosts A, B & C running the containers shown in previous figure

Compose

- It is a way of defining and running multi-container distributed applications with Docker.
- When you need to setup an application that requires other services (e.g. redis, postgres,…) it is possible to use compose.
- Next, you define the components that make your app so they can be run together in an isolate environment.

Compose

- It is based on a dockerfile and on a yaml configuration file

Dockerfile

```
FROM python:2.7
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code
CMD python app.py
```

docker-compose.yml

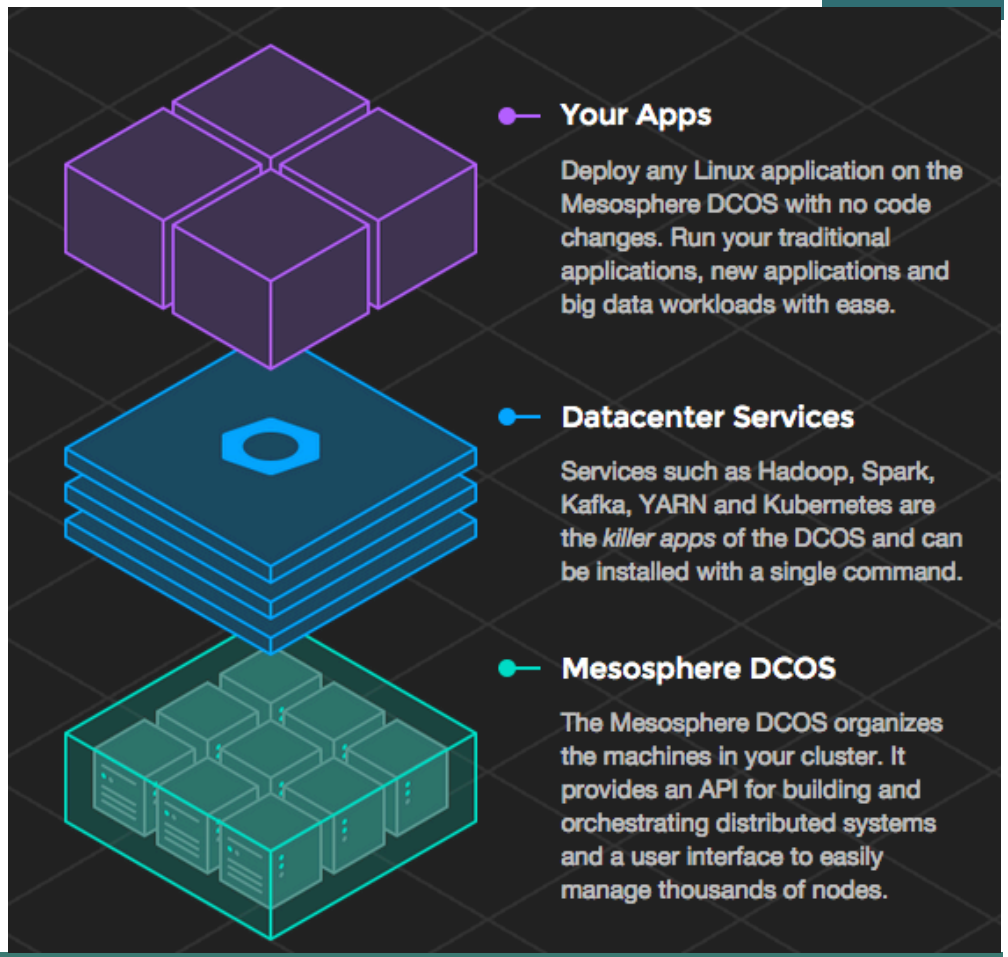
```
web:
  build: .
  links:
    - redis
  ports:
    - "5000:5000"
redis:
  image: redis
```

http://mesosphere.com/

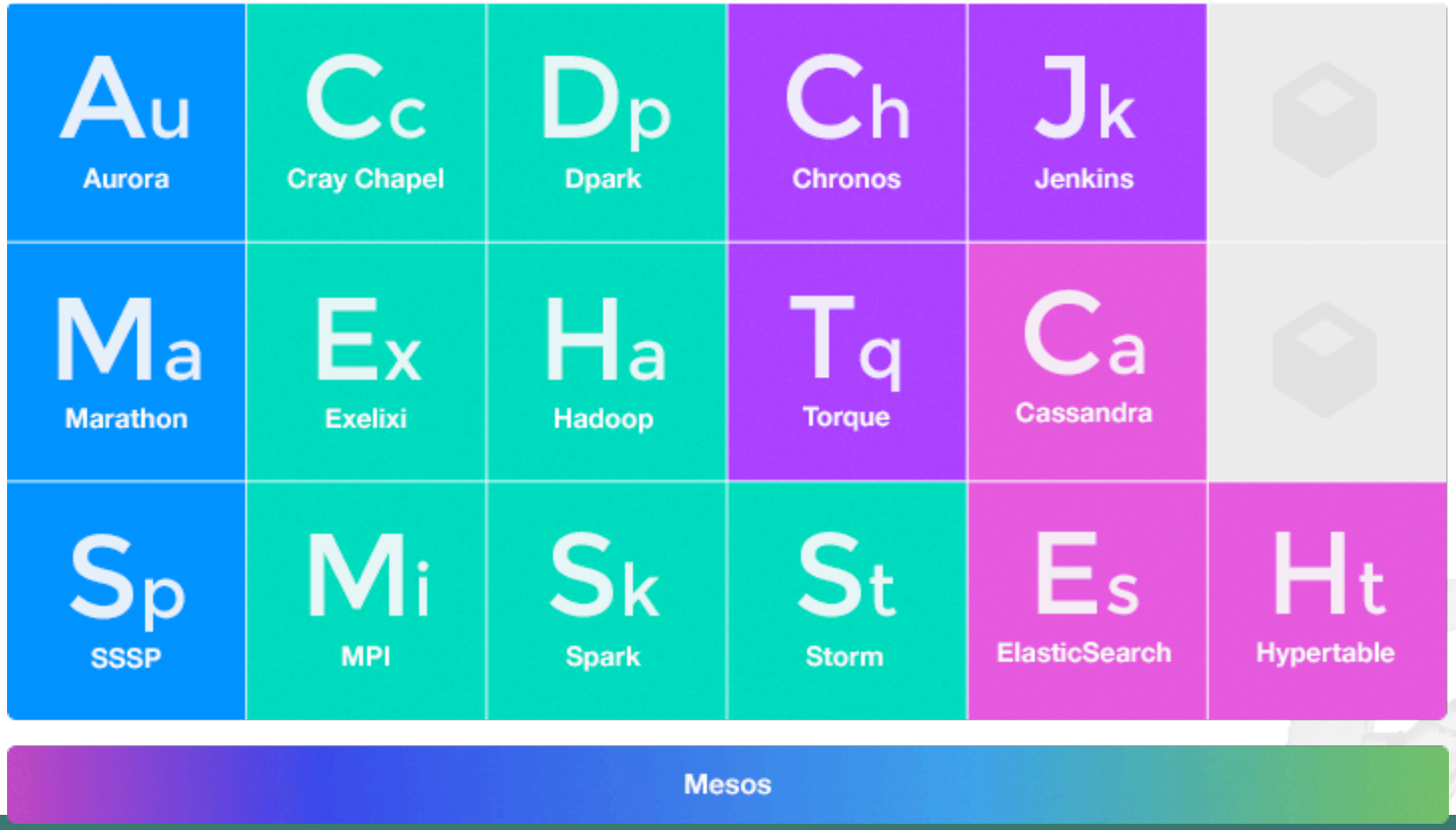
MESOSPHERE

Mesosphere

- It is an apache project that allows you to separate
  - the application you deploy
  - From the datacenter administration



Frameworks on Mesos



Additional Contributions

Play on Mesosphere

Running Play on Mesosphere enables a highly-available compute tier perfectly matched to the power and scalability of the Play web framework.

Ruby on Rails on Mesosphere

Rails infrastructure management can be simplified by running it on Mesos; turn your Rails app into an elastic and highly-available web service.

Kubernetes on Mesos

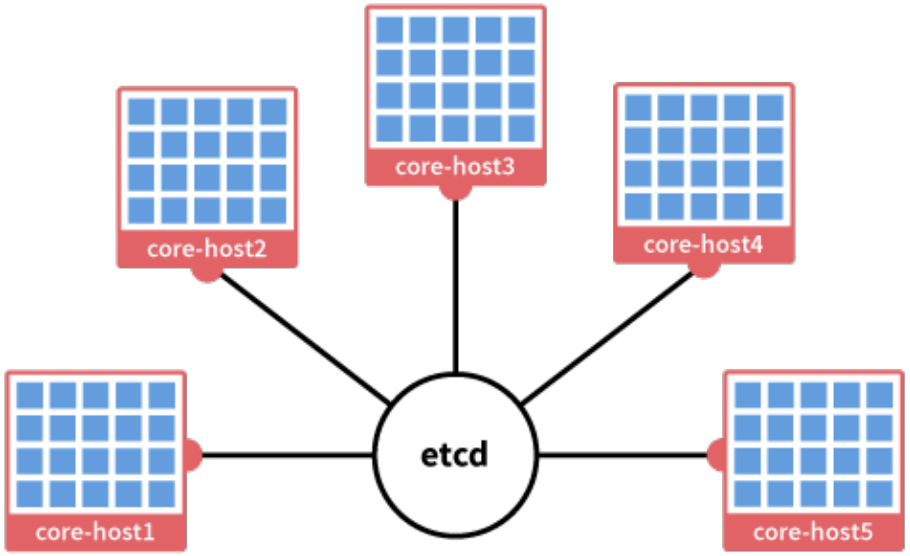
Tutorials on setting up the Kubernetes-Mesos framework.

CORE OS

CoreOS

- A minimal operating system
- Painless updating: utilizes active/passive scheme to update the OS as single unit instead of package by package.
- Docker container
- Clustered by default
- Distributed System tools: etcd key-value store
- Service discovery: easily locate where service are running in the cluster
- High availability and automatic fail-over

CoreOS



Clustered by default

High availability and a utomatic fail-over

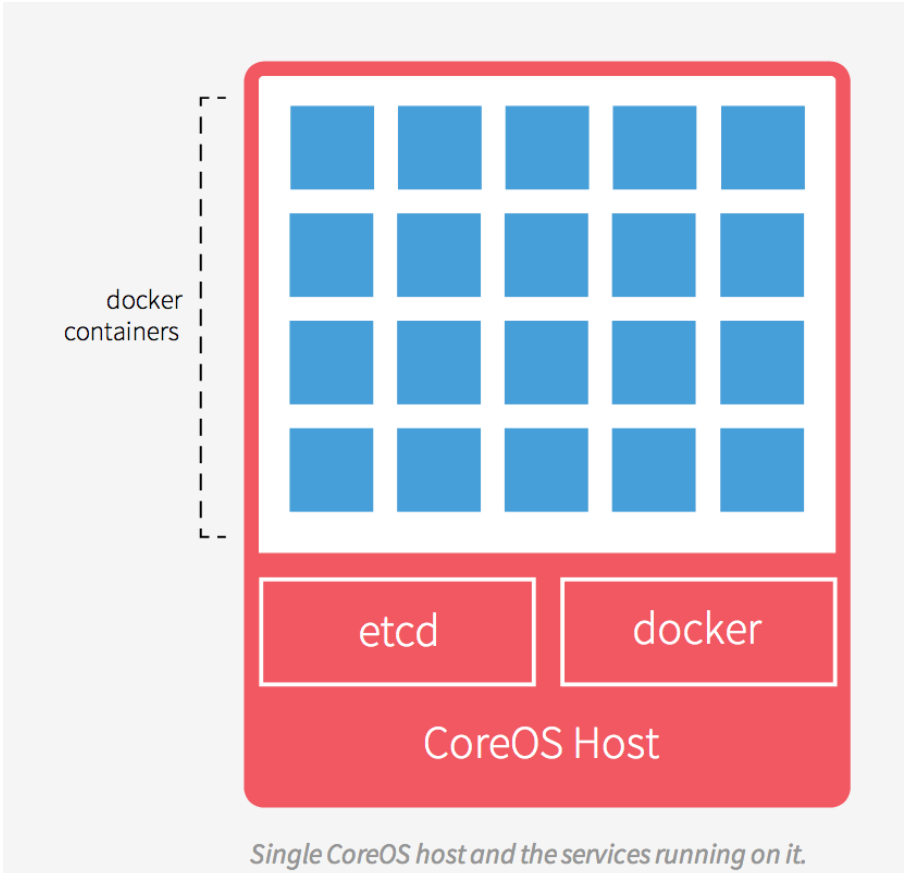
Docker with CoreOS

Features

- Automatically runs on each CoreOS machine
- Updated with regular automatic OS updates
- Integrates with etcd
- Networking automatically configured

Example Akka cluster + Docker + CoreOS

https://github.com/dennybritz/akka-cluster-deploy



Single CoreOS host and the services running on it.

References

- http://www.iheavy.com/2014/01/16/how-to-deploy-on-amazon-ec2-with-vagrant/
- https://docs.vagrantup.com/v2/
- Vagrant: Up and Running Paperback – June 15, 2013
- https://github.com/patrickdlee/vagrant-examples
- https://linuxcontainers.org/ LXC
- https://www.kernel.org/doc/Documentation/cgroups/
- http://lamejournal.com/2014/09/19/vagrant-vs-docker-osx-tales-front/
- https://medium.com/@_marcos_otero/docker-vs-vagrant-582135beb623
- https://coreos.com/using-coreos/docker/