Computación en la nube con virtualización liviana (Kubernetes)

Edgar Magaña, PhD

Temas del Curso

Temario

* Introduction to Containers

* Setting up and getting started
** Installing docker
** Installing Minikube
** Explore minikube and operations

* Kubernetes and the Cloud-native ecosystem
** Deep dive in kubernetes

* Application Deployment
** Reading and creating YAML
** Creating a namespace
** Deploy an application
** Verify health of application
** Review application logs
** K9s

* Kubernetes Architecture
** K8s Control Plane
** K8s Data Plane
** Communication between Control and Data Planes
** Exercises

* Complex Application Deployment
** Expose app to internet via LB
** Add resource requests and limits
** Operations on K8s resources

Introduction to Containers

Edgar Magaña, PhD

Outline

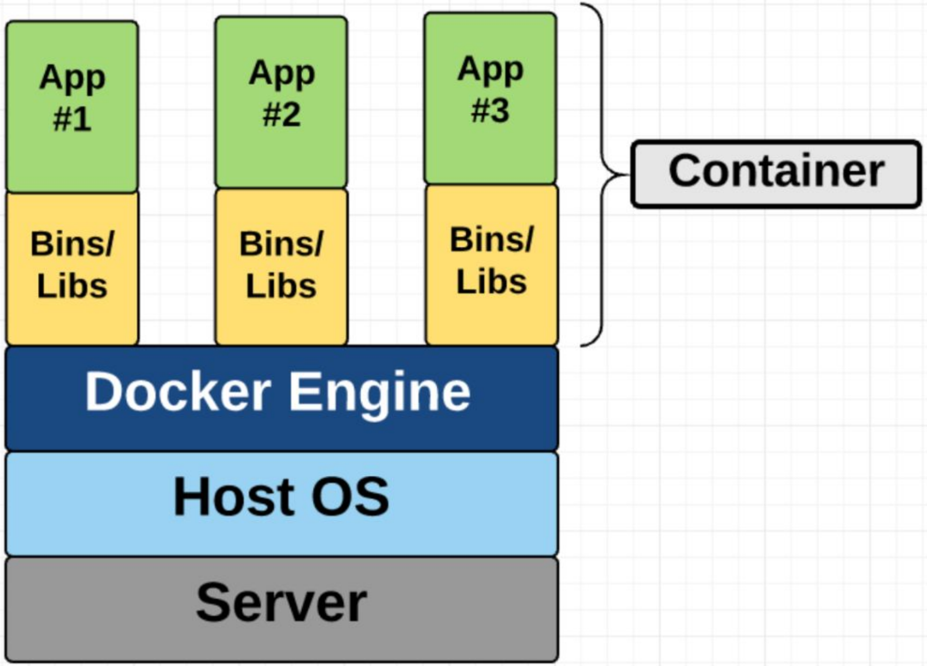
- **Container recipe**
- Why should I care?
- A quick docker demo
- Building blocks
- Security

Containers? These?

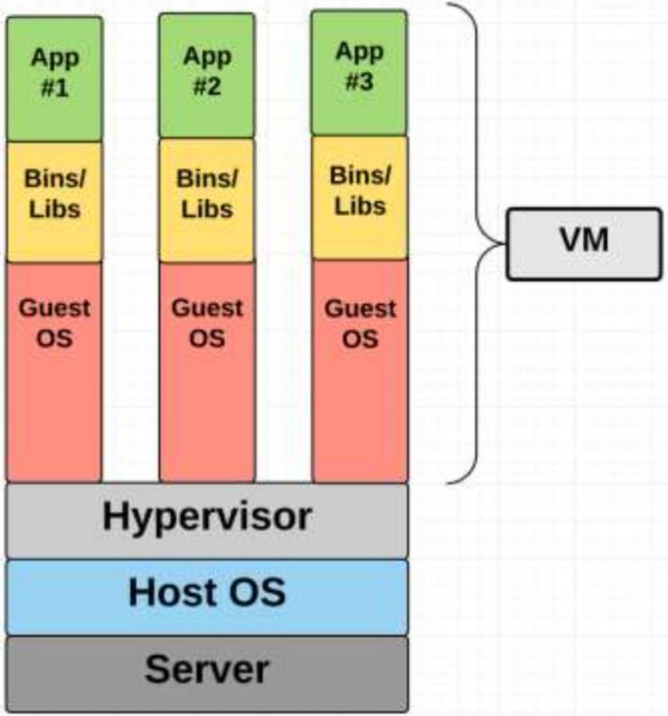
Before containers ...things were messy

source: https://en.wikipedia.org/

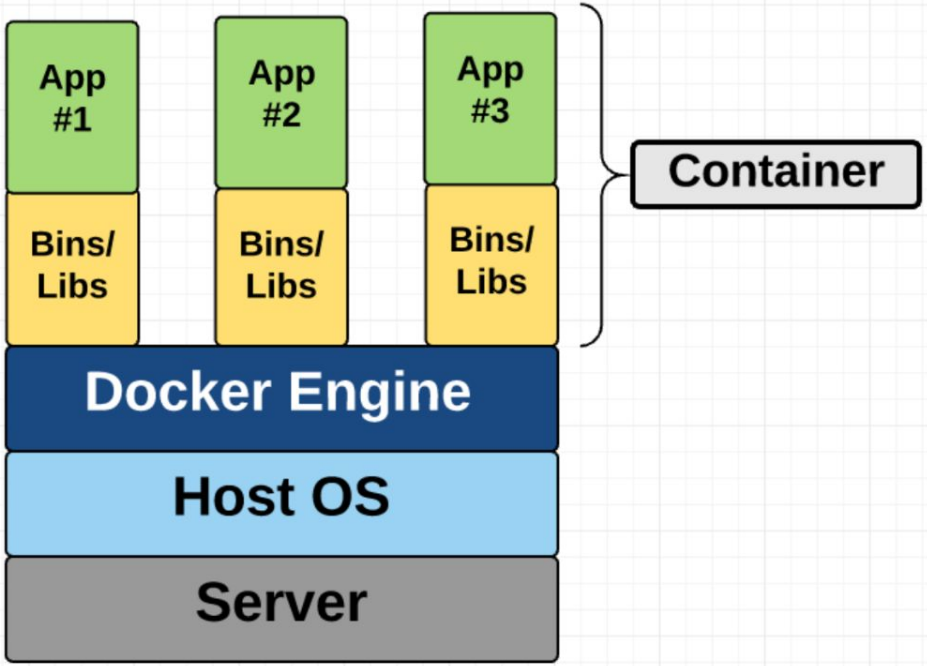
Containers



source: https://medium.com



Containers



source: https://medium.com

- All containers share the same kernel of the host system.
  Pro: Extremely reduced performance overhead.

- Better utilization of resources due to shared kernel.

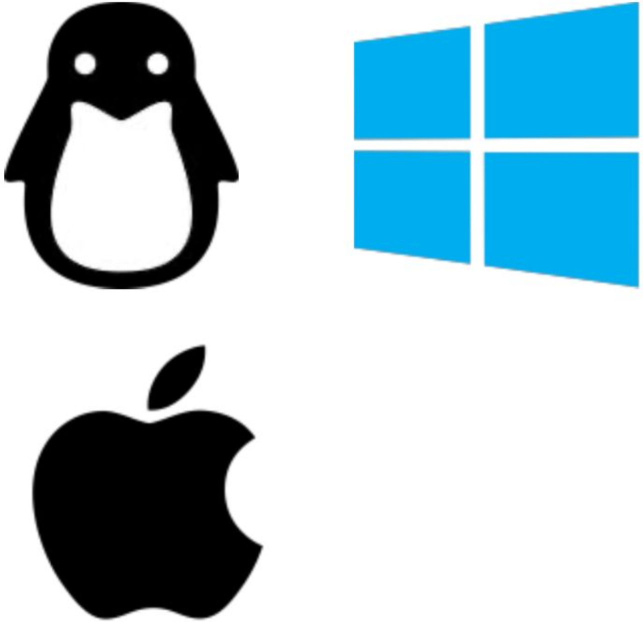
- Lightweight and uses less space on disk.

- Portable and better dependency management

Outline

- Container recipe
- **Why should I care?**
- A quick docker demo
- Building blocks
- Security

Deploy anywhere and anything



- webapps
- backends
- SQL, NoSQL
- big data
- load balancing
- … and more

... but, it was working on my machine.

Deploy reliably & consistently

- If it works locally, it will work on the server
- With exactly the same behavior
- Regardless of versions
- Regardless of distros
- Regardless of dependencies
- Typical laptop runs 10-100 containers easily
- Typical server can run 100-1000 containers

Outline

- Container recipe
- Why should I care?
- **A quick docker demo**
- Building blocks
- Security

Demo

- Docker Installation
- CLI
    - Basic Commands
    - Pull Images
    - Deploy Containers
- Docker Hub

Is docker running?

```
docker run hello-world
```

docker run

```
docker run -ti debian bash
```

-ti -> terminal interactive

Check list of images

```
docker images
```

Pulling images

```
docker pull ubuntu:xenial
```

Terminology

Images - The blueprints of our application which form the basis of containers. In the demo above, we used the `docker pull` command to download the busybox or ubuntu image.

Containers - Created from Docker images and run the actual application. We create a container using `docker run` which we did using the busybox image that we downloaded. A list of running containers can be seen using the `docker ps` command.

Docker Daemon - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system to which clients talk to.

Docker Client - The command line tool that allows the user to interact with the daemon. More generally, there can be other forms of clients too - such as Kitematic which provide a GUI to the users.

Docker Hub - A registry of Docker images. You can think of the registry as a directory of all available Docker images. If required, one can host their own Docker registries and can use them for pulling images.

Run docker container from images

```
docker run -ti ubuntu:latest bash
```

ubuntu -> image
latest -> tag (optional, by default it's latest)
bash -> what do we want to do with the image.

Leave container running in background(detatch)

```
docker run -d -ti ubuntu /bin/bash
```

-d -> detaches the container

Additional commands

- `docker ps -a`
- `docker info`
- `docker restart zeolous_darwin`
- `docker inspect blisful_saha`
- `docker inspect blisful_saha | grep -i ip`

Dockerfile

- A Dockerfile is a simple text-file that contains a list of commands that the Docker client calls while creating an image.
- It's a simple way to automate the image creation process.

Let's create a Dockerfile:

```
mkdir build

cd build

vim Dockerfile
```

Dockerfile

```
#This is a custom ubuntu image with vim already installed

FROM ubuntu:xenial

MAINTAINER emagana <emagana@gmail.com>

RUN apt-get update

RUN apt-get install -y vim
```

Dockerfile

```
#Build the new docker image

docker build -t=”emagana/ubuntuvim:v3” .
```

-t -> title

. -> dot, because Dockerfile is in the same folder.

Apache Web Server

#Building our own web server

```
docker pull httpd
```

```
docker run -d --name apache -p 80:80 httpd
```

```
http://localhost:80
```

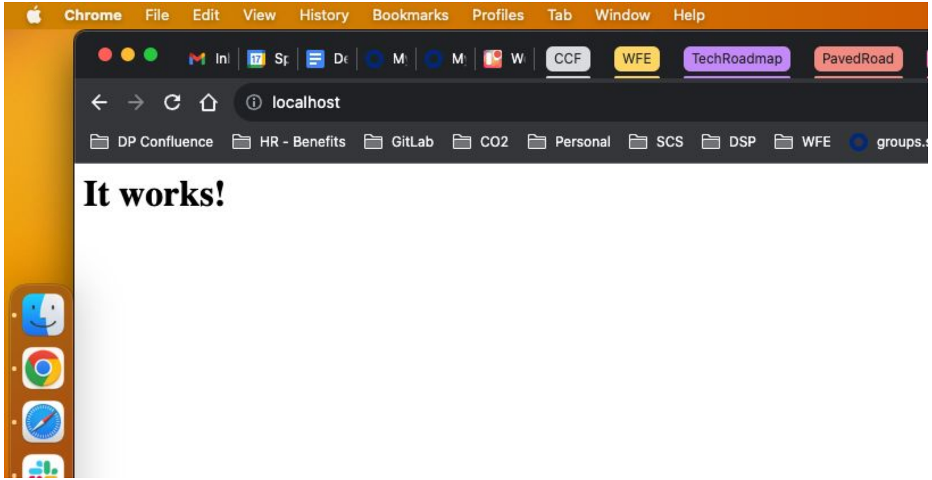
```
docker stop [container-name-or-id]
```

```
mkdir apache && cd apache
```

```
vim index.html
```

```
<h1>Test</h1>
```

```
<p>This is a test page for the Apache deployment in Docker</p>
```

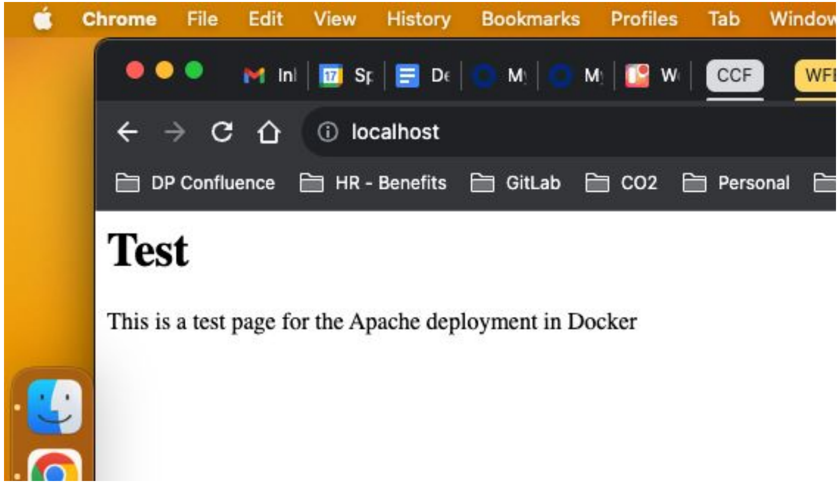


It works!

Apache Web Server

#Let’s build/customize our web server

```
vim Dockerfile

FROM httpd:latest

COPY index.html /usr/local/apache2/htdocs

EXPOSE 80

docker build -t [image-name] .

docker run -d --name apachev1 -p 80:80 apache:v1

http://localhost:80
```


Chrome File Edit View History Bookmarks Profiles Tab Window
In Sp De M M W CCF WFI
localhost
DP Confluence HR - Benefits GitLab CO2 Personal

Test

This is a test page for the Apache deployment in Docker

Outline

- Container recipe
- Why should I care?
- A quick docker demo
- **Building blocks**
- Security

Building blocks of containers

- **Again.. what is a container?**
- Cgroups
- Namespaces

What is a container?

- How it "feels" like:
  - own process space
  - own network space
  - run stuff as root
  - can install packages
  - can run services
  - can mess up routing, iptables ...

What is a container?

- It's not quite like a VM:
  - uses the host kernel
  - can't boot a different OS
  - can't have its own modules
  - doesn't need init as PID 1
  - doesn't need syslogd, cron...
- It's just normal processes on the host machine
  - contrast with VMs which are opaque

Building blocks of containers

- Again.. what is a container?
- **Cgroups**
- Namespaces

Control Groups (Cgroups)

- Resource metering and limiting
  - memory
  - CPU
  - block I/O
  - network*
- Device node (/dev/*) access control
- Crowd control

Memory cgroup: limits

- Each group can have its own limits
  - limits are optional
  - two kinds of limits:
    - Soft limits
    - Hard limits
- Soft limits are not enforced
  - they influence reclaim memory pressure

Memory cgroup: limits

- Hard limits will trigger a per-group OOM killer
- Limits can be set for different kinds of memory
  - physical memory
  - kernel memory
  - total memory
- Multiple groups use the same page, only first one is "charged"
  - but if it stops using it, the charge is moved to another group.

CPU cgroup

- Keep track of user/system CPU time
- Keeps track of usage per CPU
- Allows to set weights

CPUset cgroup

- Pin groups to specific CPU(s)
- Reserve CPUs for specific apps
- Avoid processes bouncing between CPUs

Building blocks of containers

- Again.. what is a container?
- Cgroups
- **Namespaces**

Namespaces

- Provide processes with their own system view
    - Cgroups = limits how much you can use;
    - Namespaces = limits what you can see
- Multiple namespaces:
    - pid
    - net
    - mnt
    - uts
    - ipc
    - user
- Each process is in one namespace of each type

pid namespace

- Processes within a PID namespace only see processes in the same PID namespace
- Each PID namespace has its own numbering
  - starting at 1
- If PID 1 goes away, whole namespace is killed
- Those namespaces can be nested
- A process ends up having multiple PIDs
  - one per namespace in which its nested

Network namespaces: in theory

- Processes within a given network namespace get their own private network stack, including:
  - network interfaces
  - routing tables
  - iptable rules
  - sockets

Network namespaces: in practice

- Use virtual interfaces acting as a cross-over cable
- eth0 in container network namespace, paired with vethXX in host network namespace.
- All the vethXX are bridged together via virtual switch inside the container host.
    - Docker calls the bridge docker0

Outline

- Container recipe
- Why should I care?
- A quick docker demo
- Building blocks
- **Security**

Use RunC Flaw to gain Root access on Host

- The vulnerability, identified as **CVE-2019-5736**, was discovered by two open source security researchers on 11th Feb 2019.
- “High level” container runtimes like Docker does image creation and management
- Use "Low level" runC to handle tasks related to running containers
  - creating a container
  - attaching a process to an existing container (docker exec)

The Vulnerability

Overview given by the runC team:

The vulnerability allows a malicious container to (with minimal user interaction) overwrite the host runc binary and thus gain root-level code execution on the host. The level of user interaction is being able to run any command ... as root within a container in either of these contexts:

- Creating a new container using an attacker-controlled image.
- Attaching (docker exec) into an existing container which the attacker had previous write access to.

Bibliography

What is container? | https://www.docker.com/resources/what-container

Demystifying containers 101 | https://tinyurl.com/yxd7fnpe

Getting started with Docker | https://tinyurl.com/y35e879j , http://bit.do/eJwmy

Docker for development and deployment | http://bit.do/eJwmH

RunC Flaw Lets Attackers Escape Linux Containers to Gain Root on Hosts | https://is.gd/3qPVsQ

CVE-2019-5736 | https://nvd.nist.gov/vuln/detail/CVE-2019-5736

Explaining CVE-2019-5736 | http://bit.do/eJLcG

LXC | https://linuxcontainers.org/lxc/introduction/ , https://is.gd/i0F9i4

Container basics | https://is.gd/Eq4gaP https://is.gd/pGgMqW

Bibliography

Namespaces in operation | https://lwn.net/Articles/531114/

Future of Linux Containers | https://is.gd/U0QR9K

cgroups | https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt , https://is.gd/1x1ihc

Attacks on Linux Containers | https://ieeexplore.ieee.org/document/7854163

Comparing different containers | https://ieeexplore.ieee.org/document/8075934

Beginners Guide to Containers Technology and How it Actually Works[video] | https://is.gd/CzTb1d

My Docker Guide | https://github.com/w4rb0y/devOpsNotes/blob/master/dockerGuide.md

Google Cloud | https://cloud.google.com/containers/

Containers from scratch | https://ericchiang.github.io/post/containers-from-scratch/

Thank you!