Docker

Engine

Images

Container

lib container

lsc

storage drivers

docker ecosystem

containers

Kernel namespace

Control Groups

Capabilities

Before containers and VM's

One server per application

Wasted resources

Huge redundant infrastructure for each server

Virtualization requirements exist to best serve the app while still saving money

Provides an application runtime environment

Filesystem

Devices

priviliges

API's

functions

Main requirements of VM?

Secure environemnt

Minimal OS services

QOS controls

Why virtualization?

Isolate separate applications so they don't collide

Dependency management

Resource allocation

Save resources

Virtual machines

Hypervisors

Container virtualization as opposed to OX virtualization

Physical machine carved up into separate virtual machine

Each VM appears like a separate machine

Supported more efficient use of server resources

Goods

Each VM is a security boundary

Problems

Much better than 1 server per app, but still bad

VM sprawl

Each VM requires a full blown OS

VM's remove need for separate server per app, but requires one OS per app

Each OS takes us RAM, CPU and diskspace

Some OS's need separate license per OS install

Huge overhead

Each OS is a huge resource hog

Too much overhead for what is required for each app

Containers are way more lightweight and efficient

KVM / VMWare

Container

Lightweight virtualization

Less CPU / RAM / diskspace than VM's

Allow creation of isolated separate user spaces in a single OS.

OS-level virtualization

Each container is a very lightweight OS

Each shares a common OS

Faster and more portable than VM's

Isolated instances of user space

Realized via linux kernel namespaces

At least in labs and even in some cloud hosting, containers are run inside VM's (Virtualbox, VMWare)

This isn't considered running on bare metal

Microservices architecture

Applications are designed as a suite of services that collective operate to form the app

Each service runs as a separate process in it's own container

Each service can be changed in isolation

Docker company

Used to be called dotCloud, a PAAS provider

Solomon Hykes

Written in Google GO language

Docker Platform

What started as just a container is quickly evolving into much more

docker image format

docker container runtime

docker registry

docker clustering

docker networking

..

Docker Containers General

Container runtime

Implementation of a container framework provided by linux

Containers can be dropped into Docker engine running anywhere, for example a laptop, AWS, Azure, On-premisis data center

Like write once run anywhere

Linux Features for Containers

Kernel Namespaces

Partitions system namespaces into separate user spaces

Each user space partition doesn't know about other partitions

Partitionable namespaces

pid

net

mnt

usr

User spaces can have root priviliges inside of a container, but not outside

CGroups (Control groups)

For QOS rules

Let you group together resources and apply limits

Containers map to cgroups

memory, cpu, block-io

Very flexible

More flexible than hypervisor controls

Ensure one container can't DOS others

Capabilities

Fine grain control over what user or process has

Break down root priviliges so they can be applied piecemeal to a process / usr

All capabilities are denied by default and overridden by a whitelist

User space

Root file systems

Process trees

Each has it's own process 0 and tree

Processes in different containers cannot send signals to each other (isolation)

Networking stacks

Own ip's and port ranges

Own routing table

User space partition

Docker uses a storage driver to manage files created by different containers

For example aufs is used in ubuntu

Docker stores files in /var/lib/docker/aufs

Docker architecture

Docker runtime

Also called the docker daemon, or engine

Does all the hardwork of creating the container

Docker client

Sends commands to docker daemon

Docker runtime

Client

Daemon

LXC

Execution driver

Used to be used by Docker as an interface to Linux kernel features

Not a good model since LXC isn't controlled by Docker

Replaced by libcontainer

Libcontainer

Drop in replacement for LXC

Enables cross-platform support (windows and linux-kernels)

Even so, you can only run docker containers in linux docker engines, and

Docker engine

The

Bare-metal

Hardware

Linux kernel manages the hardware underneath it

Ideally, docker runs on bare-metal, meaning not in a VM

User space

Area in an OS where applications are installed

aufs

storage driver used on ubuntu OS

default union

Installation

sudo su

service docker.io status

requires 3.8 kernel at least

uname -a

Update package indexes from source

apt-get update

SOP's

Give Docker control to non-root users

Allow normal users the ability to make and break containers

/var/run/docker.sock

srw-rw---- 1 root docker 0 Feb 21 13:37 docker.sock

s is for socket

Add users to docker group

Allows docker control w/o prefixing with sudo

cat /etc/group

sudo gpasswd -a mike docker

bounce the VM and you can now start a container

Tell docker to listen on a network port instead of a unix socket

Updating Docker

Add docker key to our keystore

wget -qO- https://get.docker.com/gpg | apt-key add -

Add docker repo to our apt sources

echo deb http://get.docker.com/ubuntu docker main > /etc/apt/sources.list.d/docker.list

apt-get install lxc-docker

Configuration

Image layering

Registries

Docker Hub

Internal registry

Storage volumes

Docker networking

Cassandra images

-seeds

List of nodes that a given cassandra node can talk to

docker run --name n2 -d tobert/cassandra -seeds 172.17.0.1