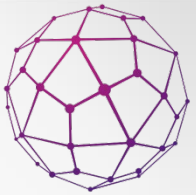


APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

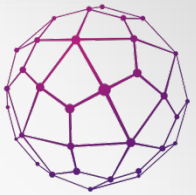
Computer Network Modelling #2

PhD. Antonenko V.A.

APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

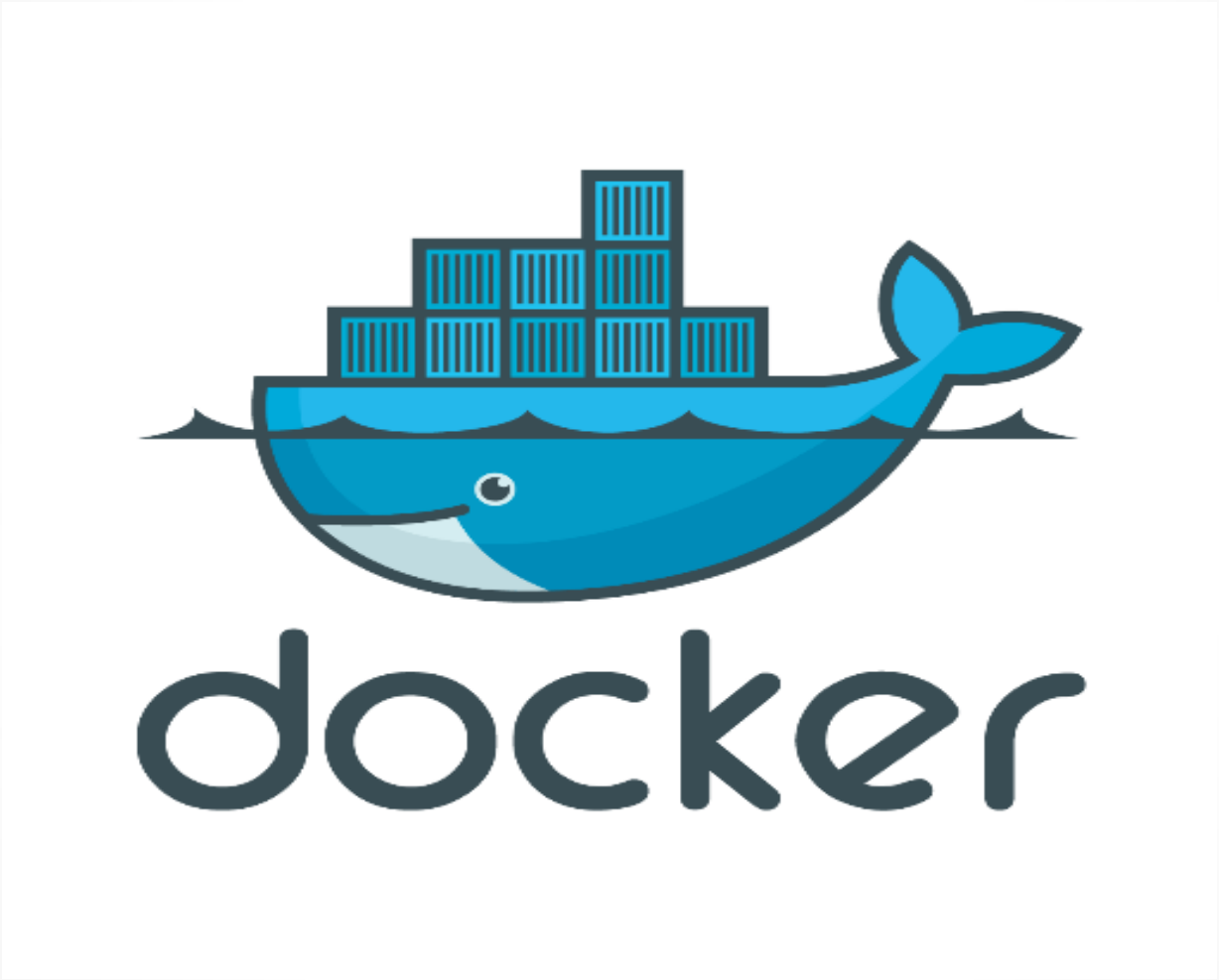
Goals Of This Lecture

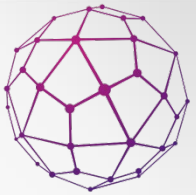
- Introduce Docker
- Introduce Mininet + Docker
- Introduce Mininet + Docker + Swarm



APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

Introduction to Docker





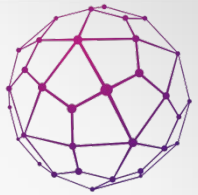
APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

DOCKER HISTORY .....

- A dotCloud (PAAS provider) project
- Initial commit January 18, 2013
- Docker 0.1.0 released March 25, 2013
- dotCloud pivots to docker inc. October 29, 2013

What is Docker ?

- Open platform for developers and sysadmins to build, ship and run distributed applications
- Can run on popular 64-bit Linux distributions with kernel 3.8 or later
- Supported by several cloud platforms including Amazon EC2, Google Compute Engine, and Rackspace.



APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

Features....

- Light-Weight
  - Minimal overhead (*cpu/io/network*)
  - Based on Linux containers
  - Uses layered filesystem to save space (AUFS/LVM)
  - Uses a copy-on-write filesystem to track changes
- Portable
  - Can run on any Linux system that supports LXC (today).
  - 0.7 release includes support for RedHat/Fedora family.
  - Raspberry pi support.
  - Future plans to support other container tools (lmctfy, etc.)
  - Possible future support for other operating systems (Solaris, OSX, Windows?)
- Self-sufficient
  - A Docker container contains everything it needs to run
  - Minimal Base OS
  - Libraries and frameworks
  - Application code
  - A docker container should be able to run anywhere that Docker can run.

The Challenge.....

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery

Production Servers

Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly?

The Matrix From Hell.....

APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Cargo Transport Pre-1960……

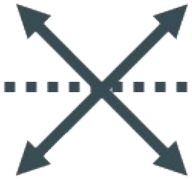
Multiplicity of Goods







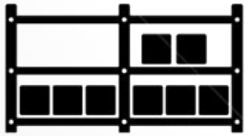
Do I worry about how goods interact (e.g. coffee beans next to spices)



Multipilicity of methods for transporting/storing



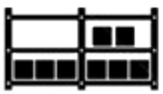



Can I transport quickly and smoothly (e.g. from boat to train to truck)

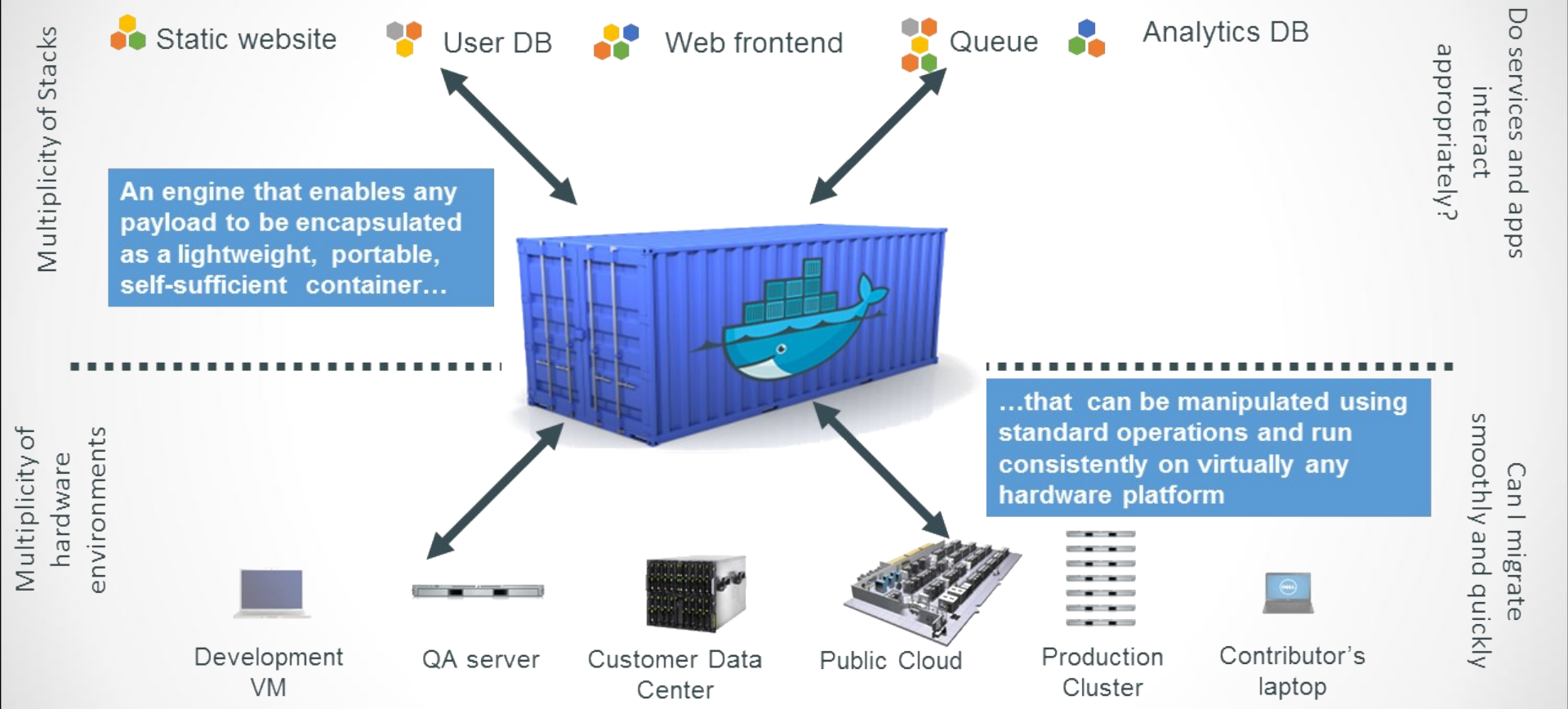
Also a Matrix from Hell.....

APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

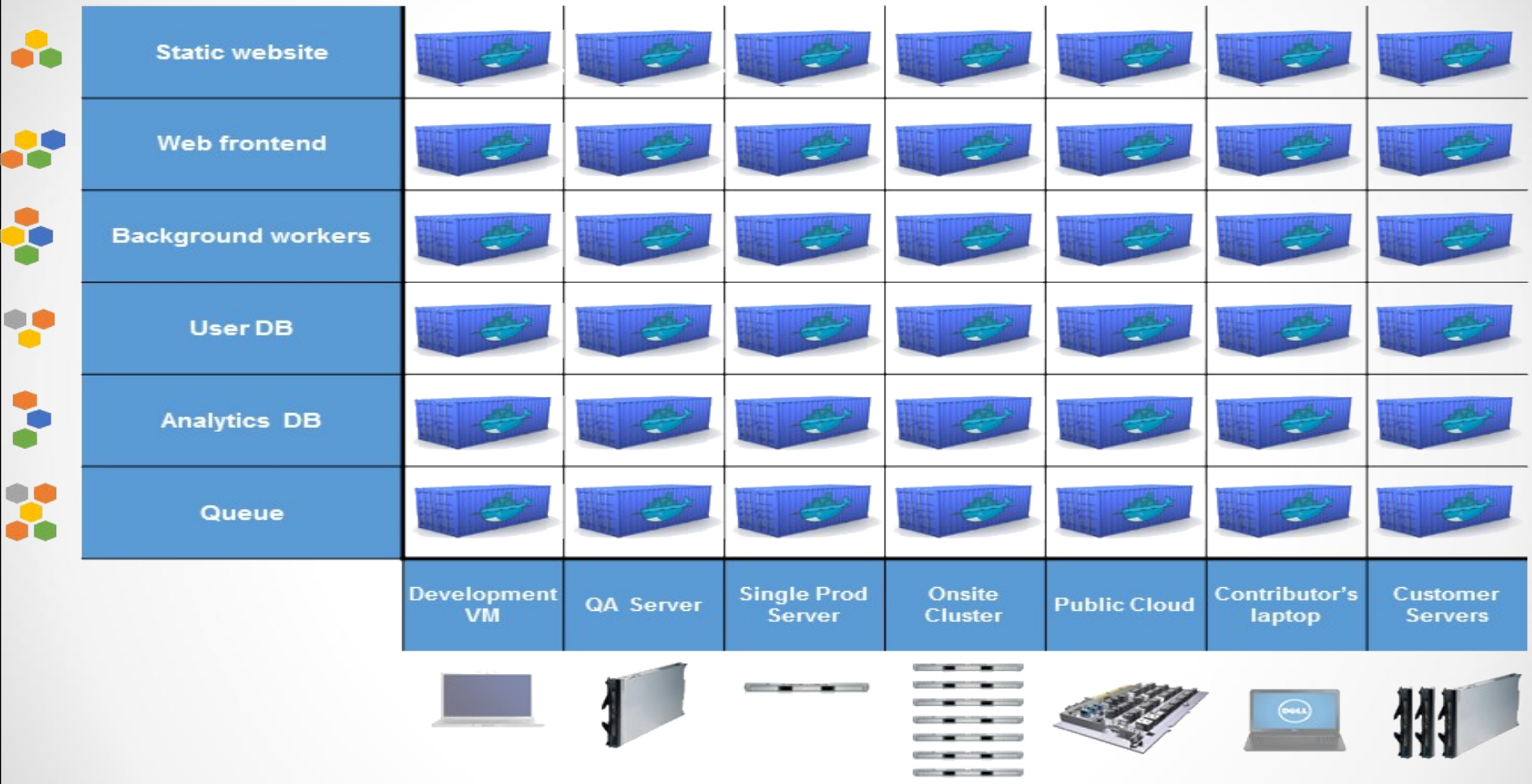
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container……

Docker is a Container System for Code...

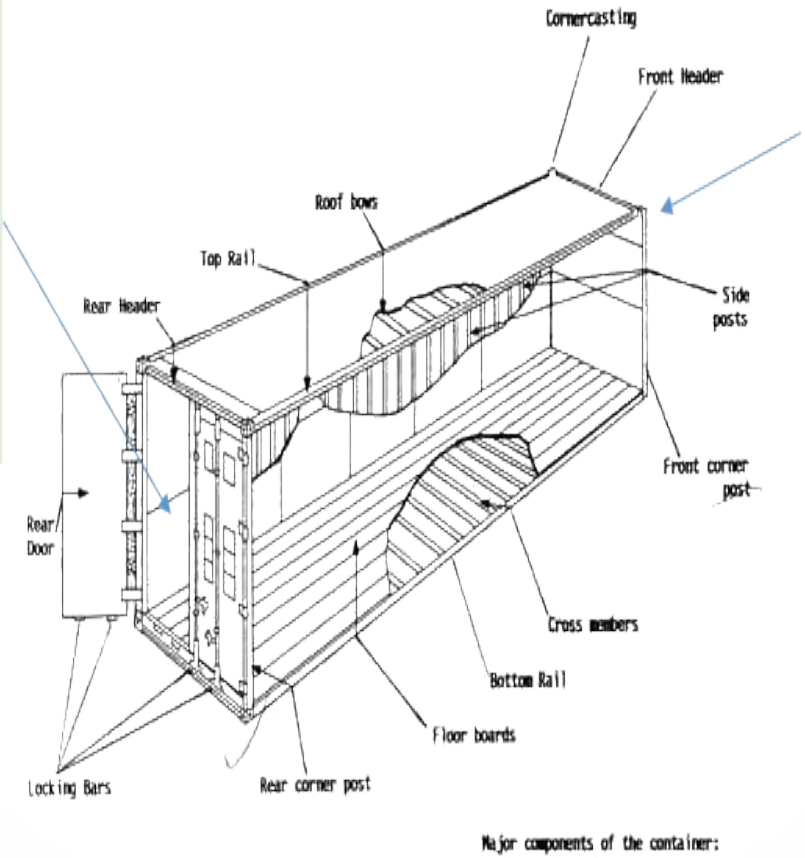


Docker Eliminates the Matrix from Hell...



Why it Works: Separation of Concerns....

- Dan the Developer
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same

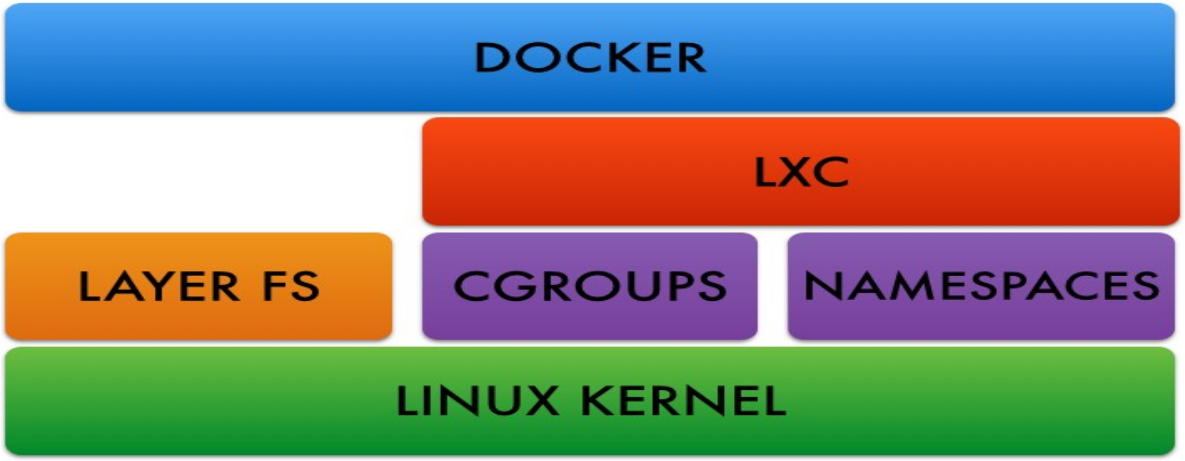


Major components of the container:

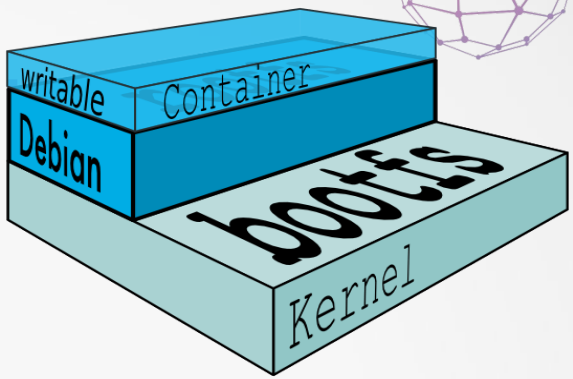
- Oscar the Ops Guy
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

Docker Architecture……

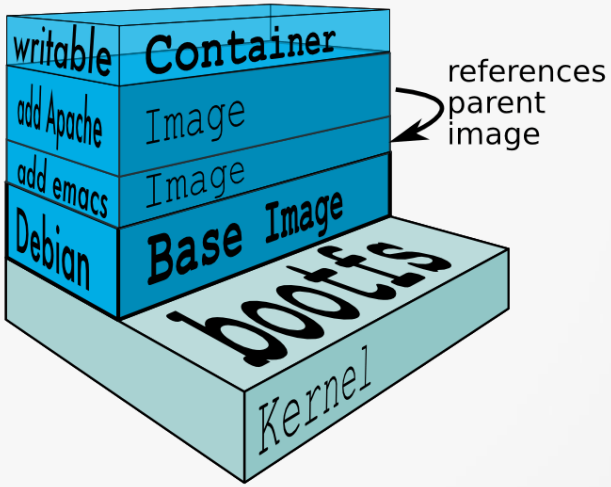
- Docker Engine
  - CLI
  - Docker Daemon
  - Docker Registry
- Docker Hub
  - Cloud service
    - Share Applications
    - Automate workflows
    - Assemble apps from components
- Docker images
- Docker containers



Docker images....



- NOT A VHD
- NOT A FILESYSTEM
- uses a *Union File System*
- a read-only *Layer*
- do not have state
- Basically a tar file
- Has a hierarchy
    - Arbitrary depth
- Fits into the Docker Registry



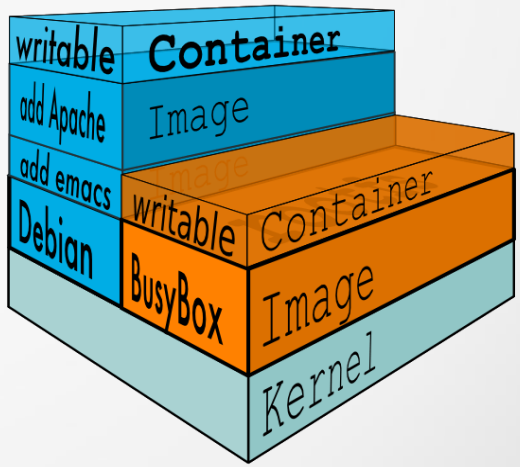
Docker Containers...

Units of software delivery (ship it!)

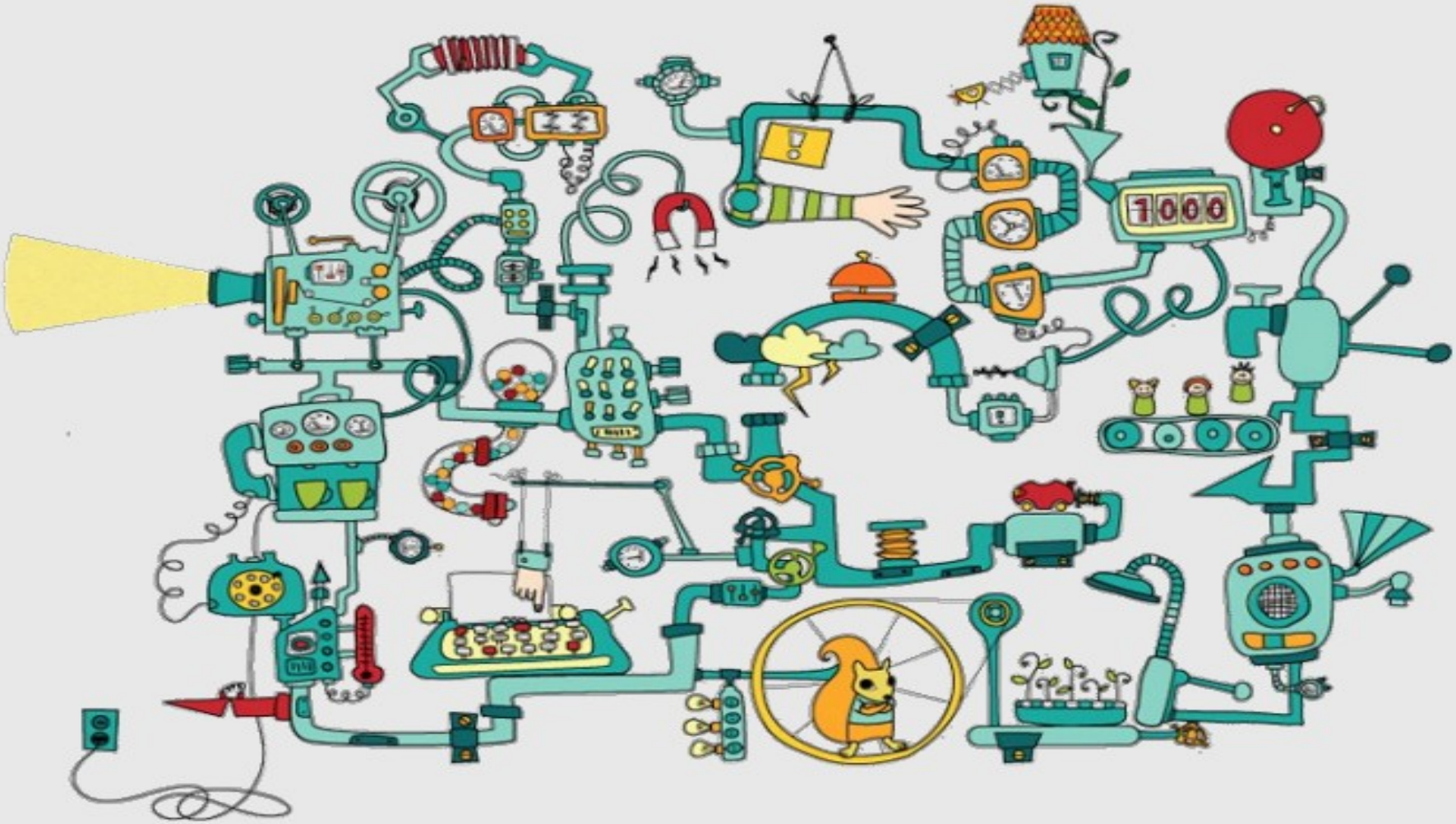
- run everywhere
  - regardless of kernel version
  - regardless of host distro
  - (but container and host architecture must match*)

- run anything
  - if it can run on the host, it can run in the container
  - i.e., if it can run on a Linux kernel, it can run

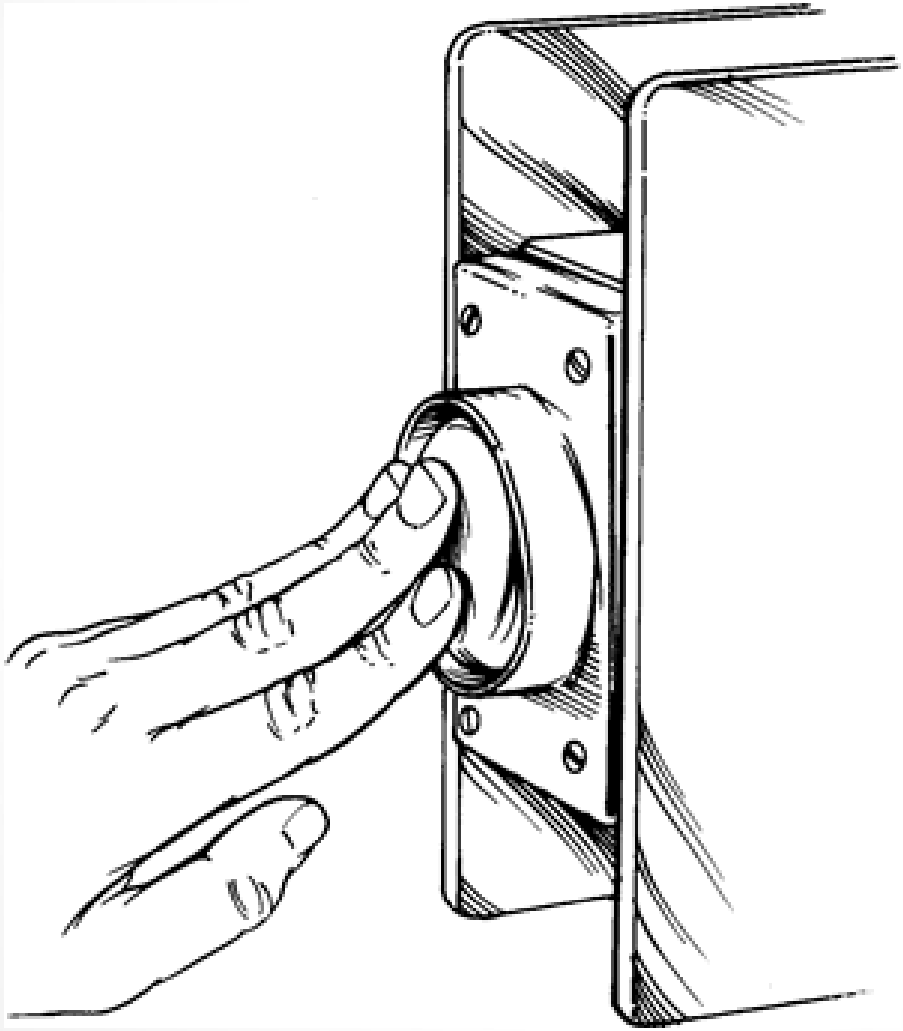
*Unless you emulate CPU with qemu and binfmt



ntainers before Docker……



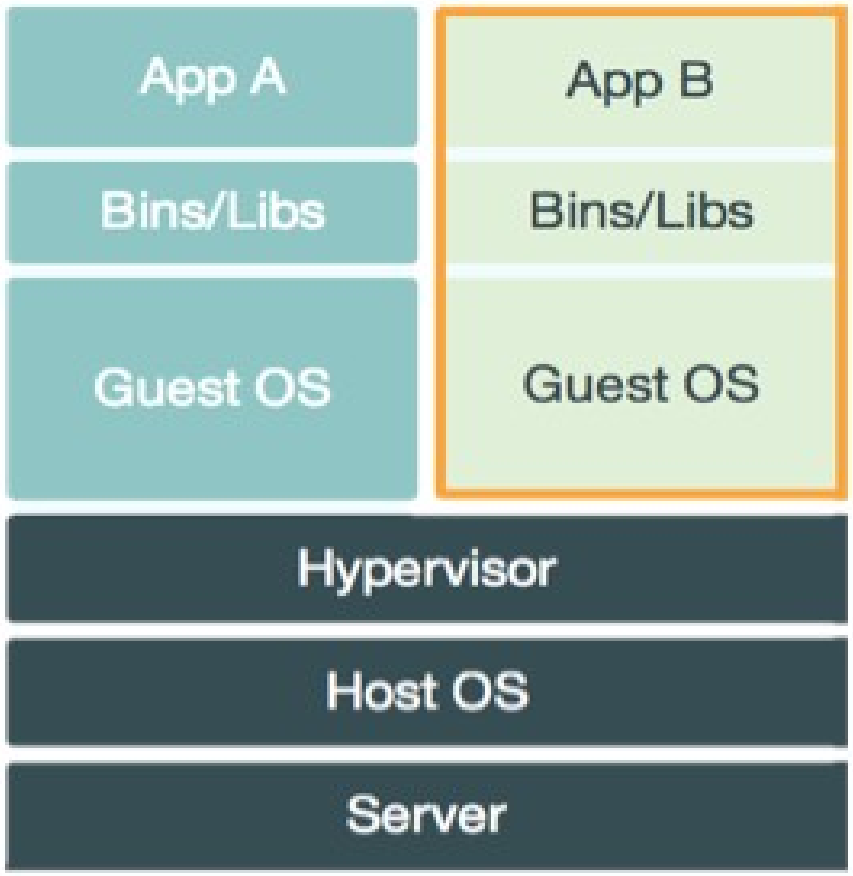
Containers after Docker .....

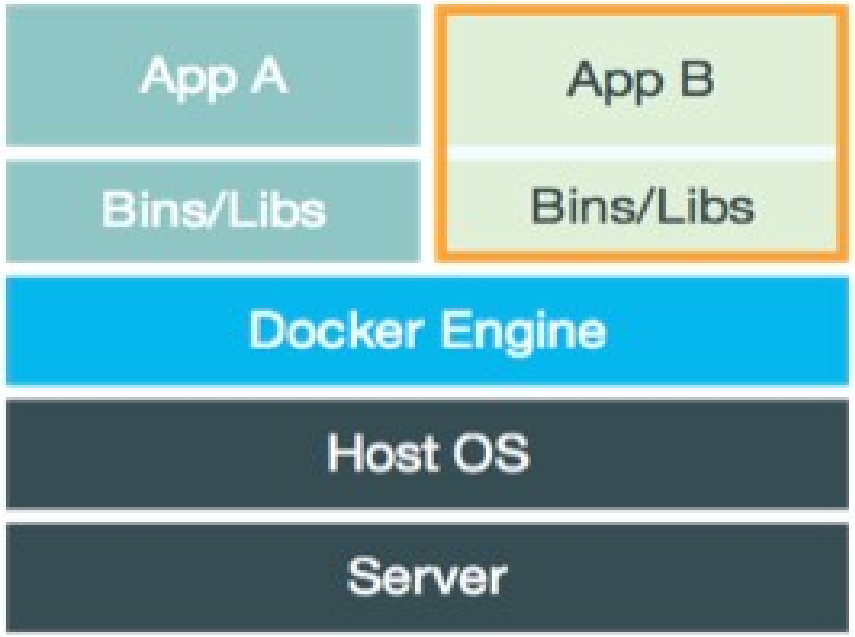


How does Docker work ?

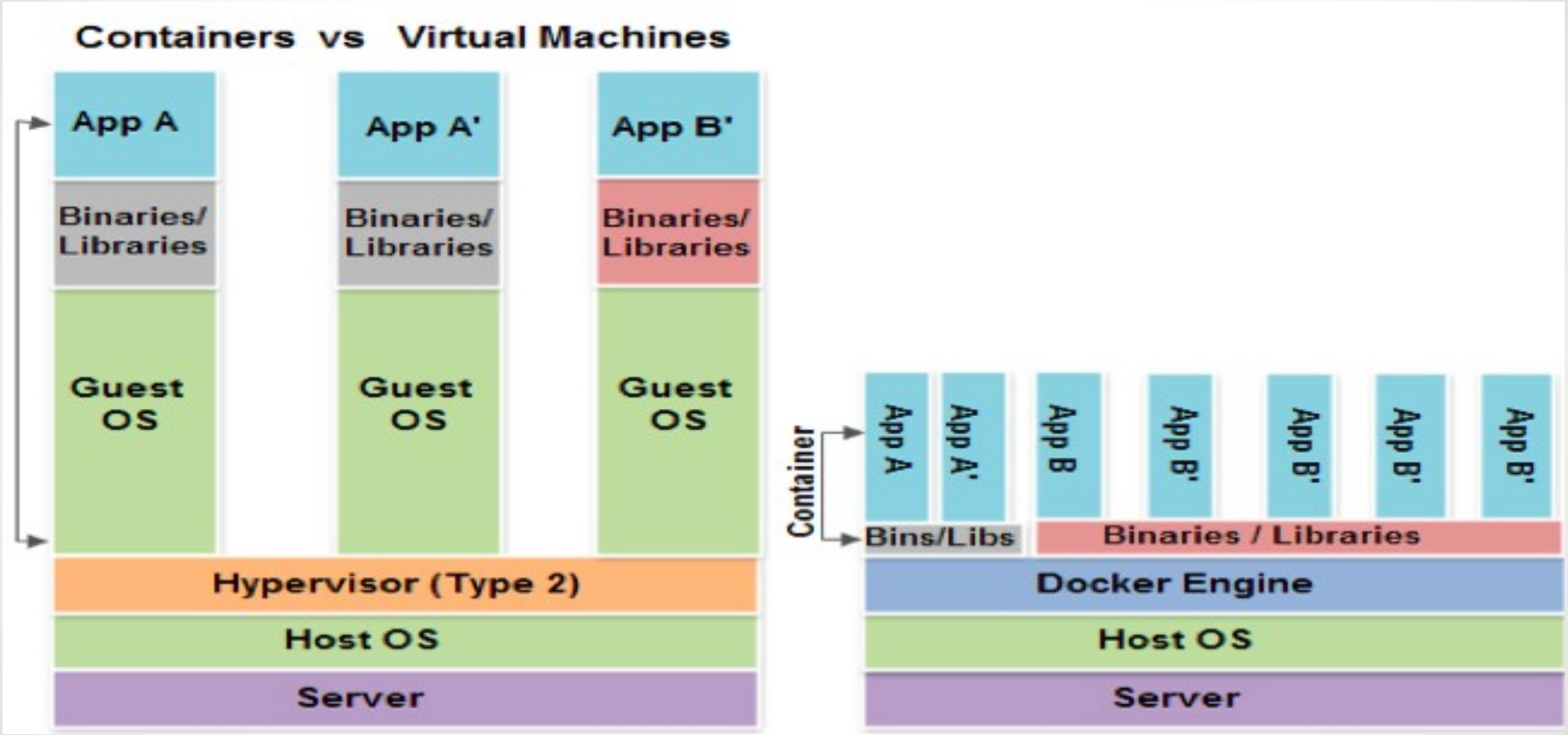
- You can build Docker images that hold your applications
- You can create Docker containers from those Docker images to run your applications.
- You can share those Docker images via Docker Hub or your own registry

Virtual Machine Versus Container……





Virtual Machine Versus Container……

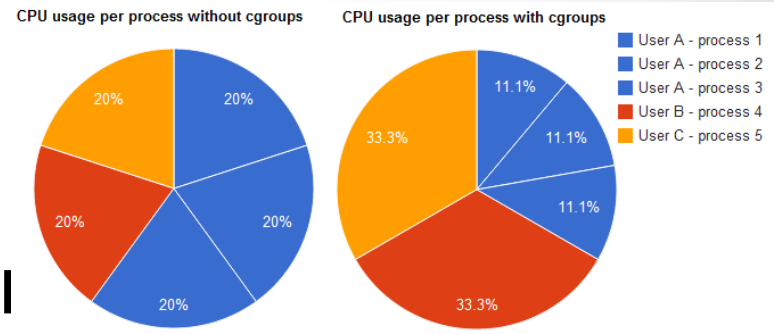


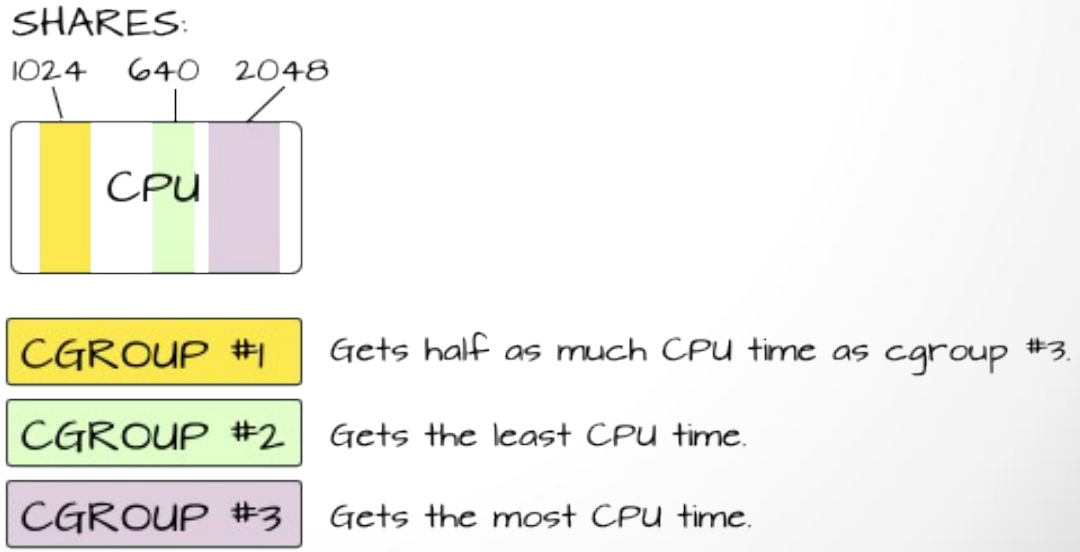
Docker Container Lifecycle ……

- The Life of a Container
  - Conception
    - **BUILD** an Image from a Dockerfile
  - Birth
    - **RUN** (create+start) a container
  - Reproduction
    - **COMMIT** (persist) a container to a new image
    - **RUN** a new container from an image
  - Sleep
    - **KILL** a running container
  - Wake
    - **START** a stopped container
  - Death
    - **RM** (delete) a stopped container
- Extinction
  - **RMI** a container image (delete image)

Linux Cgroups .....

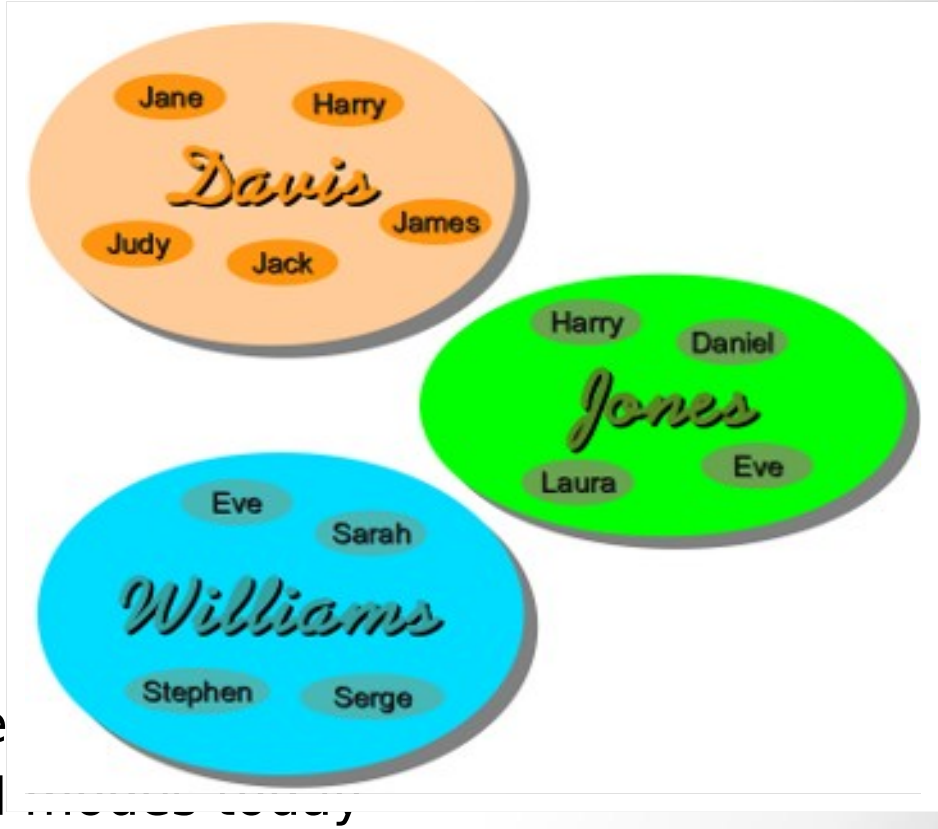
- Kernel Feature
- Groups of processes
- Control resource allocatio
  - CPU
  - Memory
  - Disk
  - I/O
- May be nested





Linux Kernel Namespaces .....

- Kernel Feature
- Restrict your view of the system
  - Mounts (CLONE_NEWNS)
  - UTS (CLONE_NEWUTS)
    - uname() output
  - IPC (CLONE_NEWIPC)
  - PID (CLONE_NEWPID)
  - Networks (CLONE_NEWNET)
  - User (CLONE_NEWUSER)
    - Not supported in Docker ye
    - Has privileged/unprivileged
- May be nested

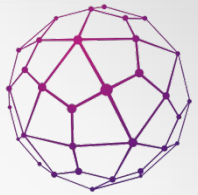


Dockerfile ……

- Like a Makefile (shell script with keywords)
- Extends from a Base Image
- Results in a new Docker Image
- Imperative, not Declarative
- A Docker file lists the steps needed to build an images
- docker build is used to run a Docker file
- Can define default command for docker run, ports to expose, etc

file | 15 lines (11 sloc) | 0.475 kb | Open | Edit | Raw | Blame | History | Delete

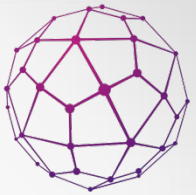
```
1  FROM ubuntu:12.04
2
3  RUN apt-get update
4
5  # Make it easy to install PPA sources
6  RUN apt-get install -y python-software-properties
7
8  # Install Oracle's Java (Recommended for Hadoop)
9  # Auto-accept the license
10 RUN add-apt-repository -y ppa:webupd8team/java
11 RUN apt-get update
12 RUN echo oracle-java7-installer shared/accepted-oracle-license-v1-1 select true | sudo /usr/bin/debconf-set-selections
13 RUN apt-get -y install oracle-java7-installer
14 ENV JAVA_HOME /usr/lib/jvm/java-7-oracle
```



APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

Docker CLI Commands (v1.1.2)……

attach	Attach to a running container	pause	Pause all processes within a container
build	**Build an image from a Dockerfile**	**ps**	**List containers**
commit	**Create new image from container's changes**	pull	Pull image or repo from docker registry
cp	Copy files from containers fs to host	push	Push image or repo to docker registry
diff	Inspect changes on a container's fs	restart	Restart a running container
events	Get real time events from the server	**rm**	**Remove one or more containers**
export	Stream contents of container as tar	**rmi**	**Remove one or more images**
history	Show the history of an image	**run**	**Run a command in a new container**
images	**List images**	save	Save an image to a tar archive
import	Create new fs image from a tarball	search	Search for an image in the docker index
info	Display system-wide information	start	Start a stopped container
inspect	Return low-level info on a container	stop	Stop a running container
kill	**Kill a running container**	tag	Tag an image into a repository
load	Load an image from a tar archive	top	Lookup running processes of a container
login	Login to the docker registry server	unpause	Unpause a paused container
logs	Fetch the logs of a container	version	Show the docker version information
port	Lookup public-facing port	wait	Block and print exit code upon cont exit



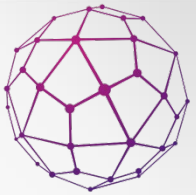
APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

Docker + Mininet

Conainernet

- Containernet is a fork of the famous Mininet network emulator and allows to use Docker containers as hosts in emulated network topologies.
- Enables interesting functionalities to build networking/cloud emulators and testbeds. One example for this is the NFV multi-PoP infrastructure emulator which was created by the SONATA-NFV project and is now part of the OpenSource MANO (OSM) project.
- Containernet is actively used by the research community, focusing on experiments in the field of cloud computing, fog computing, network function virtualization (NFV), and mobile edge computing (MEC).

Create a custom topology



APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

```
          - (c)-
         |      |
(d1) - (s1) - (s2) - (d2)
"""
from mininet.net import Containernet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import info, setLogLevel
setLogLevel('info')

net = Containernet(controller=Controller)
info('*** Adding controller\n')
net.addController('c0')
info('*** Adding docker containers using ubuntu:trusty images\n')
d1 = net.addDocker('d1', ip='10.0.0.251', dimage="ubuntu:trusty")
d2 = net.addDocker('d2', ip='10.0.0.252', dimage="ubuntu:trusty")
info('*** Adding switches\n')
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
info('*** Creating links\n')
net.addLink(d1, s1)
net.addLink(s1, s2, cls=TCLink, delay='100ms', bw=1)
net.addLink(s2, d2)
info('*** Starting network\n')
net.start()
info('*** Testing connectivity\n')
net.ping([d1, d2])
info('*** Running CLI\n')
CLI(net)
info('*** Stopping network')
net.stop()
```

Run emulation and interact with containers

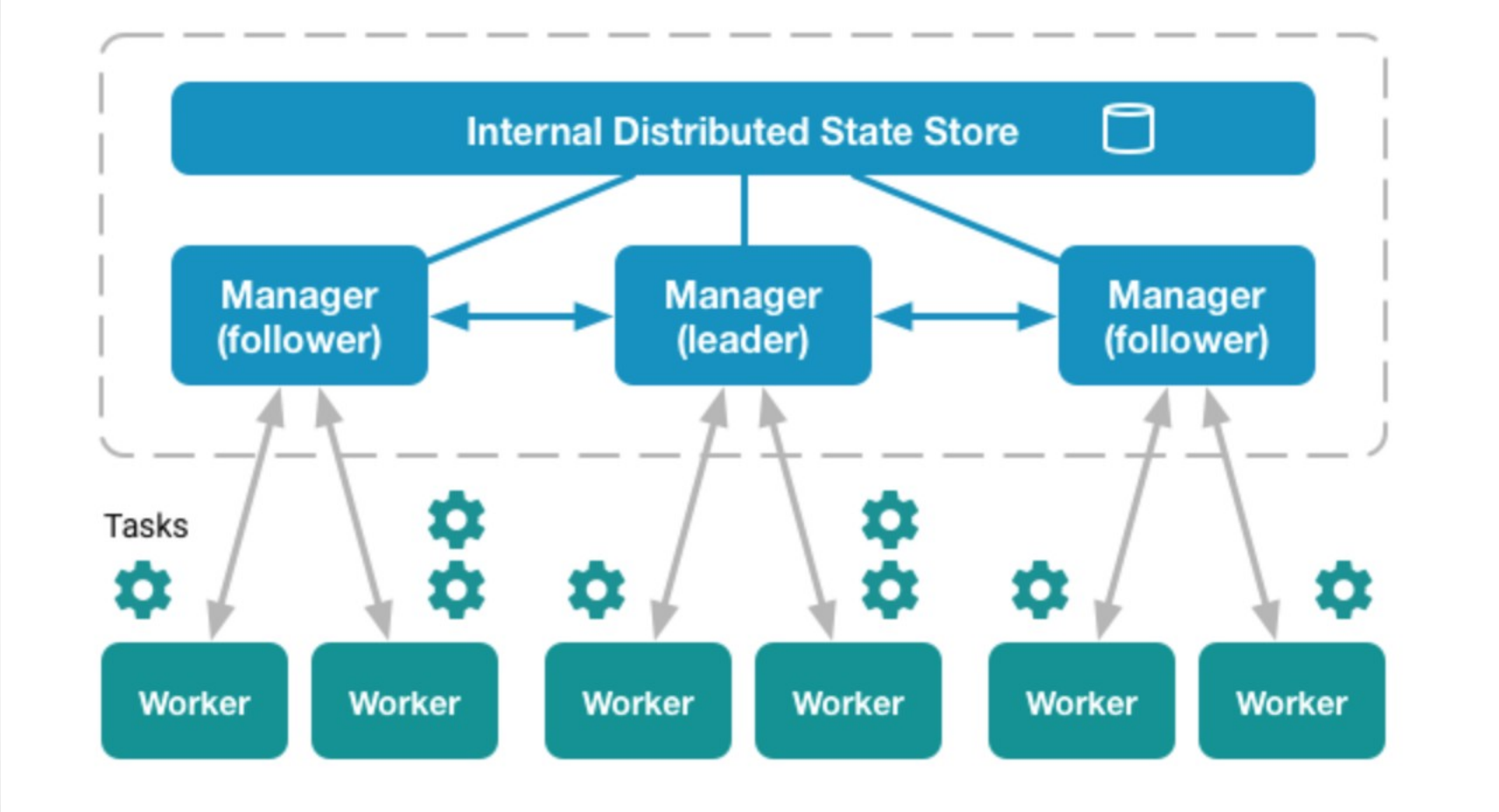
- Containernet requires root access to configure the emulated network described by the topology script:

```
sudo python containernet_example.py
```

- After launching the emulated network, you can interact with the involved containers through Mininet's interactive CLI as shown with the ping command in th

```
containernet> d1 ping -c3 d2
PING 10.0.0.252 (10.0.0.252) 56(84) bytes of data.
64 bytes from 10.0.0.252: icmp_seq=1 ttl=64 time=200 ms
64 bytes from 10.0.0.252: icmp_seq=2 ttl=64 time=200 ms
64 bytes from 10.0.0.252: icmp_seq=3 ttl=64 time=200 ms

--- 10.0.0.252 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 200.162/200.316/200.621/0.424 ms
containernet>
```

Swarm mode overview

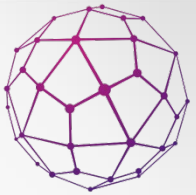


Swarm mode overview

- **Cluster management integrated with Docker Engine:** Use the Docker Engine CLI to create a swarm of Docker Engines where you can deploy application services. You don’t need additional orchestration software to create or manage a swarm.
- **Decentralized design:** Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. You can deploy both kinds of nodes, managers and workers, using the Docker Engine. This means you can build an entire swarm from a single disk image.
- **Declarative service model:** Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack. For example, you might describe an application comprised of a web front end service with message queueing services and a database backend.
- **Scaling:** For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.

- **Desired state reconciliation:** The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state. For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, the manager creates two new replicas to replace the replicas that crashed. The swarm manager assigns the new replicas to workers that are running and available.
- **Multi-host networking:** You can specify an overlay network for your services. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.
- **Service discovery:** Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. You can query every container running in the swarm through a DNS server embedded in the swarm.

- **Load balancing:** You can expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes.
- **Secure by default:** Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes. You have the option to use self-signed root certificates or certificates from a custom root CA.
- **Rolling updates:** At rollout time you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll back to a previous version of the service.

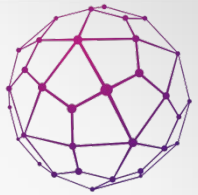


- Open a terminal and ssh into the machine where you want to run your manager node. If you use Docker Machine, you can connect to it via SSH using the following command:

```
$ docker-machine ssh manager1
```

- Run the following command to create a new

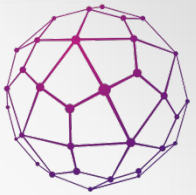
```
$ docker swarm init --advertise-addr <MANAGER-IP>
```



APPLIED RESEARCH CENTER FOR COMPUTER NETWORKS

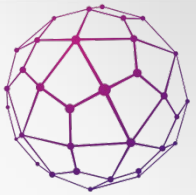
```
$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (dxn1zf6l61qsb1josjja83ngz) is now a m

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wx
    192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager'
```

- the --advertise-addr flag configures the manager node to publish its address as 192.168.99.100. The other nodes in the swarm must be able to access the manager at the IP address.
- The output includes the commands to join new nodes to the swarm. Nodes will join as managers or workers depending on the value for the --token flag.



- Run docker info to view the current state of the swarm:

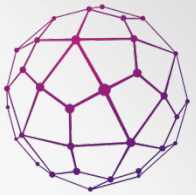
```
$ docker info

Containers: 2
Running: 0
Paused: 0
Stopped: 2
  ...snip...
Swarm: active
  NodeID: dxn1zf6l61qsb1josjja83ngz
  Is Manager: true
  Managers: 1
  Nodes: 1
  ...snip...
```



- Run the docker node ls command to view information about nodes:

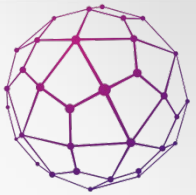
```
$ docker node ls

ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER S
dxn1zf6l61qsb1josjja83ngz *   manager1  Ready   Active        Leader
```

- The * next to the node ID indicates that you're currently connected on this node.
- Docker Engine swarm mode automatically names the node for the machine host name. The tutorial covers other columns in later steps.



APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

- https://habr.com/ru/company/redmadrobot/blog/318866/
- https://docs.docker.com/swarm/overview/
- https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/



APPLIED
RESEARCH
CENTER FOR
COMPUTER
NETWORKS

Thank You for Attention!