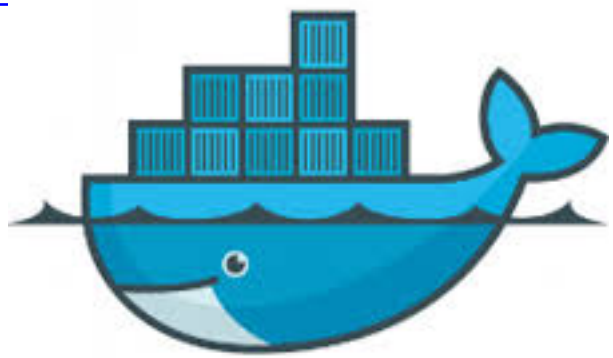
# Containers

https://linuxcontainers.org/lxc/getting-started

https://docs.docker.com/linux/started/



# Outline

- Motivation
- Linux Containers
  - Namespaces
  - Cgroups
- AUFS
- Docker

# Motivation

- sVM can be too “heavy” as deployment solution
  - Still necessary from the administrator perspective
  - … perhaps not so much from the developer’s

- Containers
  - Shipping containers changed world trade
    - *By sharply cutting costs and enhancing reliability, container- based shipping enormously increased the volume of international trade and made complex supply chains possible. (New York Times, 2006)*
  - Containers will change the world, too
    - *The goal of a Standard Container is to encapsulate a software component and all its dependencies in a format that is self- describing and portable, so that any compliant runtime can run it without extra dependency, regardless of the underlying machine and the contents of the container.*
  - The “goal” is different to sVM (server consolidation)

# Linux Containers

- LXC let you “run” a Linux “within” another Linux without VMM
- In fact, a container is just a group of processes running in a isolated environment
  - Inside the container looks like a VM
  - Outside the container looks like a bunch of normal processes
- No VM implies, lower overhead, lower memory requirements, faster boot and shutdown..
- Still weak isolation (regular OS)
  - In a VM the “surface of attack” is the VMM
  - In a container is the whole kernel

# How to work with LXC

- Similar to Xen
  - apt-get install lxc
- lxc-create
  - Setup a container (root filesystem and config)
- lxc-ls -f
  - List available containers (No mem CPU)
- lxc-start
  - Boot the container (by default, you get a console)
- lxc-console
  - Attach a console (if you started in background)
- lxc-stop
  - Shutdown the container
- lxc-destroy
  - Destroy the file system created with lxc-create

# Creating a container

- Preparing the system (using back-ports kernel)
  - `apt-get –t wheezy-backports install lxc`
  - `echo "cgroup /sys/fs/cgroup cgroup defaults 0 0" >> /etc/fstab`
  - `mount –a`
  - `lxc-checkconfig`

- Creating a container (no network)
  - `lxc-create -t debian -n my_jessie`
  - `lxc-start -n my_jessie(CTRL+a q to exit)`
  - `lxc-info -n my_jessie`
  - `lxc-ls –f`
  - `lxc-attach -n my_jessie`

- Edit config in /var/lib/lxc/my_jessie/config

- Support for many templates (and unprivileged)
  - Fedora, Arch, suse, ubuntu

# Preparing the network (LibVirtNetwork)

- Network (Libvirt approach)
  - `apt-get –t wheezy-backports install ebtables dnsmasq libvirt-bin`
  - `virsh net-info default`
  - `service dnsmasq stop`
  - `virsh net-start default`
  - `ifconfig –a`

- By default (in wheezy) no network
  - Add to `/etc/lxc/default.conf`

```
lxc.network.type = veth
lxc.network.name = eth0
lxc.network.link = br0
lxc.network.ipv4 = 10.0.3.138/24
lxc.network.flags = up
```

  - Modify `/var/lib/lxc/my_jessie later`

# lxc-create

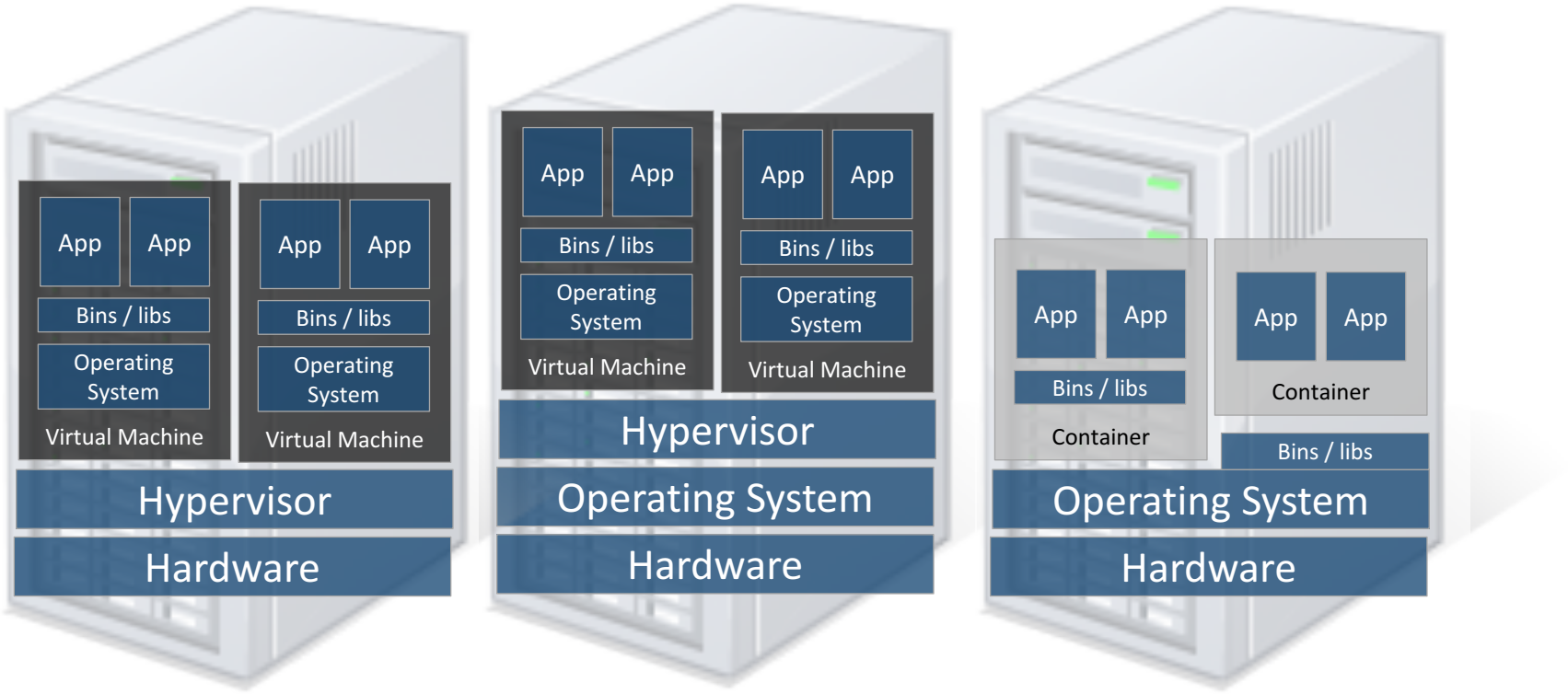
- The macro is equivalent to xen-image-create
  - Fully configurable
  - We can use LVM
  - All the configuration parameters are kept in `/var/lib/lxc`
- Many templates (fedora, arch,…) some of them requires extra work
- Ubuntu, centos seems to be working ok

# Hypervisors vs. Linux Containers

**Containers share the OS kernel of the host and thus are lightweight.**
**However, each container must have the same OS kernel.**

**taken From [Bodem Russell]**

**Containers are isolated, but share OS and, where appropriate, libs / bins.**



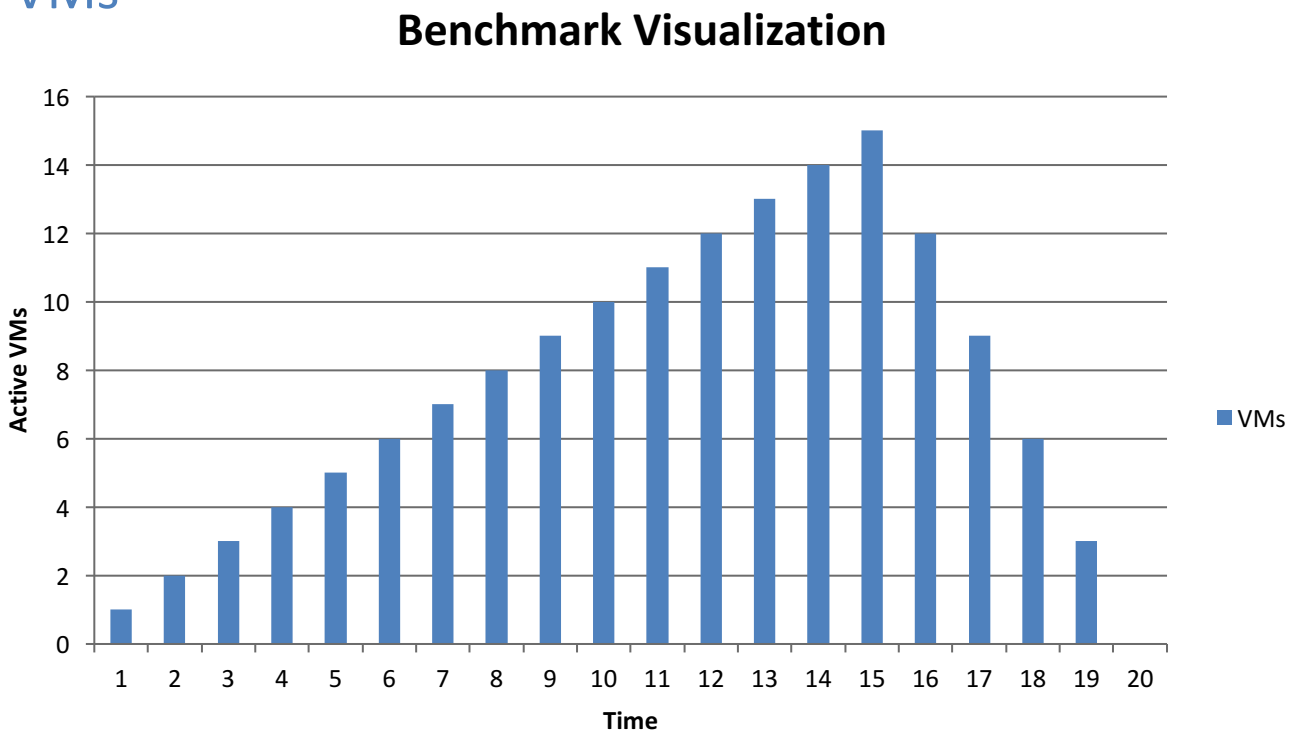

**Type 1 Hypervisor** **Type 2 Hypervisor** **Linux Containers**

# Why Linux Containers (LXC)

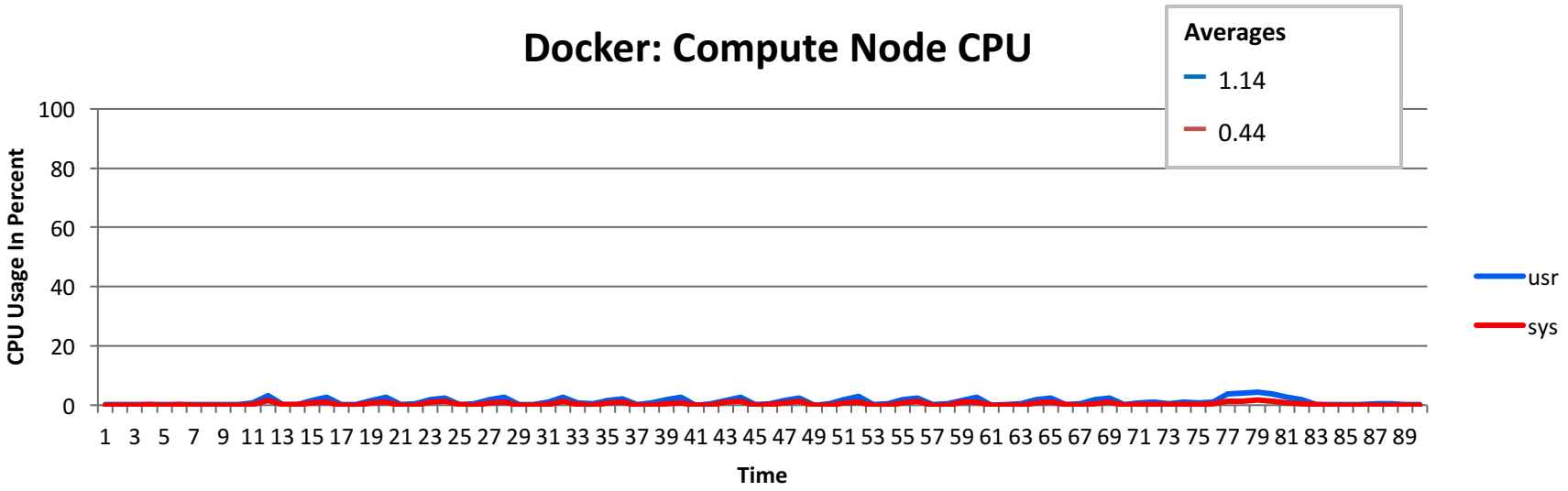
- Fast
  - Runtime performance near bare metal speeds
  - Management operations (run, stop , start, etc.) in seconds / milliseconds
- Agile
  - VM-like agility – it's still "virtualization"
  - Seamlessly "migrate" between virtual and bare metal environments
- Flexible
  - Containerize a "system"
  - Containerize "application(s)"
- Lightweight
  - Just enough Operating System (JeOS)
  - Minimal per container penalty
- Inexpensive
  - Open source – free – lower TCO
  - Supported with out-of-the-box modern Linux kernel
- Ecosystem
  - Growing in popularity
  - Vibrant community & numerous 3rd party apps

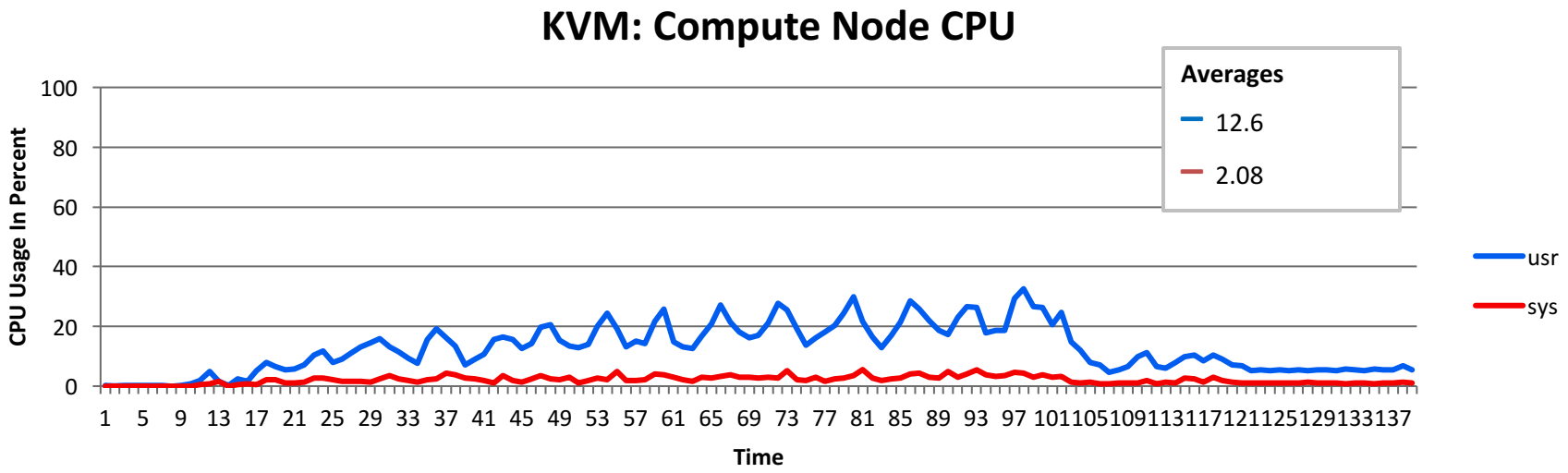
# Cloudy Performance: Serial VM Boot

- Benchmark scenario overview
  - Boot VM via OpenStack nova
  - Wait for VM to become active
  - Repeat the above steps for a total of 15 VMs
  - Delete all VMs

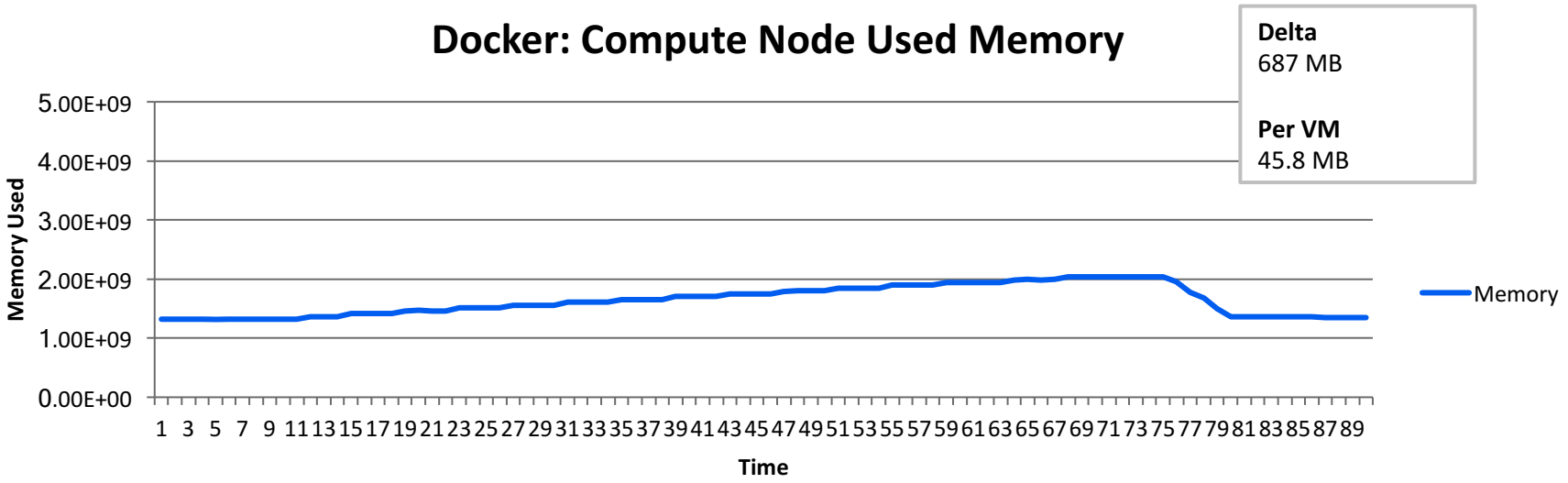


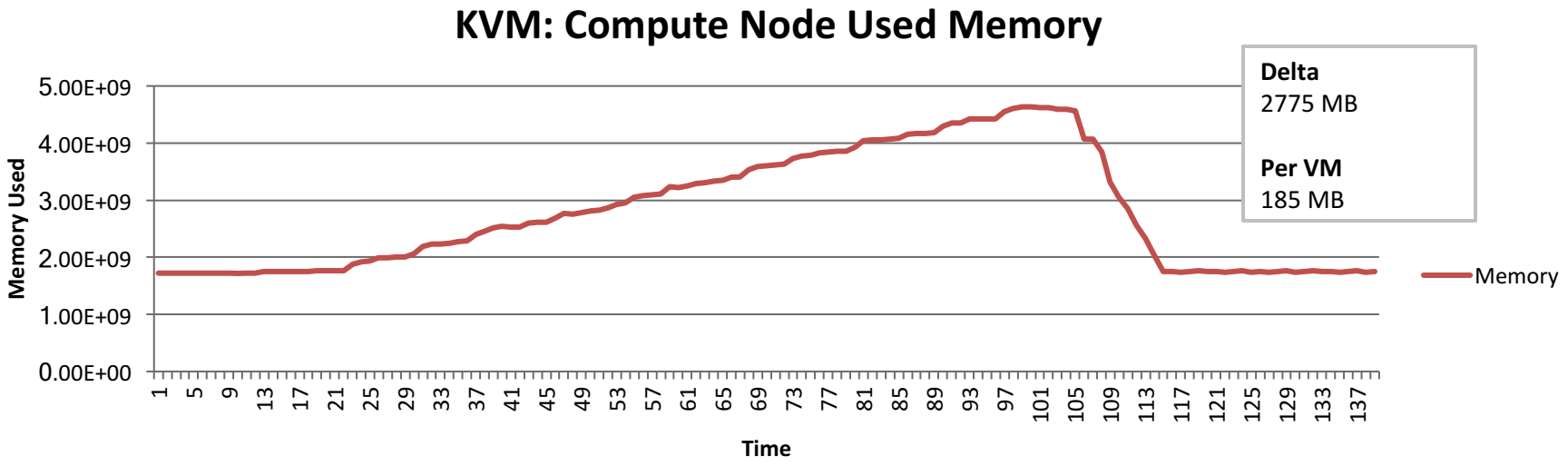
# Cloudy Performance: Serial VM Boot



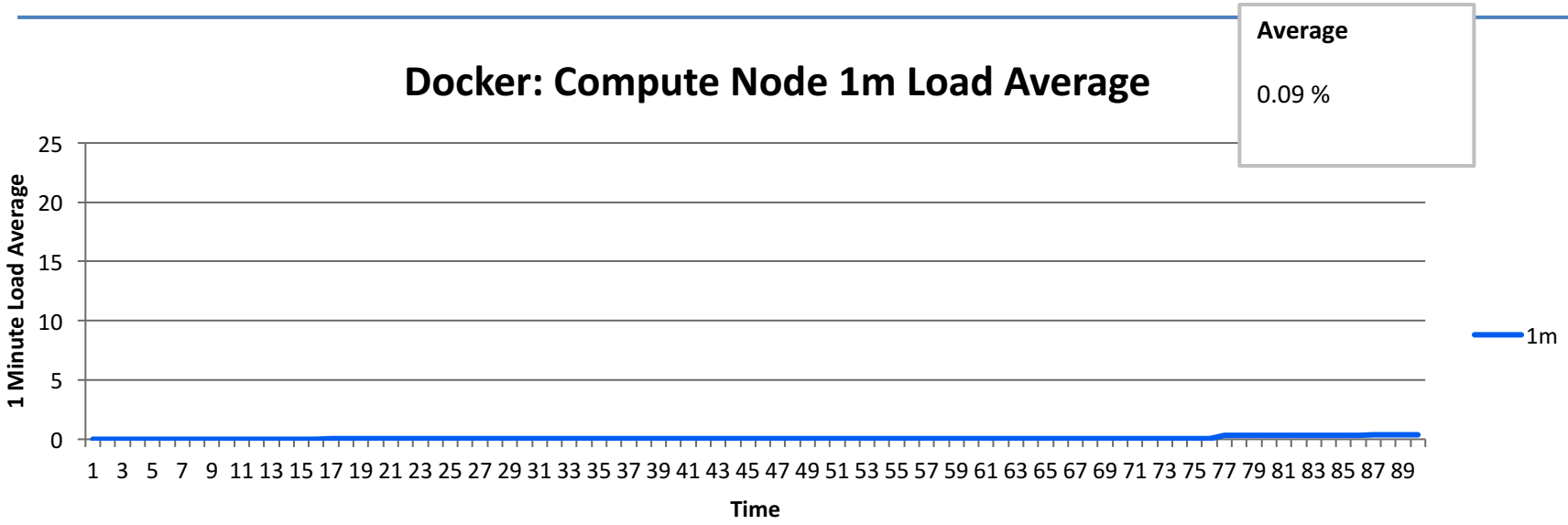



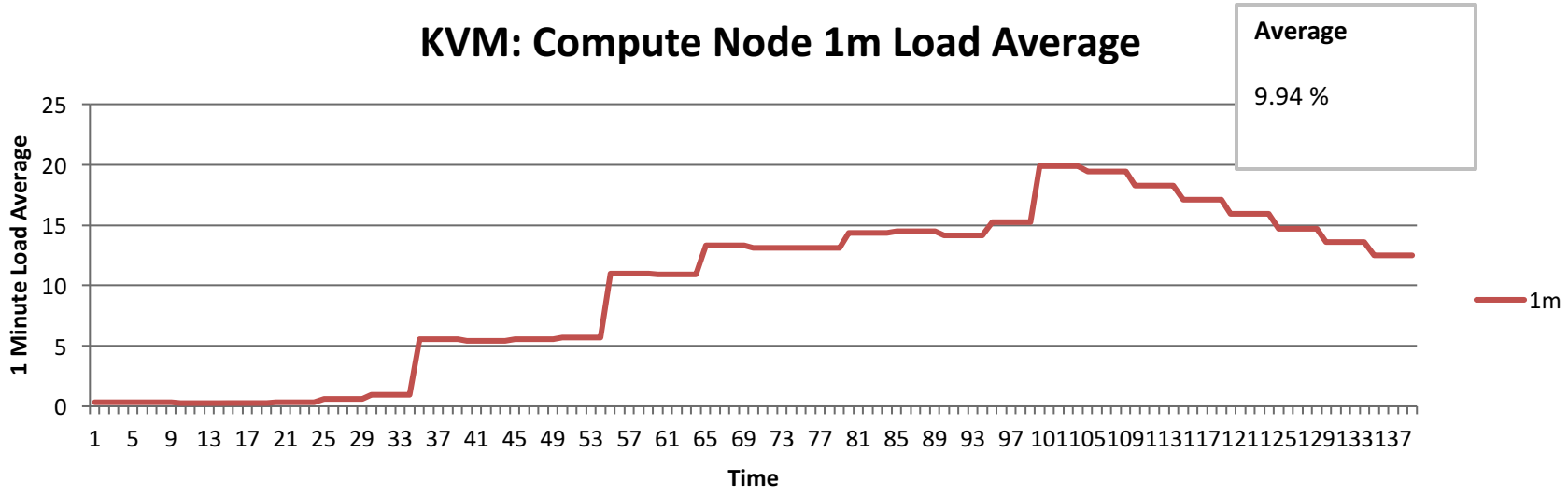
# Cloudy Performance: Serial VM Boot



# Cloudy Performance: Serial VM Boot



# And still they are *almost* VM

- Each container might have
  - Its own network interface (and IP address)
- His own “root”
  - We can mix CentOS/Debian/Ubuntu/Mint/… in the same system
- Isolation is wekaer
  - OS protect containers
  - Larger surface of attack
- Resource sharing QoS
  - Similar techniques used in Xen are applicable here
- No live migration
  - Use virtualized containers! (or a container hypervisor LXD)

# Use cases for Containers

- Development
  - Work in your computer inside a container
  - Deploy the container and ready to go

- Simplifies
  - Continuous integration
    - Container based runners are easy to handle
  - Deployment
    - Reproduce the same environment in development and production

- The container can be seen as a PAAS

# How LXC works?

- Namespaces + control groups
  - lxc-checkconfig

- Namespaces
  - Equivalent to chroot but for anything (not just disk)
    - *Partition essential kernel structures to create virtual environments (PID, network, disk, UID, IPC, etc…)*
  - Uses a kernel system call called “clone”
    - Much mature in kernels 3.8+
    - Don use wheezy original kernel!
  - Can be nested!
  - Namespace content is only visible to it
    - To communicate containers use network
  - SDN networks using *libvirt* and *openvswitch*

# Control Groups (cgroups)

- "Equivalent" to ulimit but for a group of processes
  - And no bypasseable by the user!

- Some basics
  - Create cgroup
    - **`mkdir /sys/fs/cgroup/pepito`**
  - Move a process to a cgroup
    - **`echo PID > /sys/fs/cgroup/pepito/tasks`**
  - We can control all the PIDs in the cgroup at once
  - LXC are right there
    - **`/sys/fs/cgroup/lxc/`**
  - v.gr. If we want to restrict to one core a container
    - **`echo 0 >  /sys/fs/cgroup/lxc/my_jessie/cpuset.cpus`**

# OverlayFS

- Allows to mount two or more fs/dir in a single dir, limiting the changes to one of them
  - Supported natively by new kernels
  - Debian 7 only has support for AUFS

- Example: AUFS

```
root@aos:# mkdir /tmp/p1 /tmp/p2
root@aos:# echo "t1"> /tmp/p1/t1; echo "t2">/tmp/p2/t2
root@aos:# mount -t aufs -o dirs=/tmp/p1:/tmp/p2 /mnt/
root@aos:# echo "t3" > /mnt/t3 "goes to p1"
```

- Mimics the functionality of snapshots in LVM

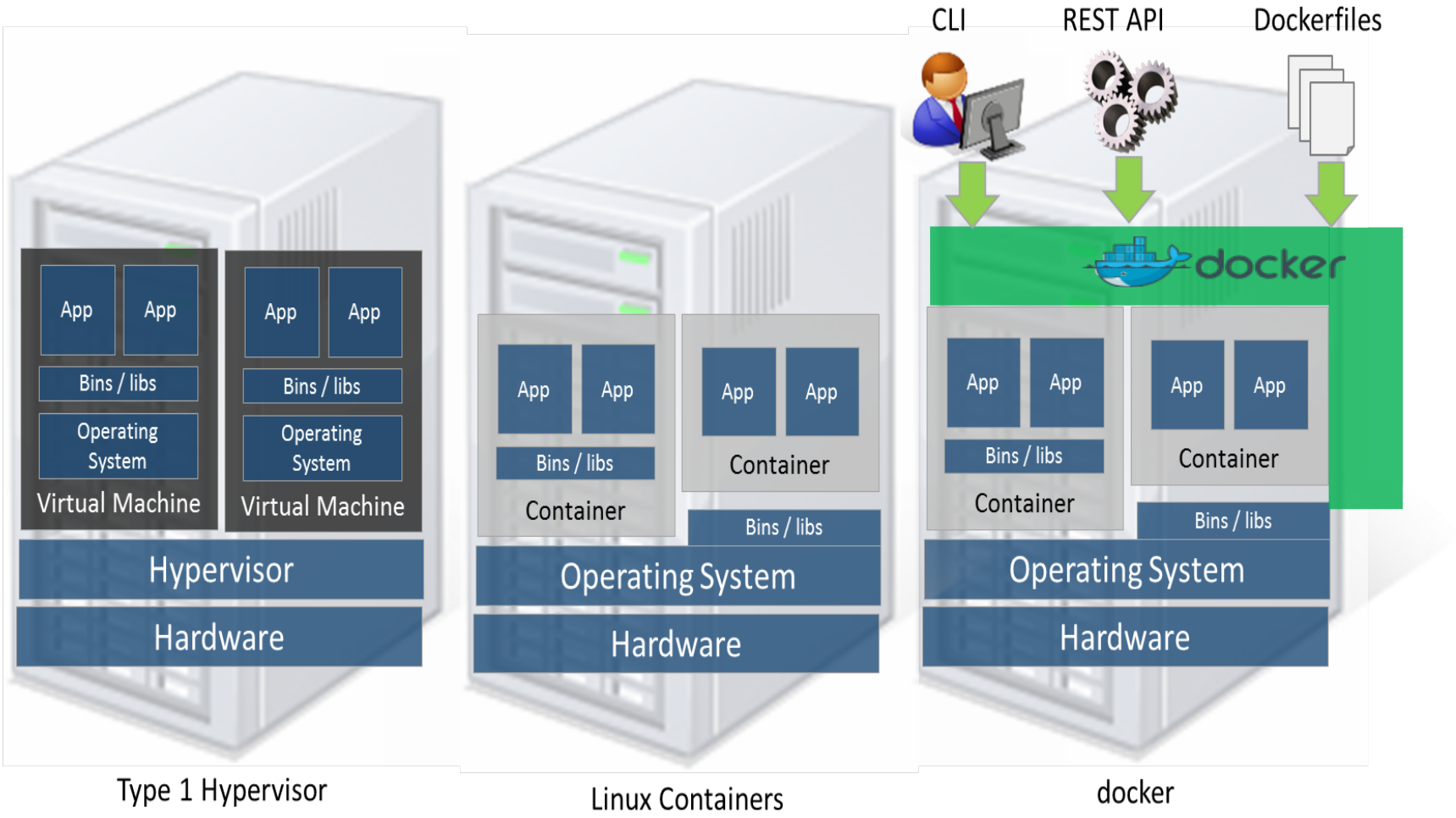
- Yep: You can use that to mount concurrently hundreds of containers with the same image!

- We still can use LVM (or ZFS, BTRFS) to do snapshots

# Sharing the same rootfs across lxc

- Clone using snapshots+aufs
  - `lxc-clone -B aufs -s my_jessie my_clon`
- Just do a union-fs over
  - `/var/lib/lxc/my_jessie/rootfs/`
- The dependency is kept in
  - `/var/lib/lxc/my_clon/lxc_rdepends`
- Changes in my_clon in
  - `/var/lib/lxc/my_clon/delta0/`
- Similar technique is used for snapshots
  - `lxc-snapshot –n my_clon` creates a snapshot in `/var/lib/lxcsnaps`

# Hypervisor VM vs. LXC vs. Docker LXC



# What it is Docker?

- Open Source project using LXC + union-FS (**AUFS**, LVM, BTRFS,…) + DVCS (i.e. something like git)
  - *automates the deployment of applications as highly portable, self-sufficient containers which are independent of hardware, language, framework, packaging system and hosting provider*
- Written in Go
- Not official support in Debian 7 repos (in jessie Backports) :-(
- **Docker is git for VMs (not really VMs:-)**

# How to Deploy Docker in Debian7

- Doing (requires 3.16 kernel)

```
curl -sSL https://get.docker.com/ | sh
```

- Just doit! (regular users too via sudo)

```
docker run -t -i ubuntu bash
apt-get update
apt-get install -qqy mysql-server
exit
docker ps –a
docker start –i 9b37
docker diff 9b37
docker commit 9b37 vpuente/mysql
<<a regular user can also doit!!!>>
```

- If it is connected with DockerHub

```
docker push vpuente/mysql
```

- From other server

```
docker run -t -i vpuente/mysql bash
```

- DockerHub is a repository of Docker containers
- Container construction can be automatized with DOCKERFILE

# Automated docker image customization

- Dockerfile

```
FROM ubuntu:latest
MAINTAINER Valentin <vpuente@unican.es>
RUN apt-get update && apt-get install -y ruby ruby-dev
RUN gem install sinatra
```

- Put “Dockerfile” in a local dir and run
  - docker build vpuente/ubuntu_sinatra:version2 .

- Assumes we start from latest ubuntu

# The most likely usage scenario

- "Containerized" apps
  - `docker run -d -P training/webapp python app.py`
  - `docker ps –l`
  - `docker logs <id>`
  - `docker top <id>`
  - `docker stop <id>`
  - `docker start <id>`
  - `docker rm <id>`

- Add a external storage to the container
  - `docker run -d -P --name web -v /tmp/webapp:/webapp training/webapp python app.py`

# Network

- Basic commands
  - `docker network ls`
  - `docker run -itd --name=networktest ubuntu`
  - `docker network inspect bridge`
  - `docker network disconnect bridge networktest`
- Assign a ip to the container
  - `docker run --rm -it --net iptastic --ip 203.0.113.2 nginx`
- Bind real port 8080 to 80 inside the container
  - `docker run --rm -it --publish 8080:80 nginx`
- Iptables

# DockerHub: The GitHub of VM (not really :)

- I need to deploy gitlab (free clone of github)
  - Postgres+ruby-on-rails+…=hours of work

```
docker pull gitlab/gitlab-ce
docker images
docker run --detach \
    --hostname gitlab.example.com \
    --publish 8443:443 --publish 8080:80 --publish 2222:22 \
    --name gitlab \
    --restart always \
    --volume /srv/gitlab/config:/etc/gitlab \
    --volume /srv/gitlab/logs:/var/log/gitlab \
    --volume /srv/gitlab/data:/var/opt/gitlab \
    gitlab/gitlab-ce:latest
```

- The whole system is ready to use!
  - (gitlab it is listening in localhost 8080 user: **root** password: **5iveL!fe**)
  - Directory /srv in the host keeps the data

- Use your own server (Registry)
  - By default Docker uses central Registry at index.docker.io
  - You can install your own Registry following (which is a docker … uh!)
    - https://docs.docker.com/registry/deploying/

# Underlying Structure

- Images
  - /var/lib/docker/
    - Not exactly lxc “layout”
  - Uncommitted containers content are persistent (/var/lib/docker/containers) and cgroups
    - To delete all of them
    - **`docker rm $(docker ps -a -q)`**
  - Images are in /var/lib/docker/aufs or /var/lib/docker/images/
  - Images and containers
    - Images are the “changetsets” committed. Similar to LVM in Xen. (used with “docker run”)
    - Containers are the status of running  underlying images. Similar to DomU in Xen (used with “docker start/attach”)

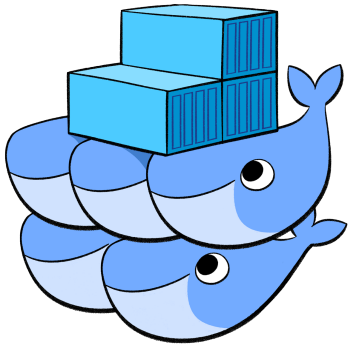
- Cgroups
  - **`/sys/fs/cgroup/docker`**
  - Can control memory/cpus/etc…
  - In linux.3.16 it is a bit glitchy (cpusets are empty after reboots)
    - **`echo 1 > /sys/fs/cgroup/cgroup.clone_children`**
    - **Don’t mount cgroup in /etc/fstab!!**

- No need to dig here: all through “**`docker`**” command

- **Best way (from a administration stand-point) is to run containers in a true VM**

# Beyond (single) docker

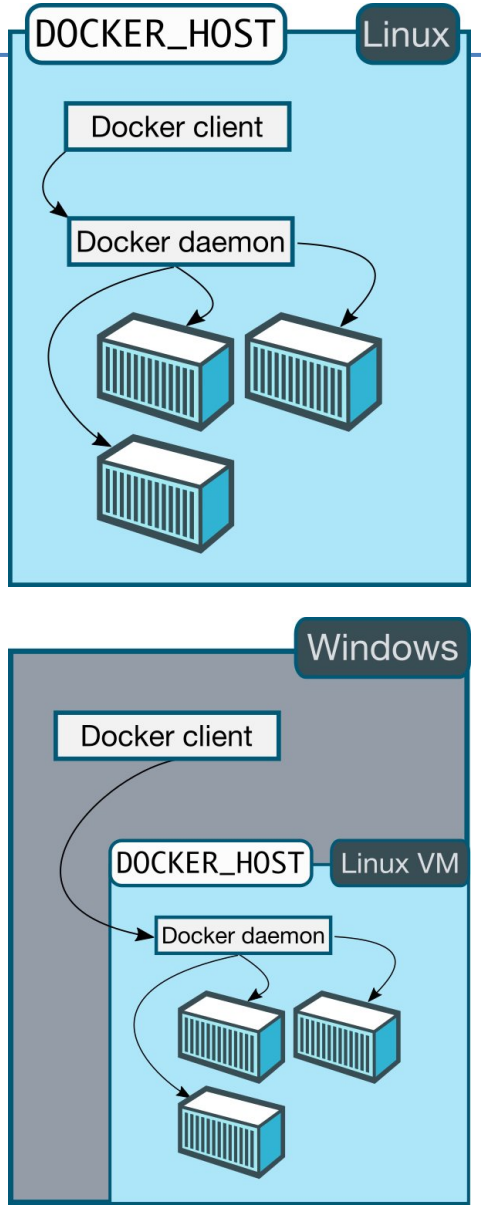
- Multi-container Applications
  - v.gr. Isolated Front-end (webserver)+Back-end (DBMS)
  - v.gr. Multiserver NoSQL (Cassandra, redis, etc…)
  - …
  - Docker compose
  - Docker machine



- Container Orchestration
  - Kubernetes , Docker swarm, …
- Other containers
  - rkt, openVZ, …
- Container optimized OS
  - CoreOS, photon, …

# Docker OS support

- Docker can be used in other OS
  - OSX
  - Windows
- It is based in Linux
  - Uses Virtual Box (toolbox) or Hyperkit (for mac), or Hyper-V (windows) to run the containers
  - All the containers share the same virtual machine
- Vagrant does something “similar” but changing also the host OS
  - Any combination host/client is supported
  - Many tools around



# How to run a Ubuntu in Windows?

- Use Vagrant!
  - Install Vagrant from www.vagrant.com
- Then:
  - `vagrant init hashicorp/precise32`
  - `vagrant up`
  - `vagrant ssh`
- Current Windows10/Linux subsystem don't support "LXC" requirements