State-of-the art containers & hypervisors

Linus Karlsson

Traditional virtual machines

- Virtual computer.
- Virtual processor, memory, storage, etc.
- Each instance runs its own copy of some operating system.
- Provides good isolation between machines.

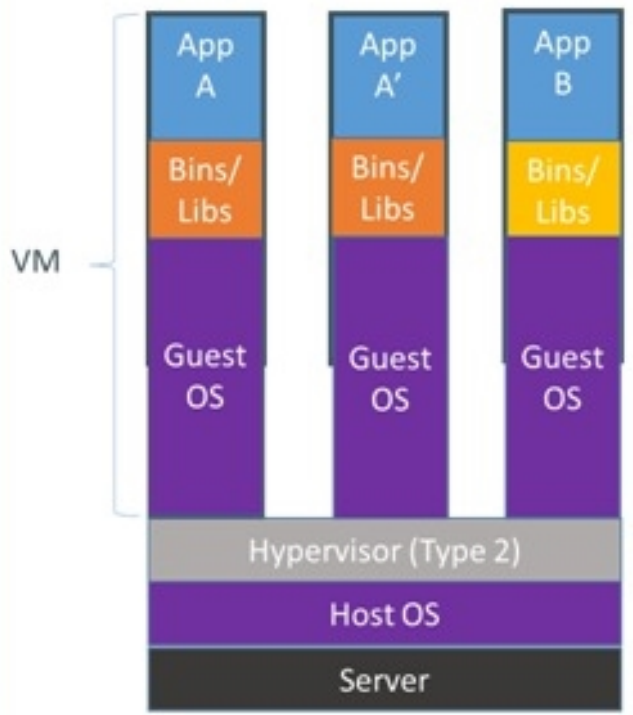


Image: `https://github.com/docker/www.docker.io/tree/master/assets/img/about/docker_vm.jpg`

Operating-system-level virtualization

- Instances called *containers*.
- Does not emulate a virtual machine
  - Each instance is isolated by the (shared) kernel
- Examples are FreeBSD jails, Solaris Zones, LXC.

FreeBSD jails

- `chroot` on steroids.
- A jail instead has:
  - Limited file system access (like `chroot`)
  - Its own set of users (include its own root account)
  - Own networking subsystem (hostname, IP-address, etc.)
- Host `root` have full access to every jail.
- `root` inside jail cannot escape from jail.

FreeBSD jails

On host

```
root@host:/usr/jails/info.example.com # ls srv/http/
386             errors          kerbprotected   paperjam
chatlog         index.html      labqueue        test.html
```

Inside jail

```
root@info:/ # ls srv/http
386             errors          kerbprotected   paperjam
chatlog         index.html      labqueue        test.html
```

FreeBSD jails

On host

```
root@host:/ # ifconfig xn0
...
inet 10.0.0.200 netmask 0xffffff00 broadcast 10.0.0.255
inet 10.0.0.202 netmask 0xffffffff broadcast 10.0.0.202
inet 10.0.0.203 netmask 0xffffffff broadcast 10.0.0.203
inet 10.0.0.204 netmask 0xffffffff broadcast 10.0.0.204
inet 10.0.0.201 netmask 0xffffffff broadcast 10.0.0.201
...
```

Inside jail

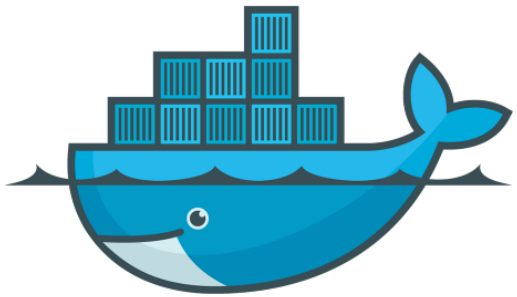
```
root@info:/ # ifconfig xn0
...
inet 10.0.0.204 netmask 0xffffffff broadcast 10.0.0.204
...
```

Jail isolation

- Force jail to only use some CPU cores
- Limit disk space used (ZFS)
  - However, no limitation on disk I/O
- Limit memory and CPU usage (`rctl`)
  - Requires kernel recompilation

Docker

- Lot of fuzz about Docker recently
- Nice whale
- Process-level virtualization



Docker

- Launches user-space processes
- Every process is isolated from each other
  - Processes in a container cannot see/interact with processes of other containers
  - Each container has its own network stack
  - Has its own file system
  - Implemented using
    - Linux namespaces
    - Union mounted filesystem
    - cgroups

Docker

- Launches user-space processes
- Every process is isolated from each other
  - Processes in a container cannot see/interact with processes of other containers
  - Each container has its own network stack
  - Has its own file system
  - Implemented using
    - Linux namespaces
    - Union mounted filesystem
    - cgroups

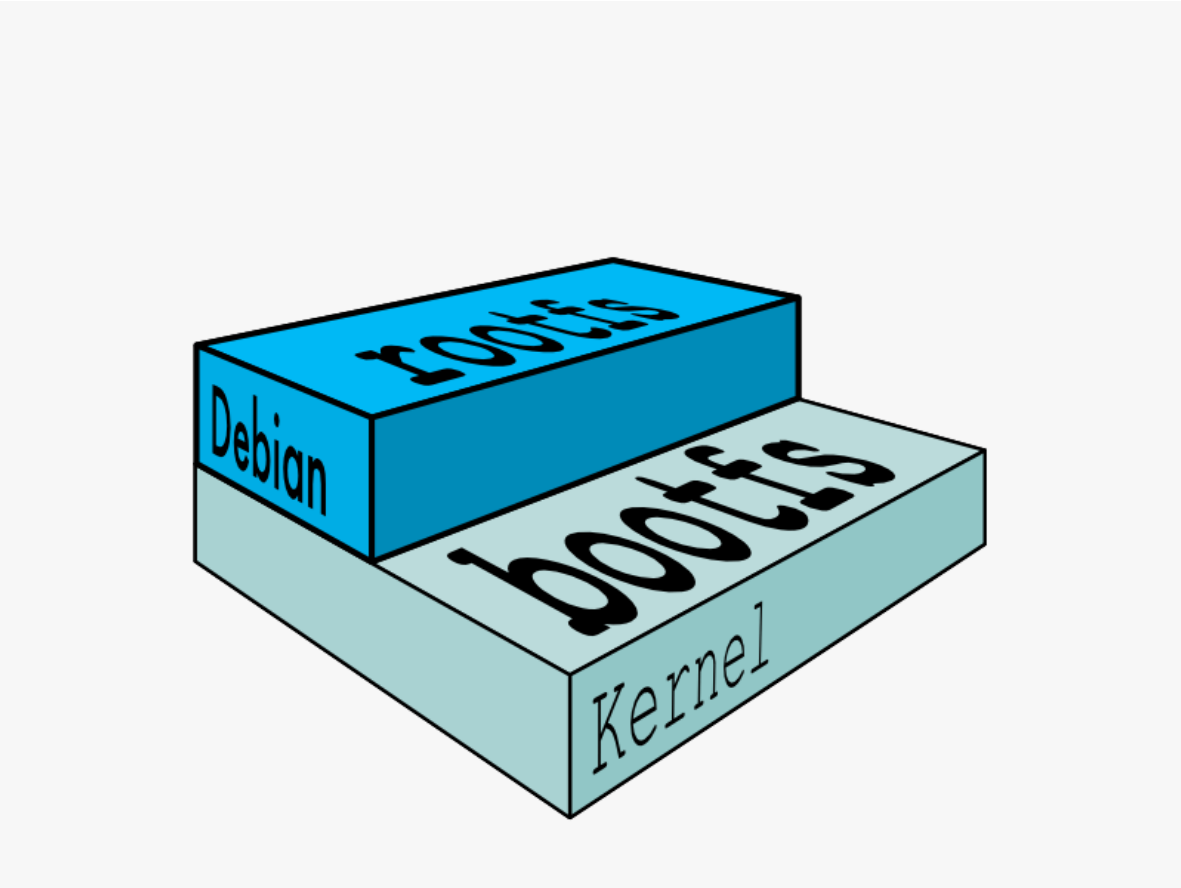
So, isn't this exactly operating-system-level virtualization?

Docker

Process-level virtualization: From the application's perspective, we abstract away differences from the underlying system.

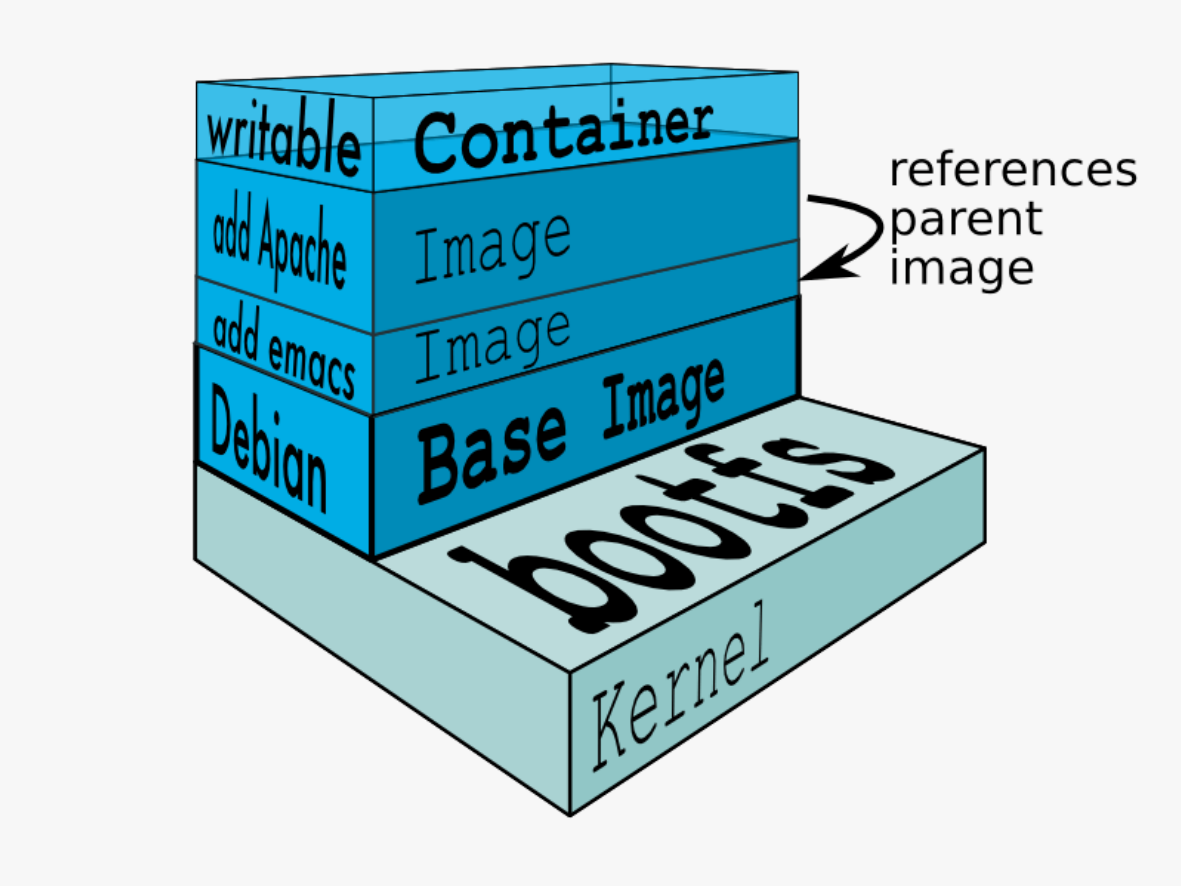
- Makes it easy to move applications between hosts.
  - Moving from development machine to server.
- This is mainly due to the use of Docker images.

Docker images



Images from: `http://docs.docker.com/terms/image/`

Docker images

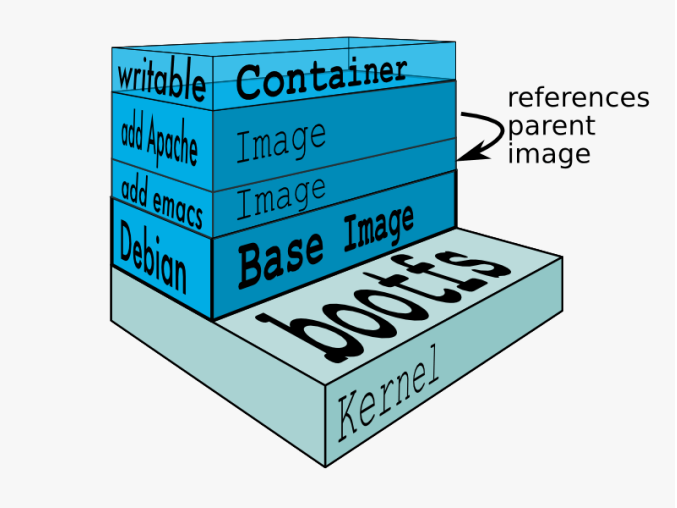


Docker can automatically fetch all images down to the base image.

Images from: `http://docs.docker.com/terms/image/`

Containers

- Images are read-only
- Union mount merges the images together with a writable top layer
- Copy-on-write
- Can be stopped, paused, restarted
- Container can be converted to image for re-use
- Migration using CRIU (checkpoint/restore), still "pre-alpha".



Images from: `http://docs.docker.com/terms/image/`

Building Docker images

- Can be built by manually.
  1. Start container
  2. Make change
  3. Commit
  4. Repeat
- Quite cumbersome
- Can be automated by using Dockerfiles

Dockerfiles

Dockerfile

```
FROM ubuntu:14.04
MAINTAINER Linus Karlsson <linus.karlsson@eit.lth.se>

RUN apt-get update && apt-get install -y python-pip
RUN pip install Flask
ADD server.py /srv/server.py

EXPOSE 5000
CMD python /srv/server.py
```

Docker demo

1. Ubuntu 14.04 and /bin/bash

```
# docker run -it ubuntu:14.04 /bin/bash
```

2. Build image from my previous Dockerfile

```
# docker build -t linus/moln .
# docker run -d -p 5000:5000 linus/moln
# docker ps
```

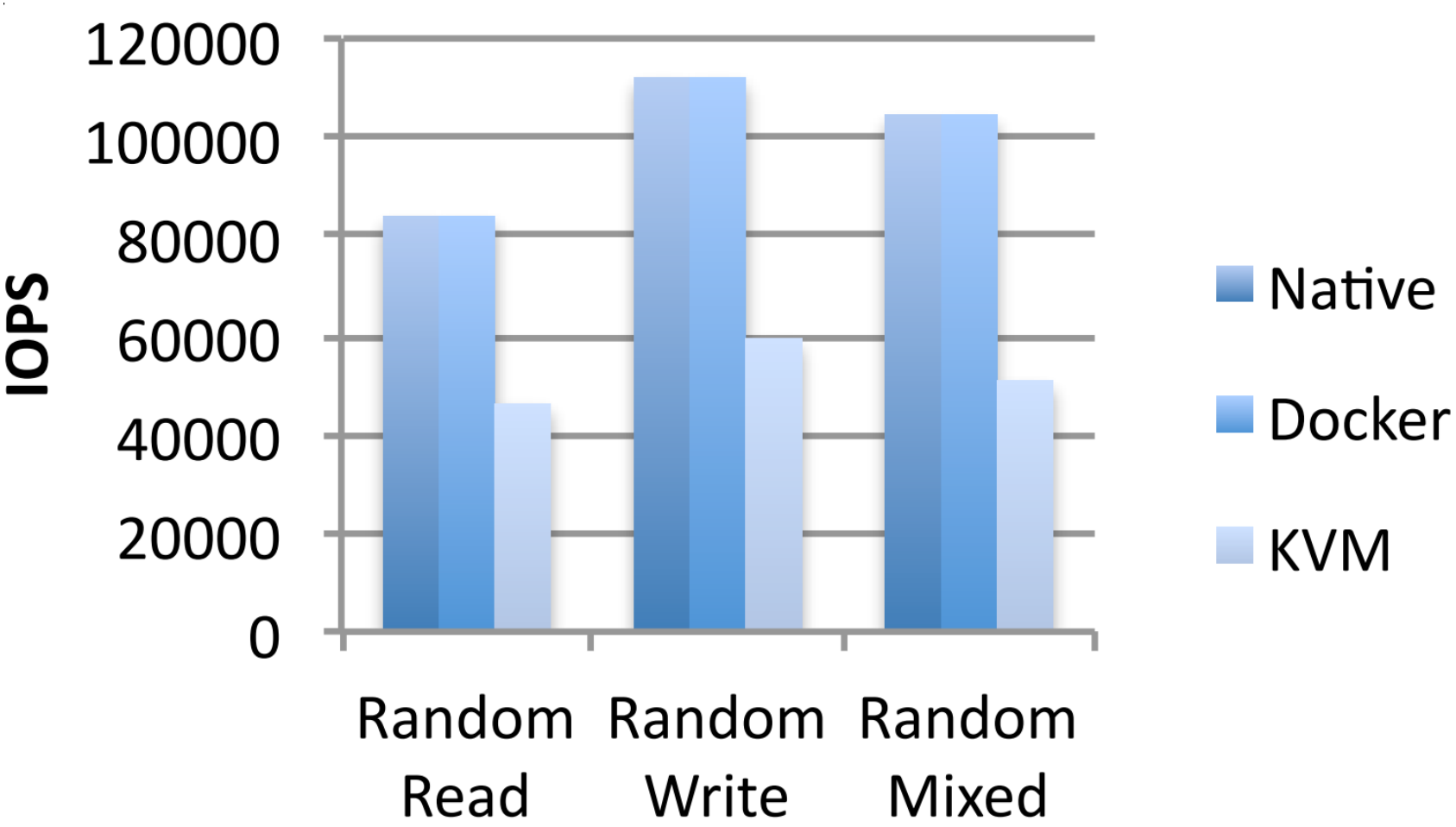
Docker container isolation

- Force Docker container to only use some CPU cores
- Limit CPU usage share
- Limit disk I/O (read/write speed)
- Some disk space limits
  - By switching storage backend from default
  - Only limits sum of images + writable layer
  - Other creative options are loopback mounts etc.
- Limit memory usage

Docker performance

- Startup times: seconds for Docker.
- Runtime performance:
  - Depends on workload.
  - CPU, and RAM: minor differences.
  - I/O overhead

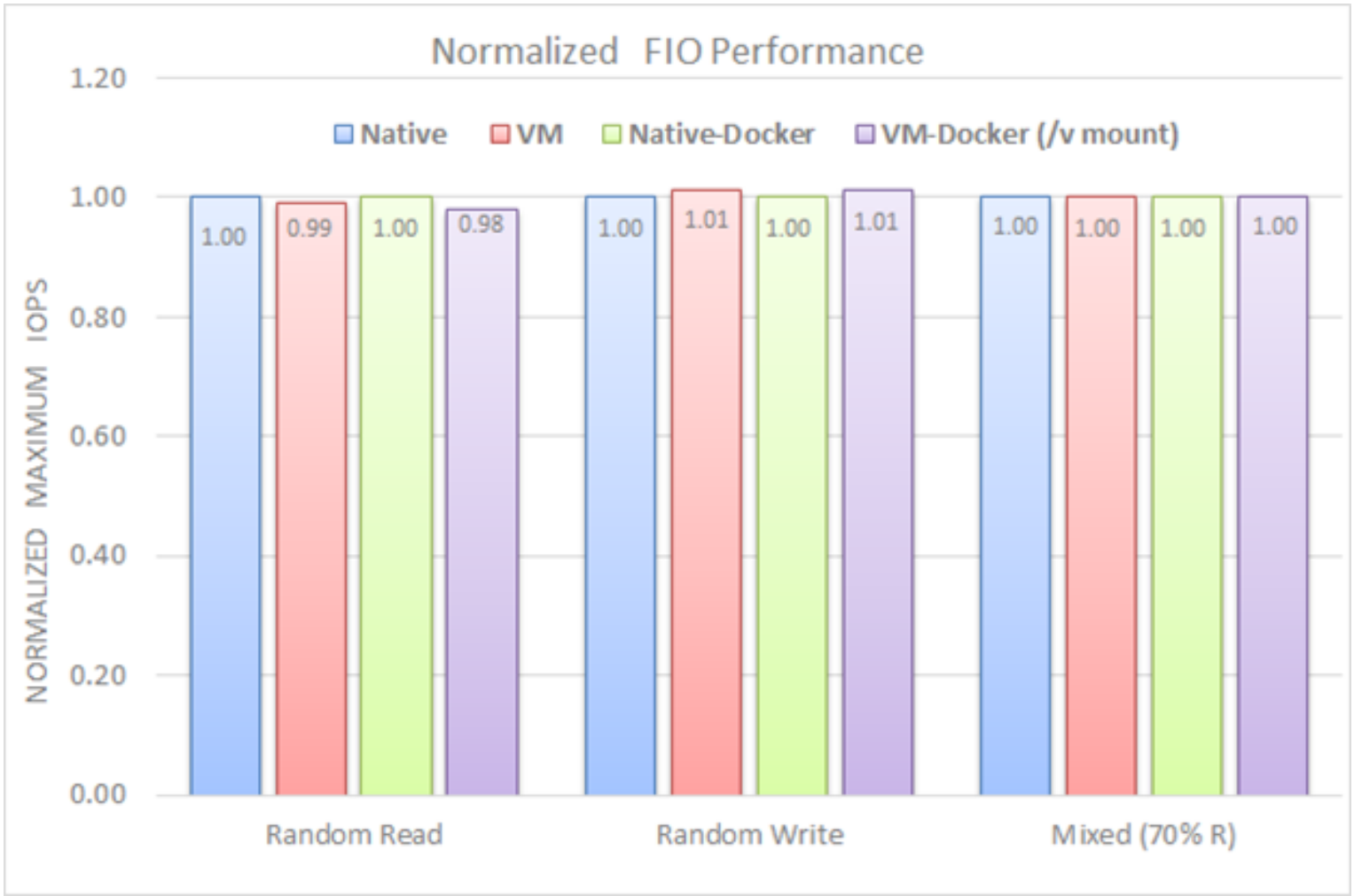
Docker performance – random I/O



IBM Research, *An Updated Performance Comparison of Virtual Machines and Linux Containers*, 2014-07-21,

`http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf`

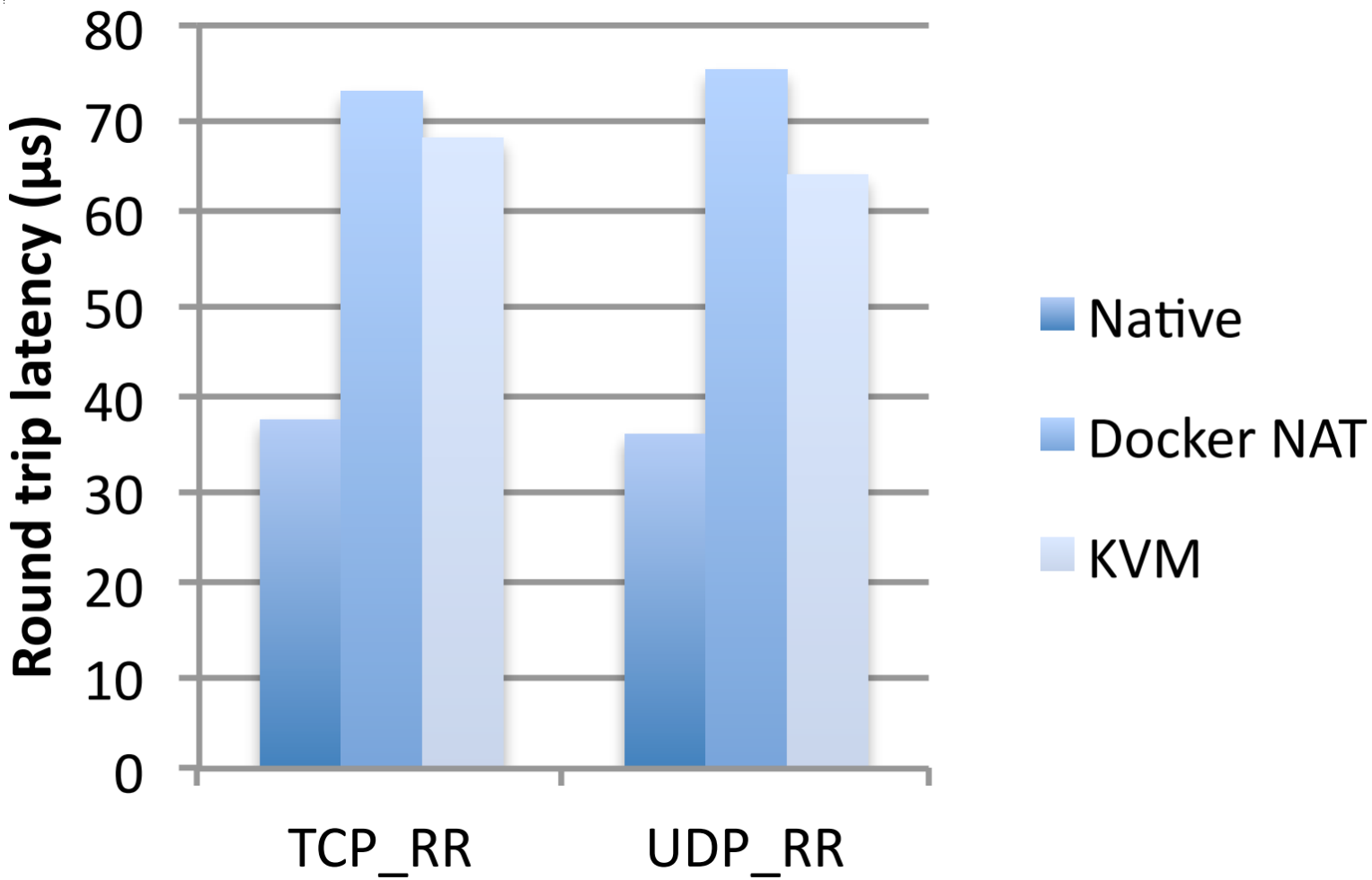
Docker performance – random I/O



VMware, *Docker Containers Performance in VMware vSphere*, 2014-10-15,

`http://blogs.vmware.com/performance/2014/10/docker-containers-performance-vmware-vsphere.html`

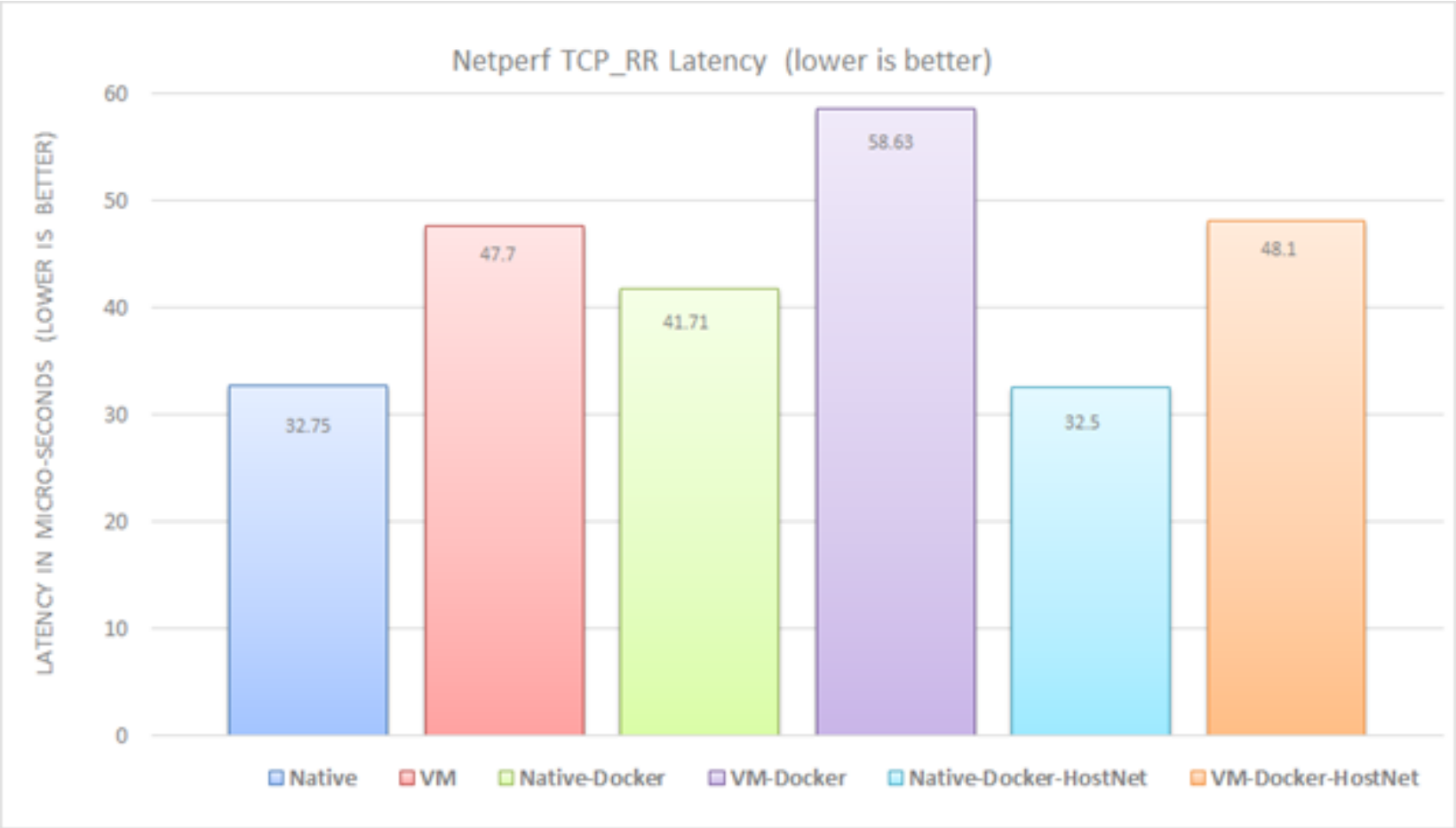
Docker performance – Network latency



IBM Research, *An Updated Performance Comparison of Virtual Machines and Linux Containers*, 2014-07-21,

`http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf`

Docker performance – Network latency

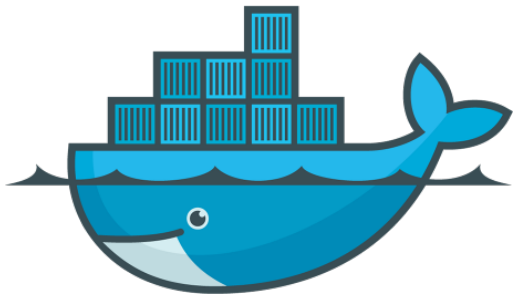


VMware, *Docker Containers Performance in VMware vSphere*, 2014-10-15,

`http://blogs.vmware.com/performance/2014/10/docker-containers-performance-vmware-vsphere.html`

Docker compared to Virtual machines

- Virtual machines have their own complete guest OS.
  - Separate kernel.
  - Userland.
  - Application we want to run.
  - Quickly adds up to much data.
  - Typically several processes.
- Docker
  - Shares kernel with host OS.
  - Runs as a process inside the host.
  - Only applications and its dependencies.
  - May consist of only a single process.



The good, the bad

Good

- Good-looking whale.
- Contains everything needed.
  - Easy to deploy to other hosts.
- Automatically assembles containers.
  - Docker Hub

Bad

- Linux only.
- Docker daemon runs as `root`. Potential security issue.