Performance of Docker vs VMs

Presented By
Ali Hussain

August 21, 2014

Flux7

Today's Presenter:

Ali Hussain

Co-founder & CTO Flux7

Prev: CPU Performance Analyst at Intel and ARM

Flux7: Cloud and DevOps Solutions

Cloud and Devops for Web teams

Enterprise DevOps management

AWS Certified Team

Partners:

Clients:

Other team members

Samprita Hegde

Performance Engineer at Flux7

Setup, execution, and collection of data

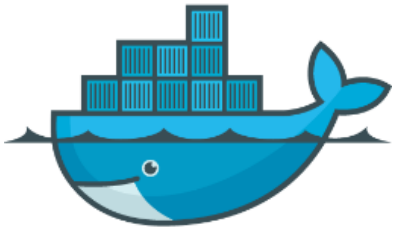
Check out our work on blog.flux7.com

Agenda

- Background
- Experimental results & deductions
- Application in the real world

What is Docker?

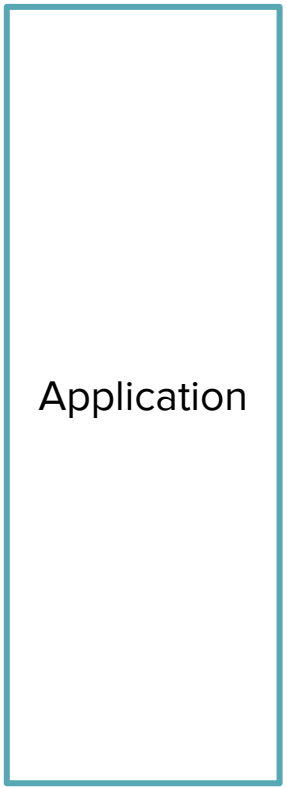
Linux OS isolation tools made easy



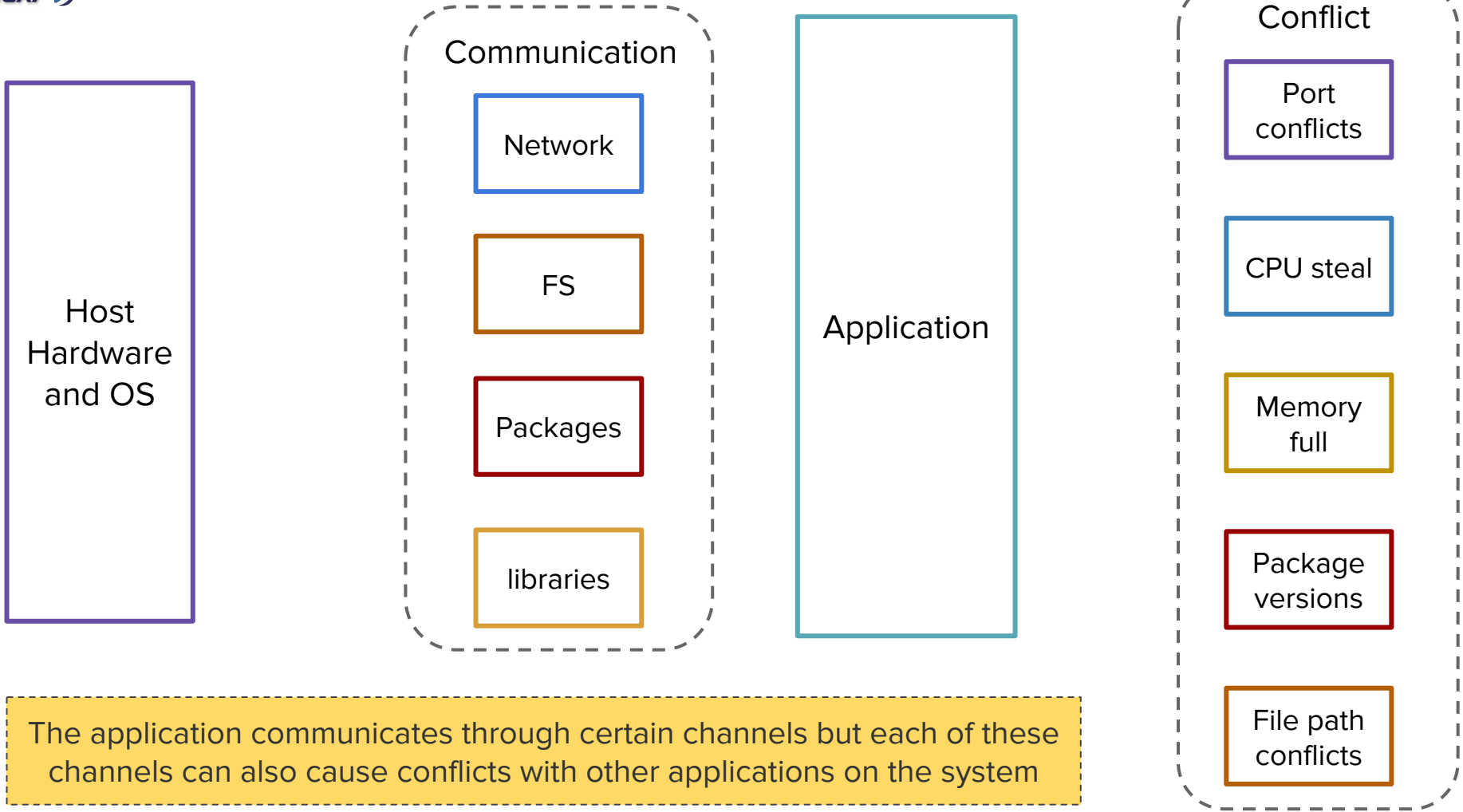
A Docker container looks like a virtual machine

Provide additional ‘goodies’ for app development

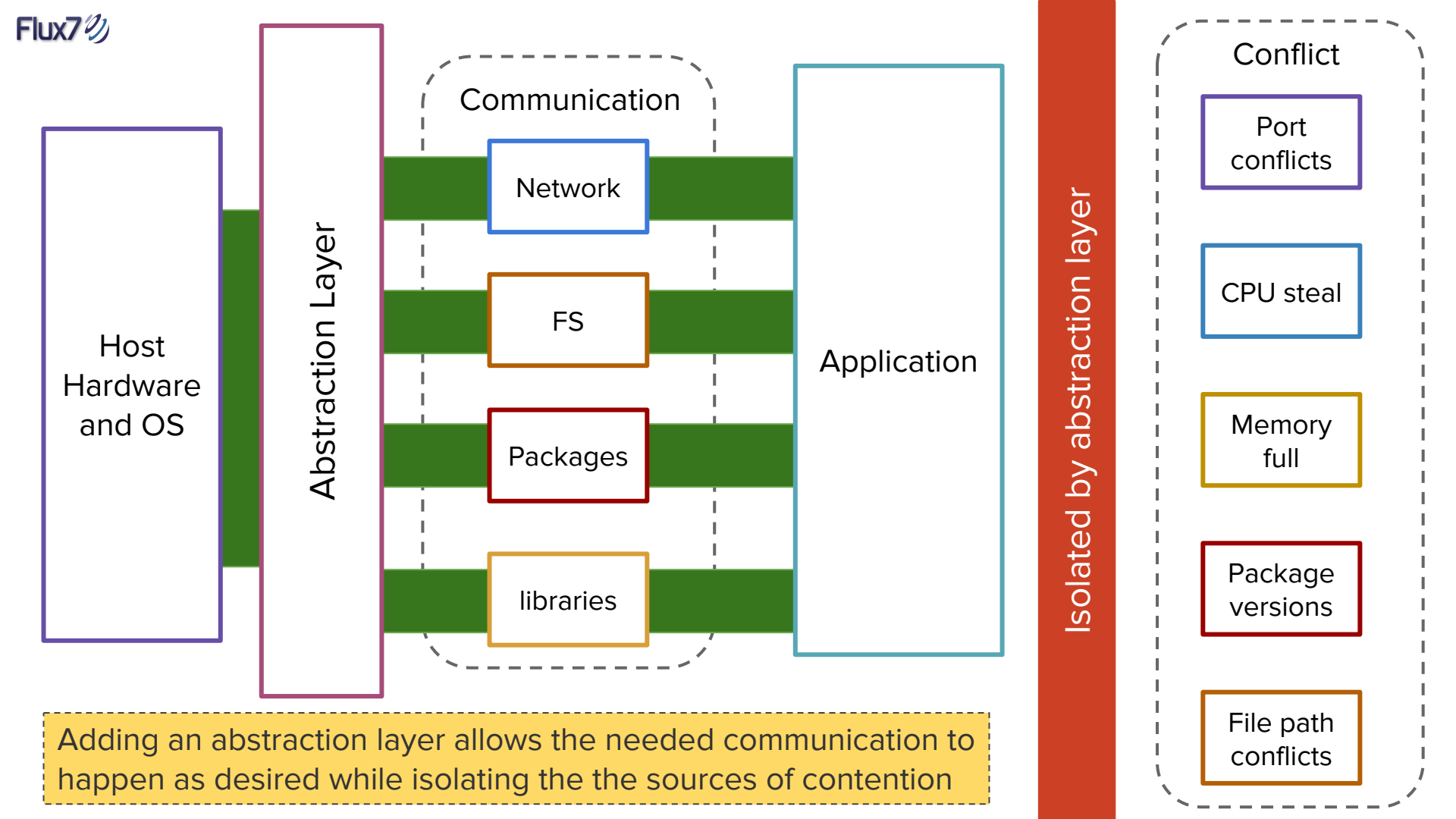
Holy Grail of “Virtualization”



An application needs to communicate with outside world: H/W, OS, other processes



The application communicates through certain channels but each of these channels can also cause conflicts with other applications on the system


Flux7
Host Hardware and OS
Abstraction Layer
Communication
Network
FS
Packages
libraries
Application
Isolated by abstraction layer
Conflict
Port conflicts
CPU steal
Memory full
Package versions
File path conflicts
Adding an abstraction layer allows the needed communication to happen as desired while isolating the the sources of contention

What we want

Allow communication

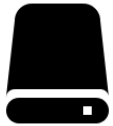
Prevent conflict over resources

A continuum of abstraction levels

How we pay for it

Resource overhead

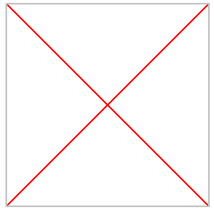
(Memory, CPU)



Increased latency

(Disk, Network)

Snapshot time and space



Start and stop time

Where is Docker on this spectrum?

Virtual Machines

Native

Docker Abstraction

- Use isolation features of the Linux kernel to give a VM-like environment
- Filesystem using chroot jails
- Network using port forwarding
- Resource and process isolation using cgroups

Test Platform

OS: Ubuntu 12.04 LTS , kernel - 3.8.0-33-generic

Docker version: 0.7.2, build 28b162e

Processor	Memory	Disk
Intel(R) Core (TM) i7 CPU	47 GB RAM 23 GB Swap	2 drive raid1 (mirroring) configuration

* Some tests run on AWS because of specific needs

Start and stop times

	Start Time	Stop Time
Docker Containers	<50ms	<50ms
VMs	30-45 seconds	5-10 seconds

Memory

No overhead of running a guest kernel

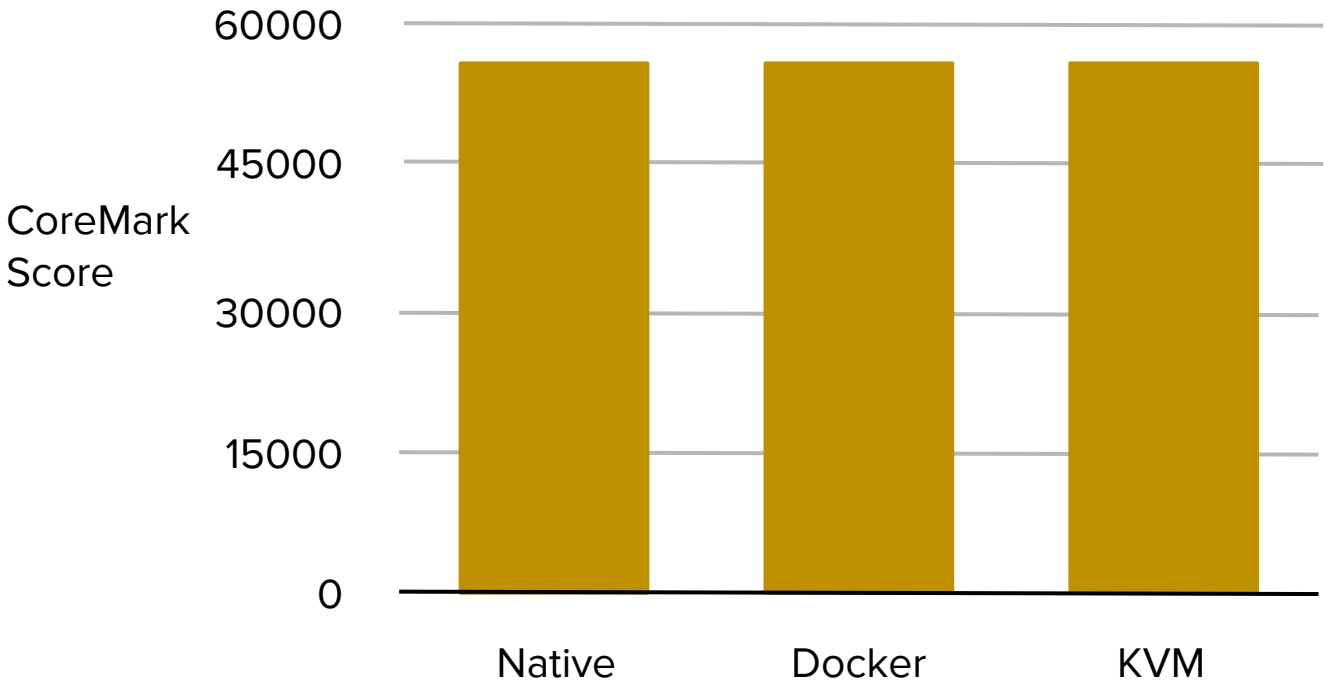
Docker requires consumed memory not provisioned memory

CPU Overhead - Methodology

Measured using coremark

Microbenchmark measuring CPU performance

CPU Overhead



CPU Overhead

No noticeable difference between native, KVM, and Docker performance

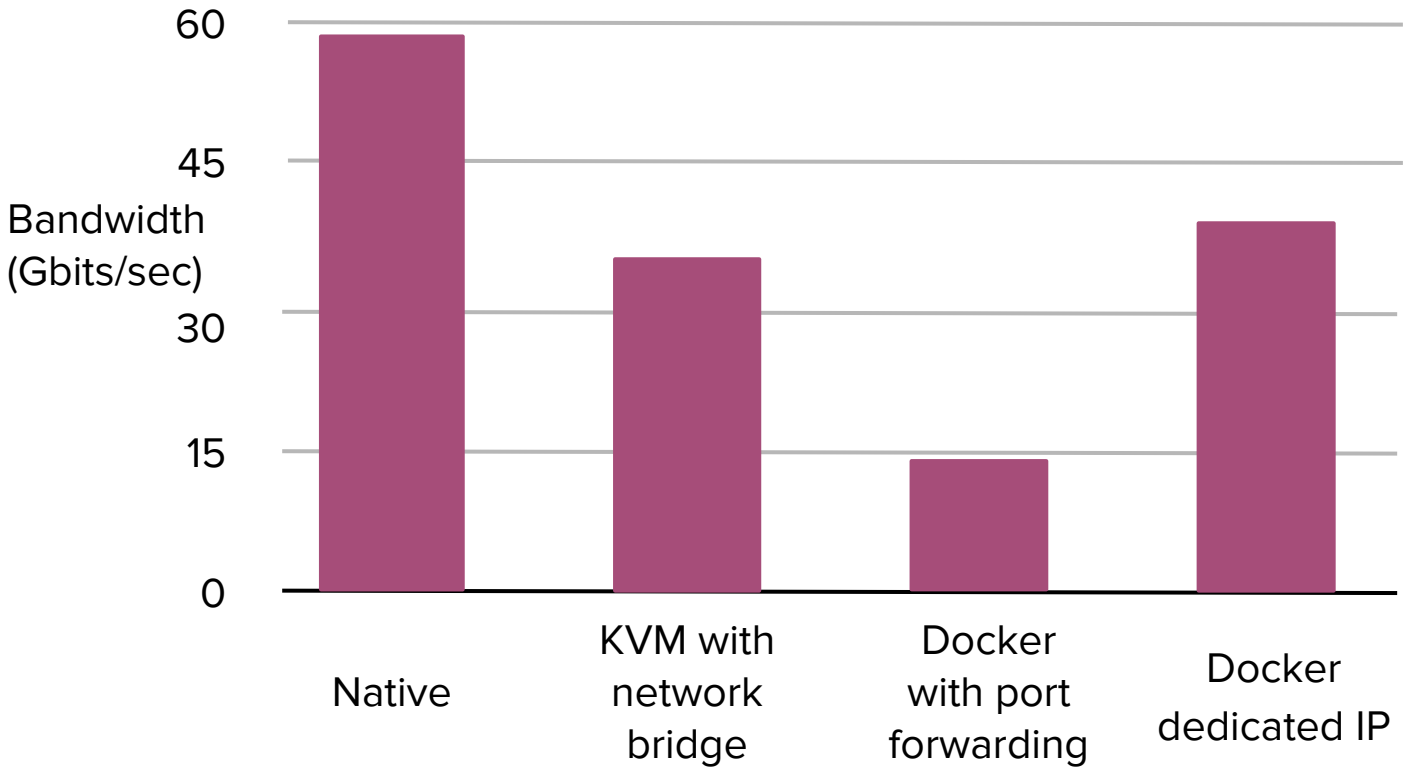
KVM uses ~1.5% more CPU usage compared to Docker when idle

Network Performance - Methodology

Performance measured using iperf

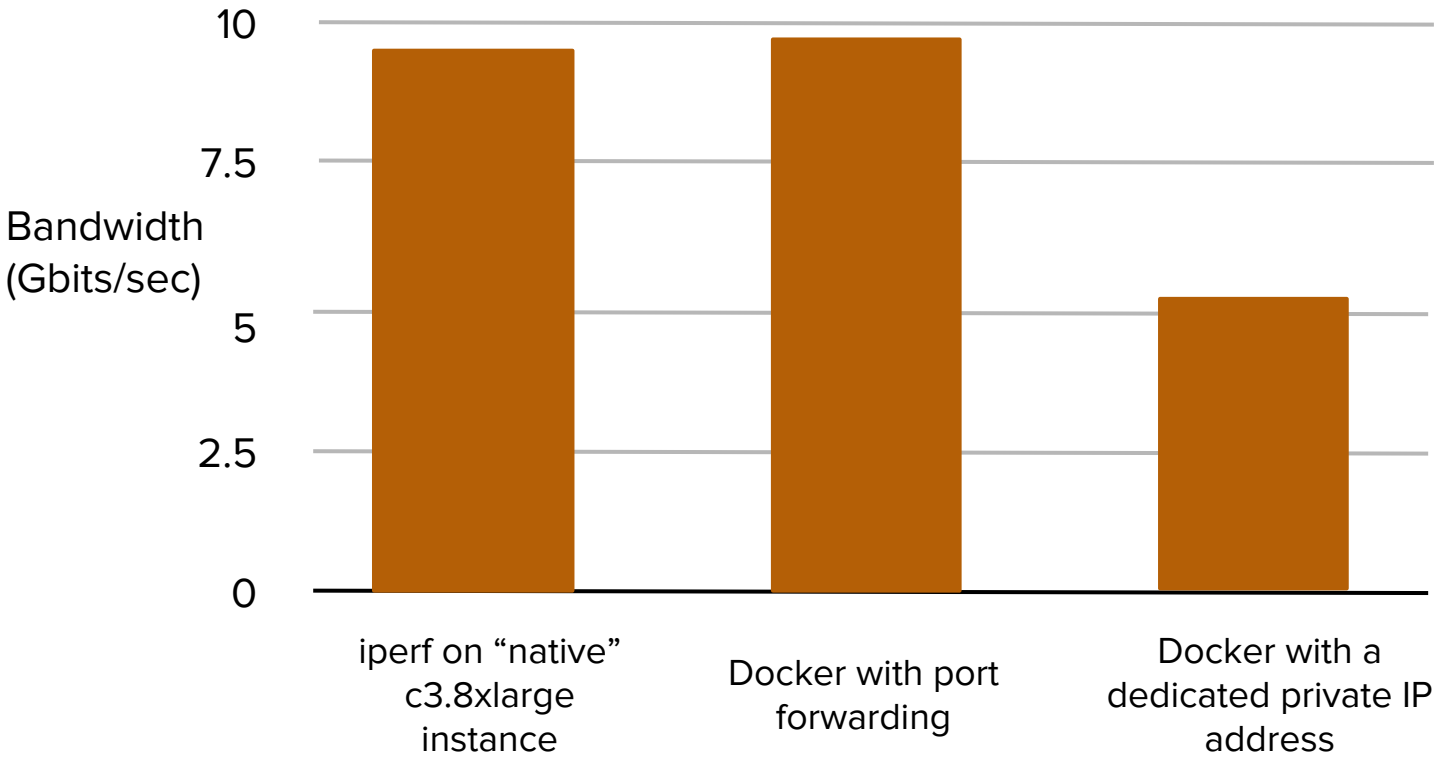
Care about both latency and throughput

Network Throughput



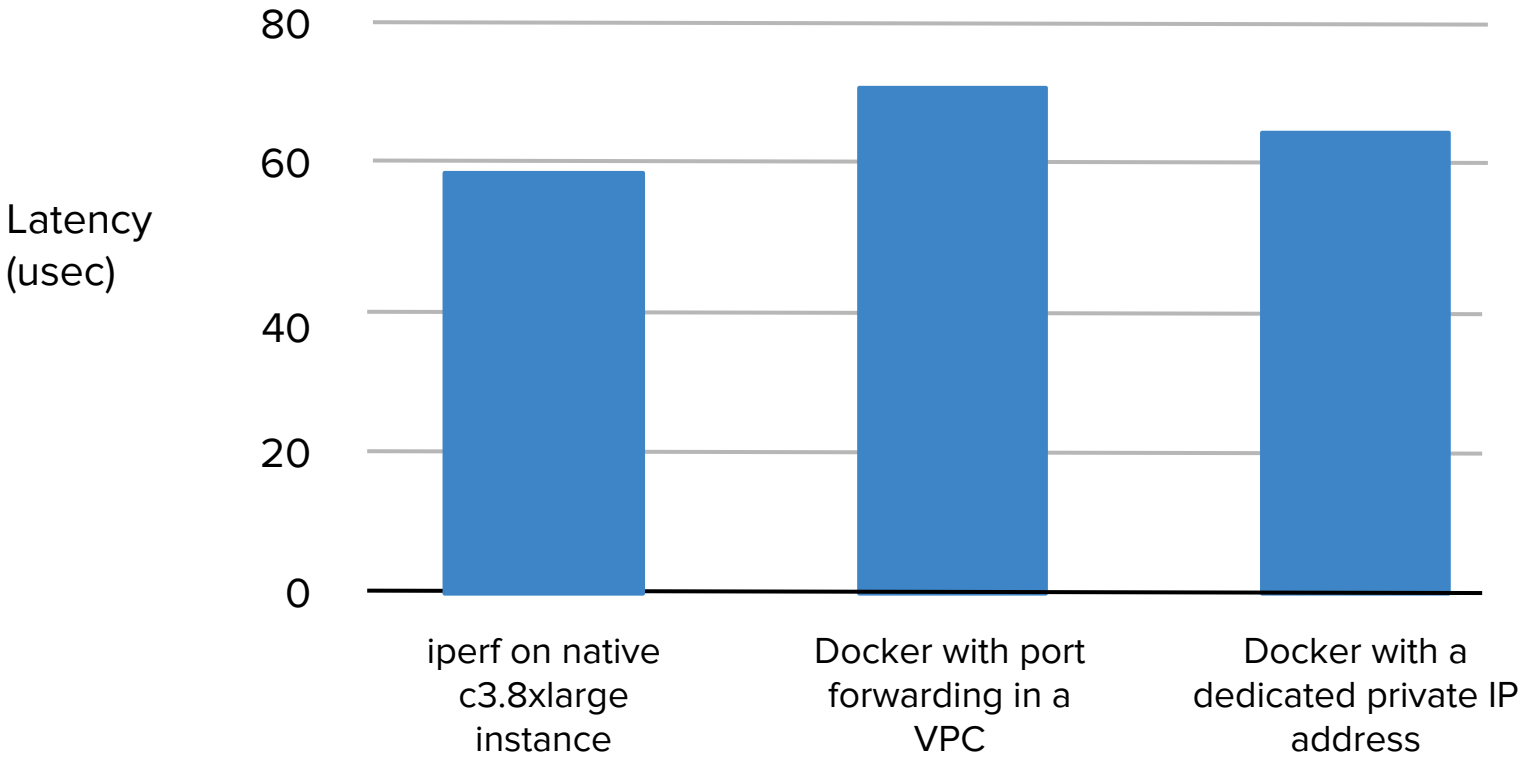
Theoretical Scenario:
Server and Client on same instance

Network Throughput



Practical Scenario: Server and Client on different instances

Network latency (lower is better)



Network Performance

Both Docker and KVM can saturate 10GbE

Docker running inside a VM can still saturate 10GbE

Docker redirection latency ~10us

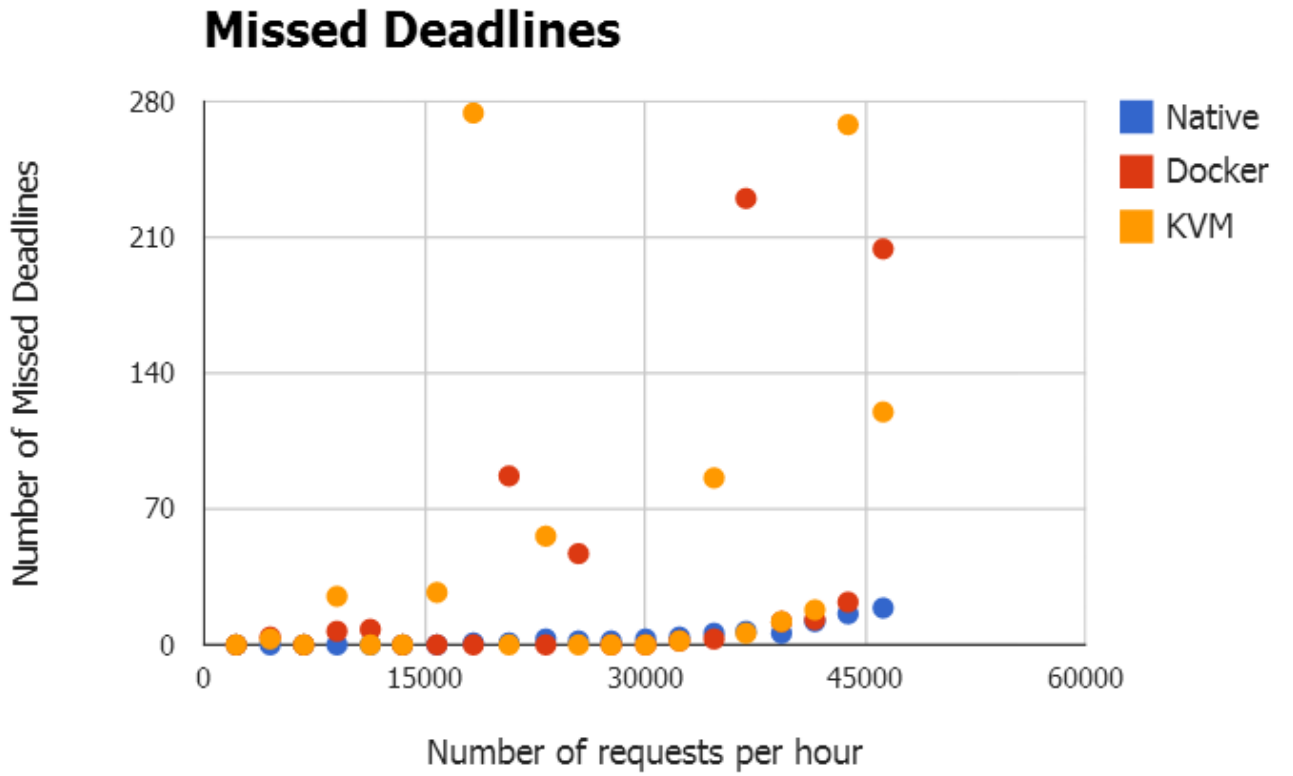
Disk Bandwidth

Measured using FIO - Stresses disk with different streams

A lot of variation but no clear winner

Different drivers causing differences with different scenarios

Application Benchmarking - Wikibench



Benchmark summary

Load performance:

- Native, KVM, and Docker performed comparably
- H/W and S/W advances in virtualization
- Over micro and macro benchmarks

Docker shines in

- ✓ Idle resource usage
- ✓ Start and stop times

The nature of disruptive changes

New technologies provide benefit only if you drop the rules that were designed to overcome the shortcomings of the old technology

-- Dr. Eliyahu Goldratt

Use Cases

Configuration Management

Multiple instances per box

Basic unit of provisioning resources

Configuration Management

Create configuration to work in your canonical environment and emulate it

We already do this

- Virtual machines
- Python virtualenvs

Configuration Management

Why Docker?

Consistent environment

Many dev niceties

No memory overhead

Minimize performance overhead

Configuration Management

What we get?

Running identical code + configs across environments

Better code pipeline management

Use Cases

Configuration Management

Multiple instances per box

Basic unit of provisioning resources

Multiple Instances per box

Multiple instances each running in own Docker container

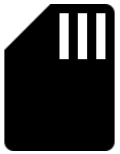
Containers may communicate over TCP

Multiple Instances per box

Why Docker?

Process and FS isolation

Port forwarding



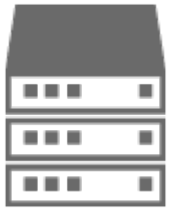
No idle memory and CPU overhead

Multiple Instances per box

What we get

High fidelity local dev environments

Multi-tenancy

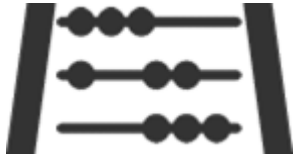


Server consolidation

Use Cases

Configuration Management

Multiple instances per box



Basic unit of provisioning resources

Basic Unit of Resource Provisioning

Allocate Docker containers instead of VMs

Can be both

Persistent resources

Very short tasks

Basic Unit of Resource Provisioning

Why Docker

Sharing of resources

Quick start and stop times

Management of multiple images

Basic Unit of Resource Provisioning

What we get

Faster provisioning

Lower overhead on host

Config management

Thank you, Docker!

For developing a very cool piece of technology

For making it OpenSource

Special thanks to **Jerome Petazzoni** for supporting us

Thank You!

Twitter: @Flux7Labs

Blog: blog.flux7.com