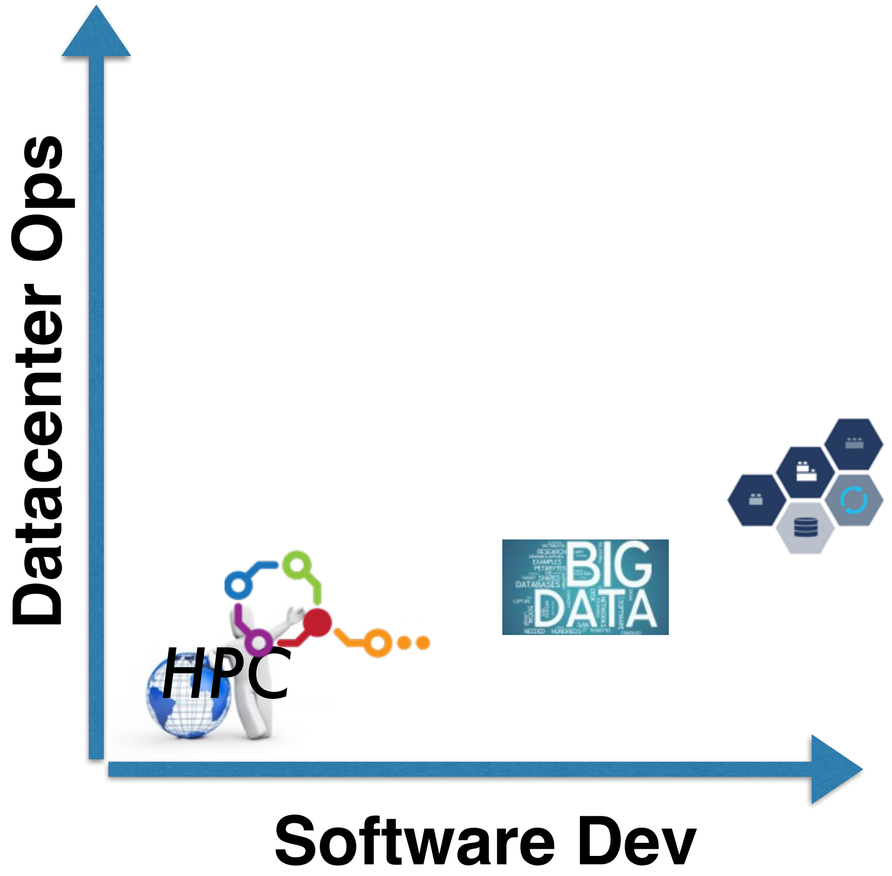
What drives Docker in Non-HPC?

ISC2016 - Linux Container Workshop

Docker Momentum

- IT Tinkering (Hello World)
- Continuous Dev/Int/Dep
- Microservices, hyper scale
- Big Data
- High Performance Computing



Disclaimer: **subjective exaggeration**

Docker in Software Development

Spinning up production-like environment is great

- MongoDB, PostreSQL, memcached as separate containers
- python2.7, python3.4

Like python's *virtualenv* on steroids,
iteration speedup through reproducibility

Docker in HPC development

Spinning up production-like environment is…

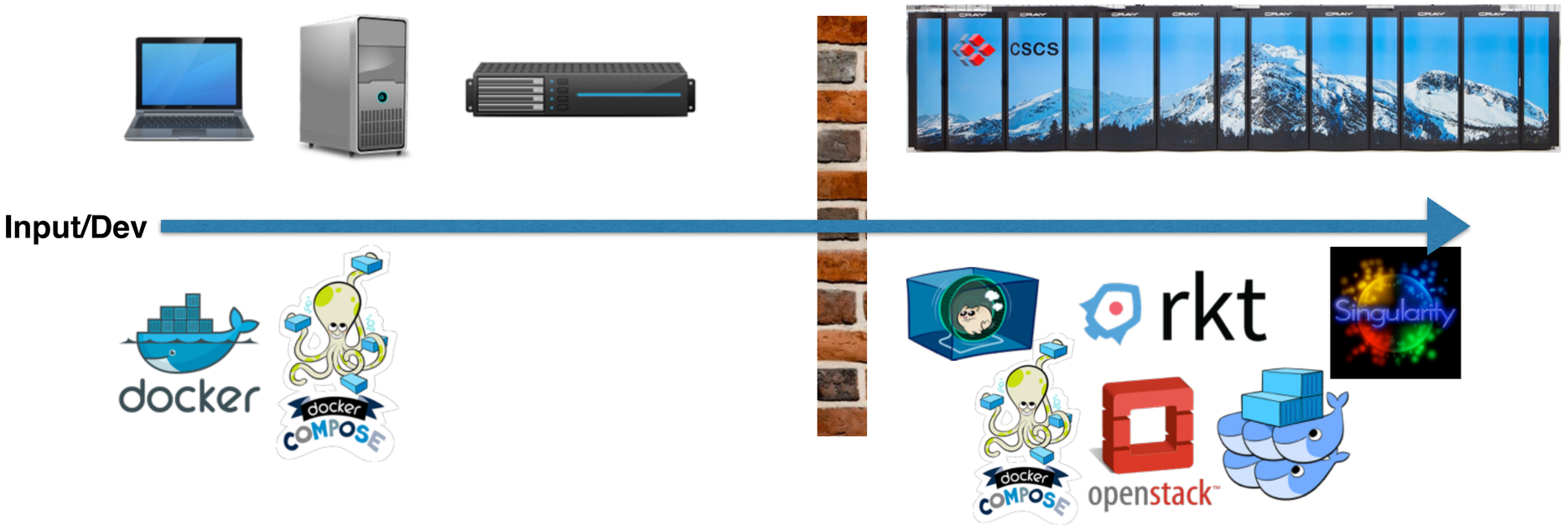
- …not that easy
- focus more on engineer/scientist, not the software-developer

1. For development it might work
   - close to non-HPC software dev
2. But is that the iteration-focus?
   - rather job settings / input data?

Separation of Concerns?

Split input iteration / development from operation

- non-distributed stays vanilla
- transition to HPC cluster using tech to foster operation



Orchestration

Non-HPC is about Services

… not Workloads

- BigData crunching (batch processing) is more in HPC camp
- where Stream-Processing (Spark, Storm, Kafka Stream) is service again

1. High Performance Big Data
   - Hardware vendor supply performance, but does the stack change?
2. Bio-science (other fast iterating disciplines)
   - repeatability by packaging via Container

Batch Systems

1. Workload Scheduler
   - single host SLURM jobs are low-hanging fruit
   - complex for distributed workloads (how to spawn remote tasks?)
2. Simple Queue-System
   - RabbitMQ, ZeroMQ, Redis
3. Data-Driven Computational Pipelines
   - E.g. nextflow

Service Orchestration

1. Docker SWARM
   - Simple to grasp
   - extends and builds on Docker API
2. Datacenter Orchestration
   - Derived from DC operational need of Google (Kubernetes)
   - …academic project from Berkeley. (Mesos)

Integration

The Social/Shrink Part

1. When disrupting a piece
   - everyone throws in his wish-list
   - describes the current state as desired state
   - some resist change
2. With Container in particular
   - VMs were easy to align with physical machine world
   - Containers break this model

The Social/Shrink Part #2

1. With AWS EC2…
   - IT departments waved with the CreditCard as a threat
   - it was easy to try something at scale
2. With Containers
   - no-one is going to tolerate complex setup procedures
   - ‘works on my laptop’ is going to be extinct
3. With Unikernels/AWS Lambda
   - ms spin-up redefines state-less… (but that’s for next time)

Downscaling

‘Bottom up’-Stack

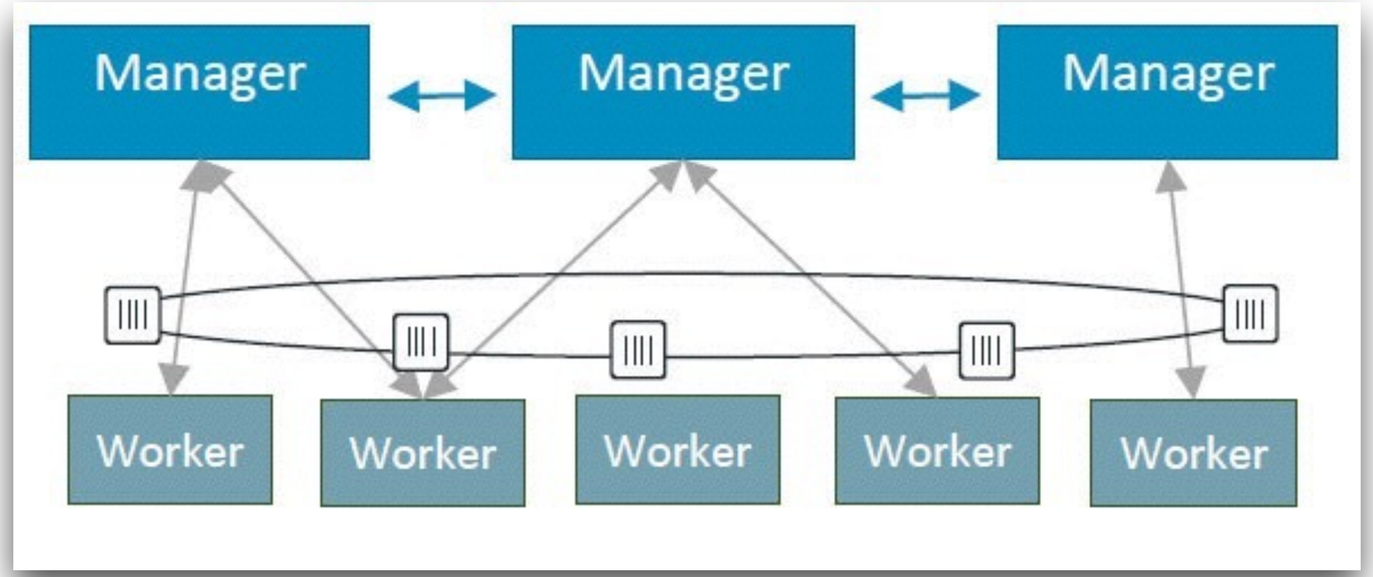
1. Work on ProjectX
   - Spin up (minimalistic) dependency stack
     - should reside within ProjectX
   - only change the code of ProjectX
2. Possible in HPC?
   - Are HPC stacks reducible to laptop-consumable size?

Docker 1.12

Docker-Engine 1.12 [SWARM]

SWARM now included

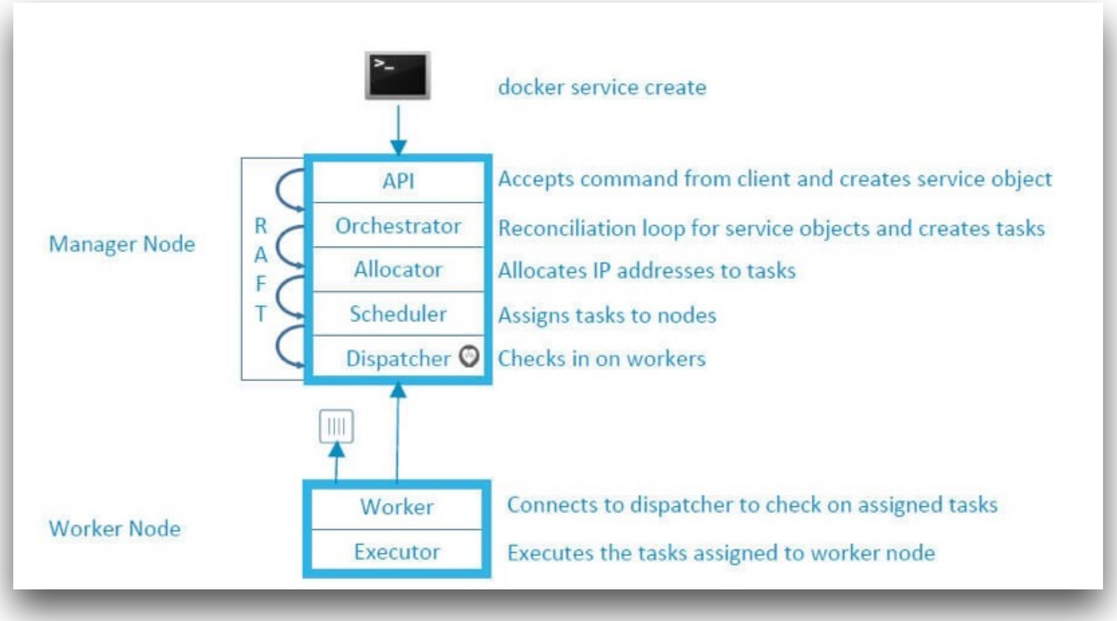
- docker-engines are either manager/worker
- they form a SWARM cluster
- no external Key/Value store needed anymore

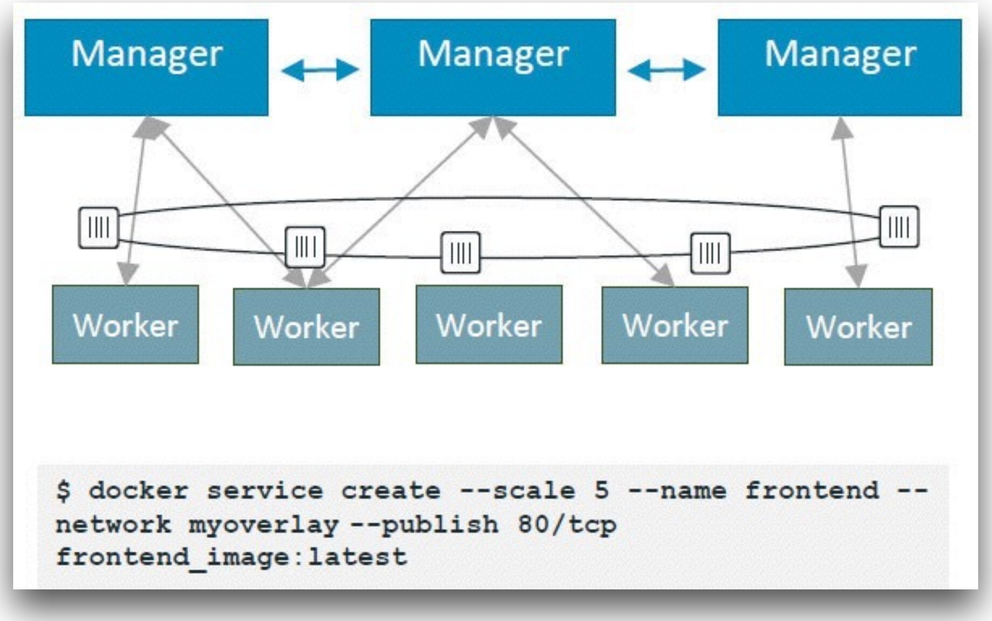


Docker-Engine 1.12 [Services]

docker service

- stacks are bundled as services
- health-checks (!), self-healing, rolling-update, canary deployment

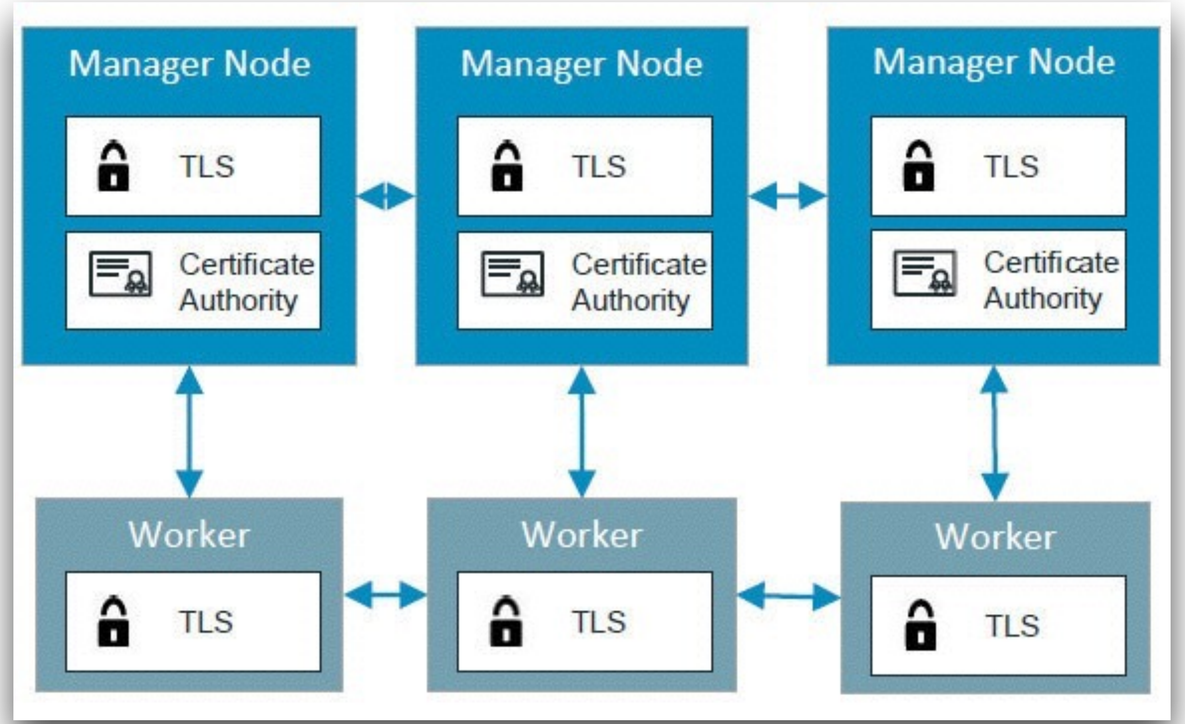




Docker-Engine 1.12 [TLS]

build-in (but swappable) TLS

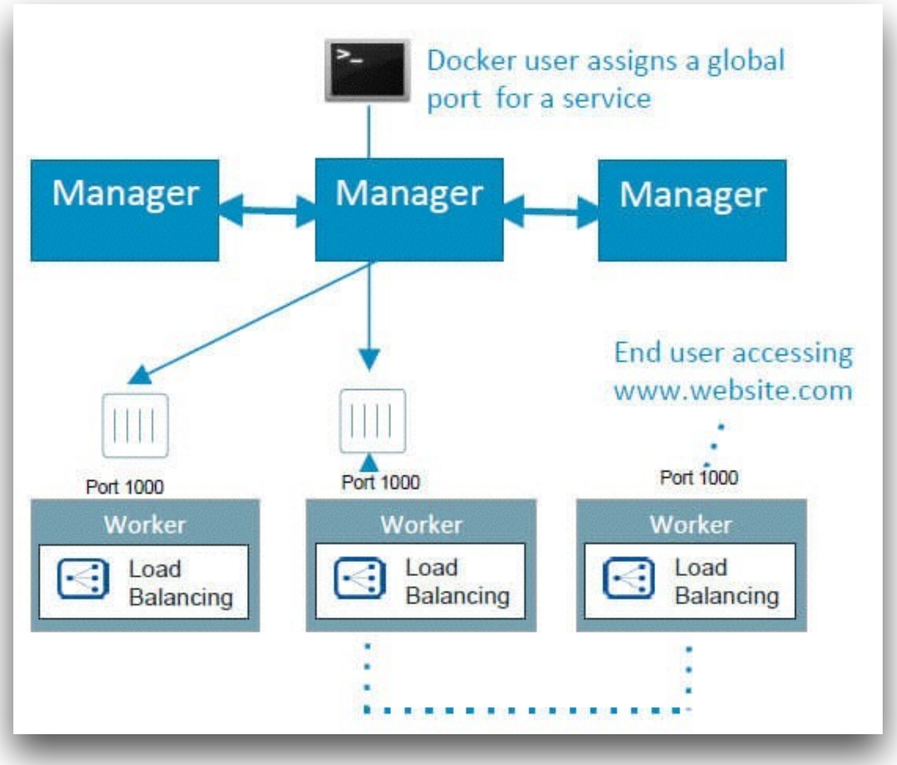
- all traffic encrypted



Docker-Engine 1.12 [MeshNet]

Build in Load-Balancer

- services are load-balanced by default

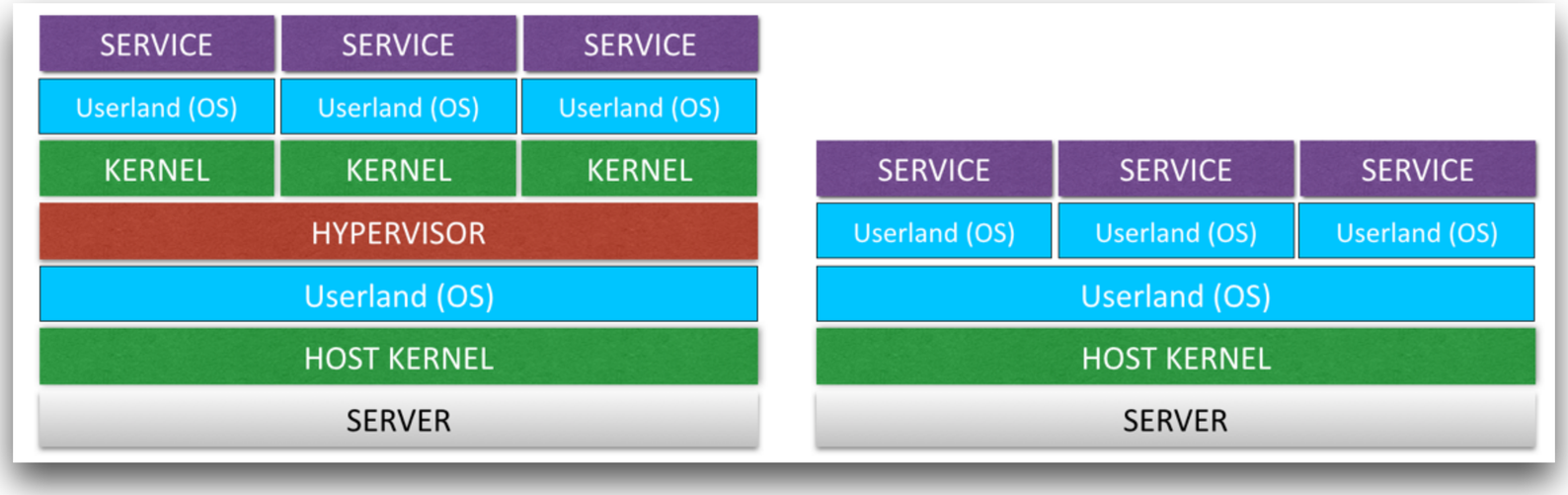


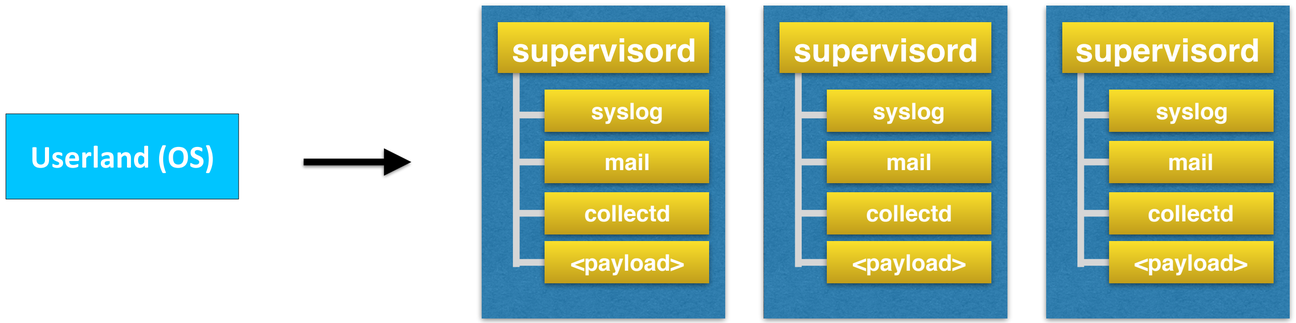
Docker-Engine 1.12 [HPC]

1. You guys don’t scream loud enough
   - Feature-wise they target us cloud guys. :)
2. OK, not true…
   - HP-Big Data could employ the service model for their workloads
   - it’s more service deployment then distributed workload-deployment (MPI)

Chunk Systems Up

Reminder VMs / System Containers

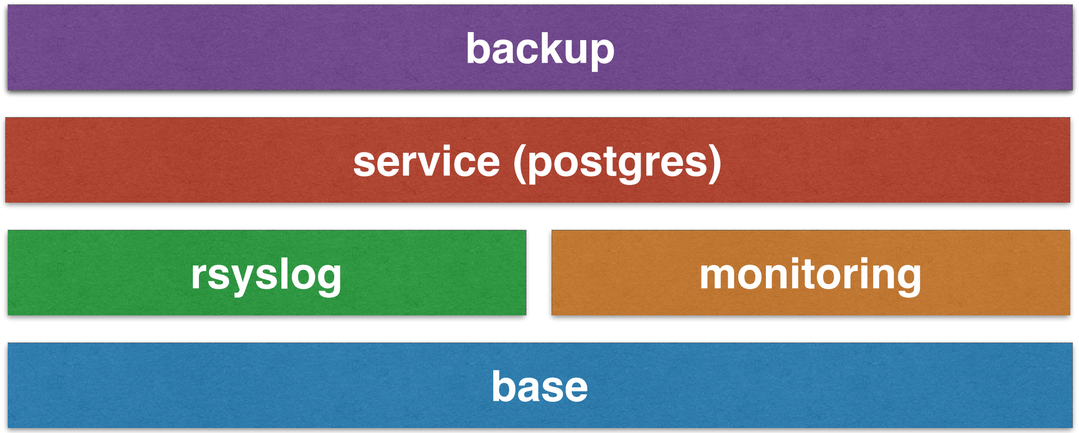




One Layer at a time

Ideally the container runs one process

- ENTRYPOINT might be something to set up the inner working
- no init-system needed



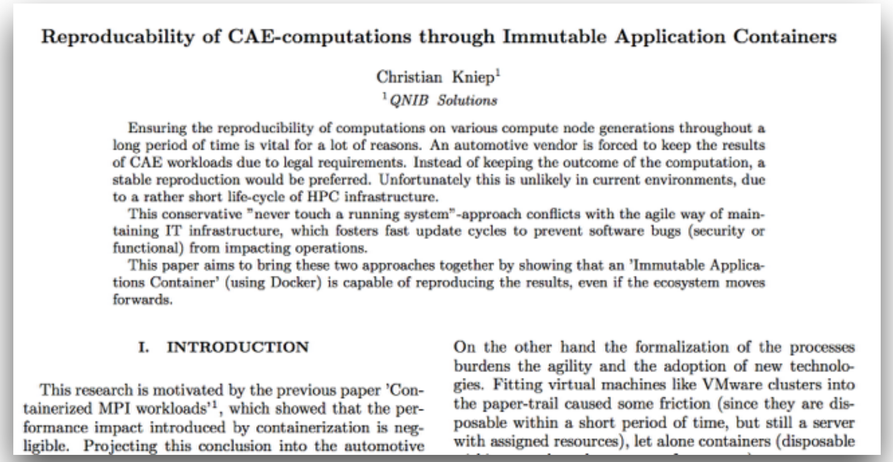
User-Land Optimisation

Reproducibility / Downscaling

Running OpenFOAM on small scale is cumbersome

- manually install OpenFOAM on a workstation
- be confident that the installation works correctly

☑ A containerised OpenFOAM installation tackles both



Reproducability of CAE-computations through Immutable Application Containers

Christian Kniep[1]
[1]*QNIB Solutions*

Ensuring the reproducibility of computations on various compute node generations throughout a long period of time is vital for a lot of reasons. An automotive vendor is forced to keep the results of CAE workloads due to legal requirements. Instead of keeping the outcome of the computation, a stable reproduction would be preferred. Unfortunately this is unlikely in current environments, due to a rather short life-cycle of HPC infrastructure.

This conservative "never touch a running system"-approach conflicts with the agile way of maintaining IT infrastructure, which fosters fast update cycles to prevent software bugs (security or functional) from impacting operations.

This paper aims to bring these two approaches together by showing that an 'Immutable Applications Container' (using Docker) is capable of reproducing the results, even if the ecosystem moves forwards.

I. INTRODUCTION

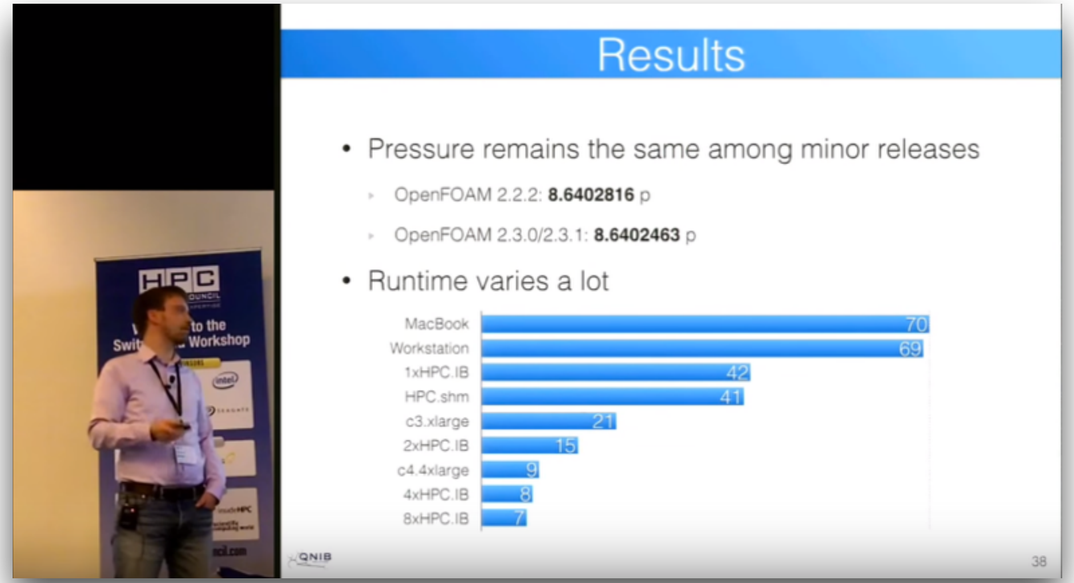
This research is motivated by the previous paper 'Containerized MPI workloads'[1], which showed that the performance impact introduced by containerization is negligible. Projecting this conclusion into the automotive

On the other hand the formalization of the processes burdens the agility and the adoption of new technologies. Fitting virtual machines like VMware clusters into the paper-trail caused some friction (since they are disposable within a short period of time, but still a server with assigned resources), let alone containers (disposable

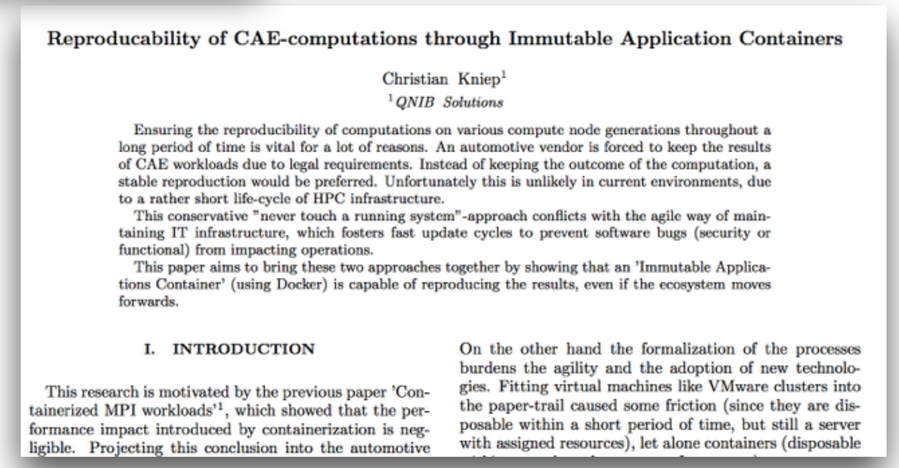
http://qnib.org/immutable-paper

YouTube http://qnib.org/immutable

OpenFOAM Benchmark



YouTube http://qnib.org/immutable



Reproducability of CAE-computations through Immutable Application Containers

Christian Kniep[1]

[1] *QNIB Solutions*

Ensuring the reproducibility of computations on various compute node generations throughout a long period of time is vital for a lot of reasons. An automotive vendor is forced to keep the results of CAE workloads due to legal requirements. Instead of keeping the outcome of the computation, a stable reproduction would be preferred. Unfortunately this is unlikely in current environments, due to a rather short life-cycle of HPC infrastructure.

This conservative "never touch a running system"-approach conflicts with the agile way of maintaining IT infrastructure, which fosters fast update cycles to prevent software bugs (security or functional) from impacting operations.

This paper aims to bring these two approaches together by showing that an 'Immutable Applications Container' (using Docker) is capable of reproducing the results, even if the ecosystem moves forwards.

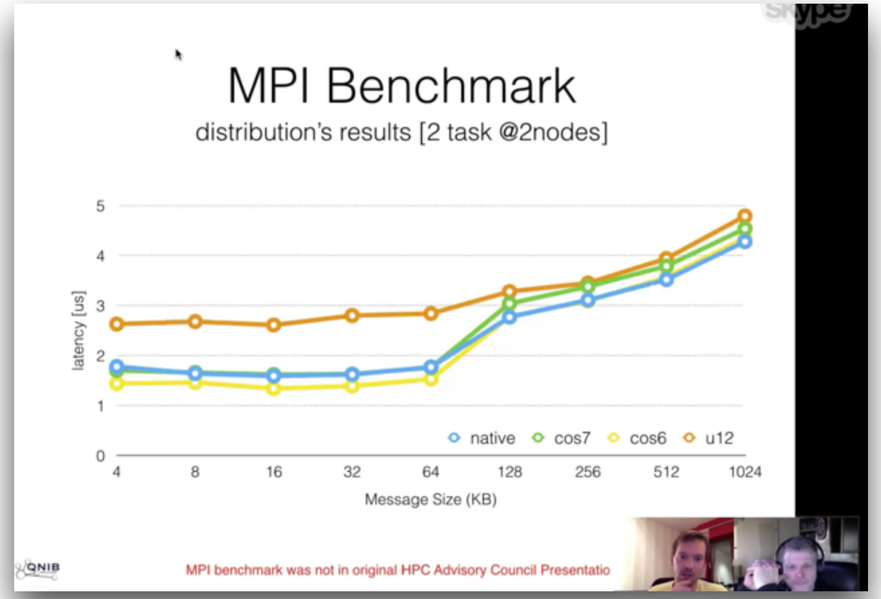
I. INTRODUCTION

This research is motivated by the previous paper 'Containerized MPI workloads'[1], which showed that the performance impact introduced by containerization is negligible. Projecting this conclusion into the automotive

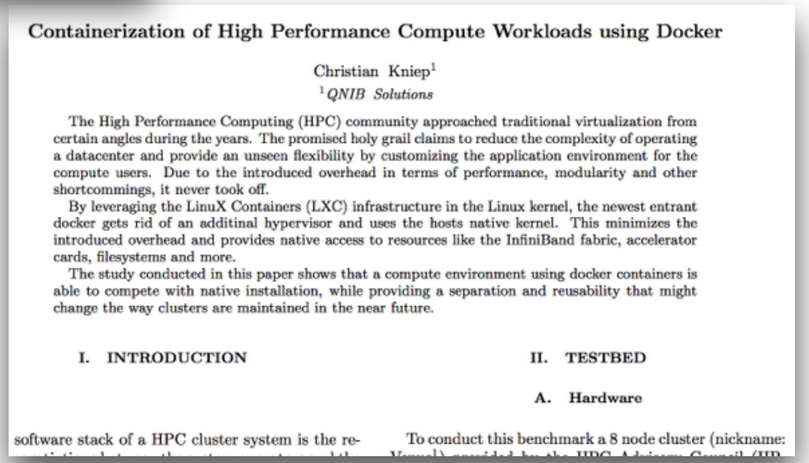
On the other hand the formalization of the processes burdens the agility and the adoption of new technologies. Fitting virtual machines like VMware clusters into the paper-trail caused some friction (since they are disposable within a short period of time, but still a server with assigned resources), let alone containers (disposable

http://qnib.org/immutable-paper

MPI Benchmark



YouTube http://qnib.org/mpi



Containerization of High Performance Compute Workloads using Docker

Christian Kniep[1]
[1]*QNIB Solutions*

The High Performance Computing (HPC) community approached traditional virtualization from certain angles during the years. The promised holy grail claims to reduce the complexity of operating a datacenter and provide an unseen flexibility by customizing the application environment for the compute users. Due to the introduced overhead in terms of performance, modularity and other shortcommings, it never took off.

By leveraging the LinuX Containers (LXC) infrastructure in the Linux kernel, the newest entrant docker gets rid of an additinal hypervisor and uses the hosts native kernel. This minimizes the introduced overhead and provides native access to resources like the InfiniBand fabric, accelerator cards, filesystems and more.

The study conducted in this paper shows that a compute environment using docker containers is able to compete with native installation, while providing a separation and reusability that might change the way clusters are maintained in the near future.

I. INTRODUCTION

software stack of a HPC cluster system is the re-

II. TESTBED

A. Hardware

To conduct this benchmark a 8 node cluster (nickname:

http://qnib.org/mpi-paper