Technical Fundamentals of Cloud Systems (2)

Cloud Computing

Universiteit Leiden

Towards other OS models

- Virtual machine instances are typically set up to run a single application.
  - "Single-purpose appliance".
  - Especially in large-scale applications; these simple instances are then scaled up when necessary.
- However, Linux is a fully featured, general-purpose, multi-user, multi-process operating system.
  - Does it make sense to boot up a fully featured operating system to run a single application?
  - Do we need a local file system, multi-user support, authentication?
  - Do we need dual-mode operation? A (guest) kernel crash is handled by the hypervisor and the host OS will not crash.

Towards other OS models (2)

- Development of “unikernels”. Purpose: run a single application.
  - Single process, single address space and only paravirtualized drivers. Designed to run under a hypervisor.
- An example is OSv
  - Originally developed to run a JVM with a Java application.
    - (JVM already deals with memory protection).
  - Later extended to run single Linux C applications.

A. Kivity et al. 2014. OSv—Optimizing the Operating System for Virtual Machines. USENIX 2014.

Towards other OS models (3)

- Library Operating Systems: back from the past.
- The OS kernel is a library and is linked against an application to form a single executable.
  - (single-address space)
- Example: MirageOS
  - Application is written in a higher-level programming language (protects against memory errors). Compiled down into high-performance bare-metal executable.

A. Madhavapeddy and D. Scott. 2014. Unikernels: the rise of the virtual library operating system. Commun. ACM 57, 1 (January 2014) 61-69.

Containers

- Why boot a full OS for a single application?
- Containers take a different approach: virtualize only the environment offered by the OS, instead of virtualizing a full system.
- OS environment consists of e.g.:
  - Root file system
  - A process tree
  - Network connection

Containers (2)

- Make a single OS kernel offer multiple of such environments, instead of just one.
  - These environments are isolated from one another.
  - Processes are “trapped” in such a container. Other containers are not visible.
  - Container has its own root file system with software installation.

- Easy software distribution. Fast boot times.
  - (Think about upscaling!!)

Containers (3)

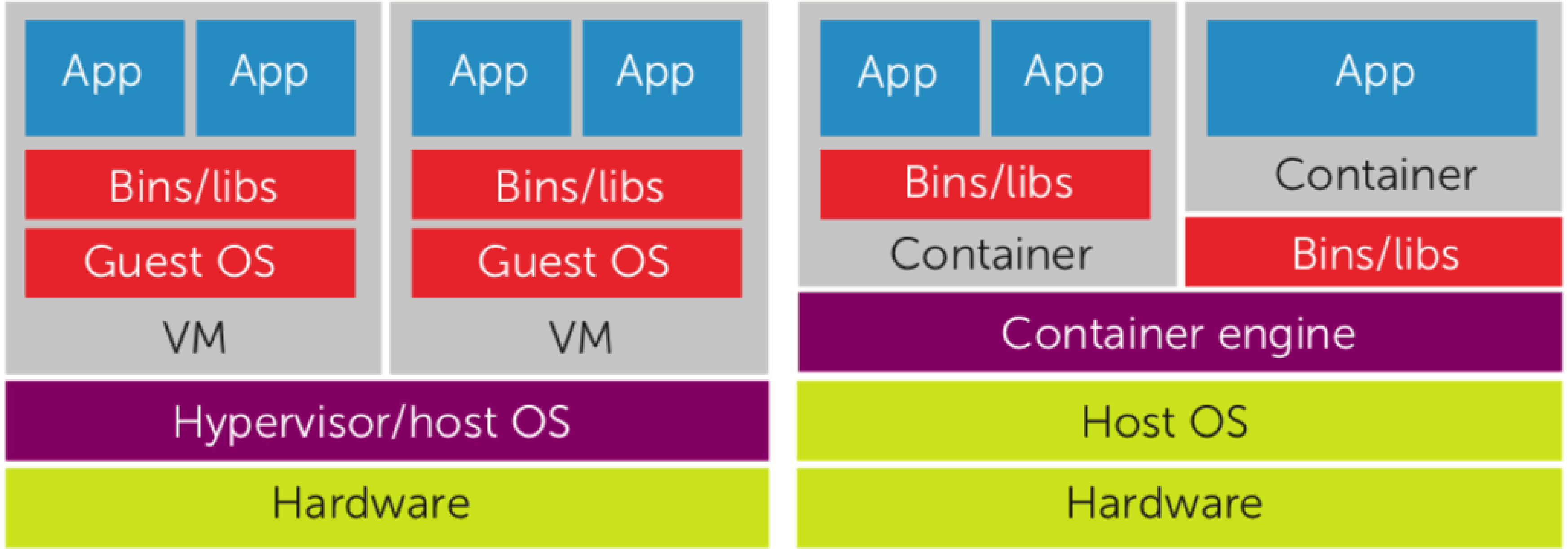
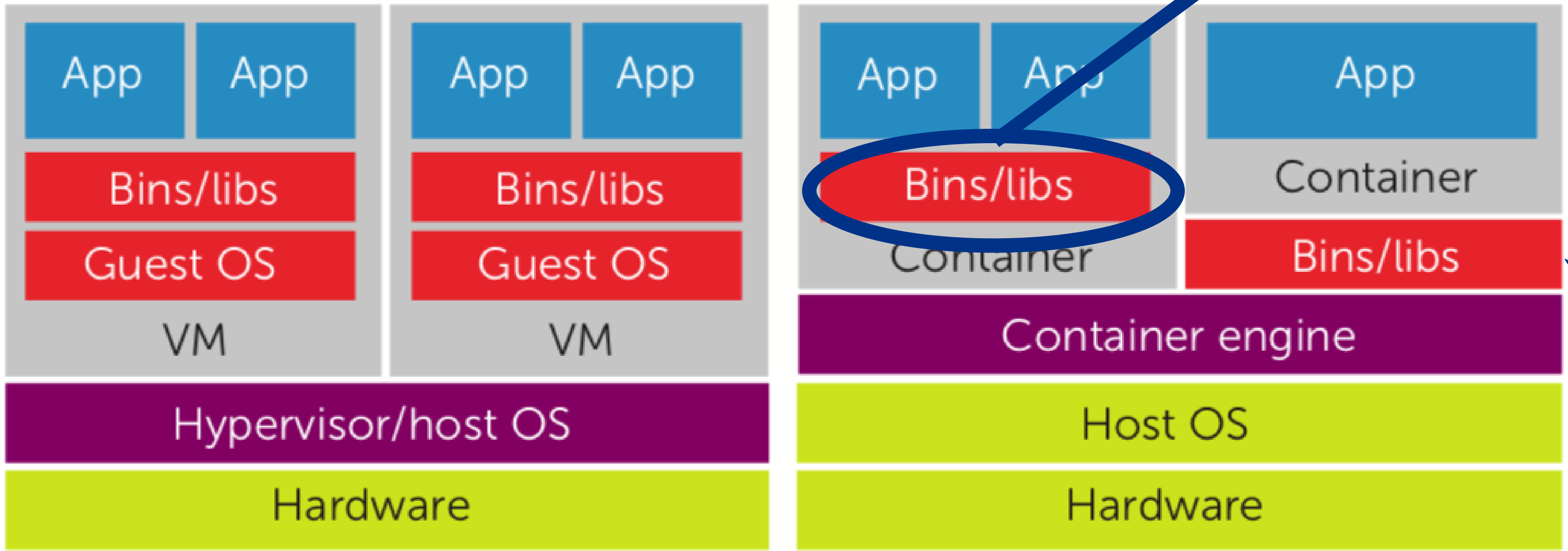


FIGURE 1. Virtualization architecture. The two possible scenarios, a traditional hypervisor architecture on the left and a container-based architecture on the right, differ in their management of guest operating system components.

Image source: C. Pahl. 2015. Containerization and the PaaS Cloud. In: IEEE Cloud Computing May/June 2015. IEEE Computer Society.

Containers (3)

Paper means to say that this is included “if necessary”.



Seems to be a little arbitrary…

FIGURE 1. Virtualization architecture. The two possible scenarios, a traditional hypervisor architecture on the left and a container-based architecture on the right, differ in their management of guest operating system components.

Image source: C. Pahl. 2015. Containerization and the PaaS Cloud. In: IEEE Cloud Computing May/June 2015. IEEE Computer Society.

Containers (4)

Differences with hypervisors (whole-system virtualization):

- Containers do not run a separate kernel. Therefore, all container processes run in user mode and don't execute privileged instructions.
  - No need for special handling of privileged instructions or page tables. Significantly less overhead.
- Containers are smaller than fully fledged OS VM images.
- Faster boot times.
- Isolation not as stringent: if the kernel is compromised from a container, the other containers can potentially be accessed.

Containers (5)

To implement containers on Linux, a number of specific kernel features are required:

- chroot/pivot_root: change the root file system.
  - Essentially restrict the container to a subtree of the file system tree.
- CGroups: manage allocation of system resources such as CPU time, memory and network bandwidth.

(continues on next slide)

Containers (6)

- seccomp: security feature to limit the system calls that can be done from the container to the kernel.
- Kernel namespaces: isolate processes running in a container from anything outside of the container.
  - PID namespace: we can have a PID 1 within multiple containers (and the host). PID remapping is used.
  - mount namespace: hide outside mounts from container processes. Each container can have own /tmp partition.
  - network namespace: same for network devices.

Containers (7)

- Fortunately, not everybody has to deal with these low-level kernel features. Low-level container runtimes abstract these away.

- Examples are:
  - LXC (https://linuxcontainers.org/)
  - runC
  - crun
  - kt

- Goal: to create an environment as close as possible to a standard Linux installation without the need for a separate kernel.

Hypervisors vs. containers

- Performance differences between hypervisors and containers have been researched in the literature.
- Generally, no real differences in CPU and memory performance.
  - Near native performance is achieved.

R. Morabito et al. 2015. Hypervisors vs. Lightweight Virtualization: a Performance Comparison. In: 2015 IEEE International Conference on Cloud Engineering. IEEE Computer Society.

W. Felter et al. 2015. An Updated Performance Comparison of Virtual Machines and Linux Containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE.

Hypervisors vs. containers (2)

- However, disk I/O incurs an overhead.

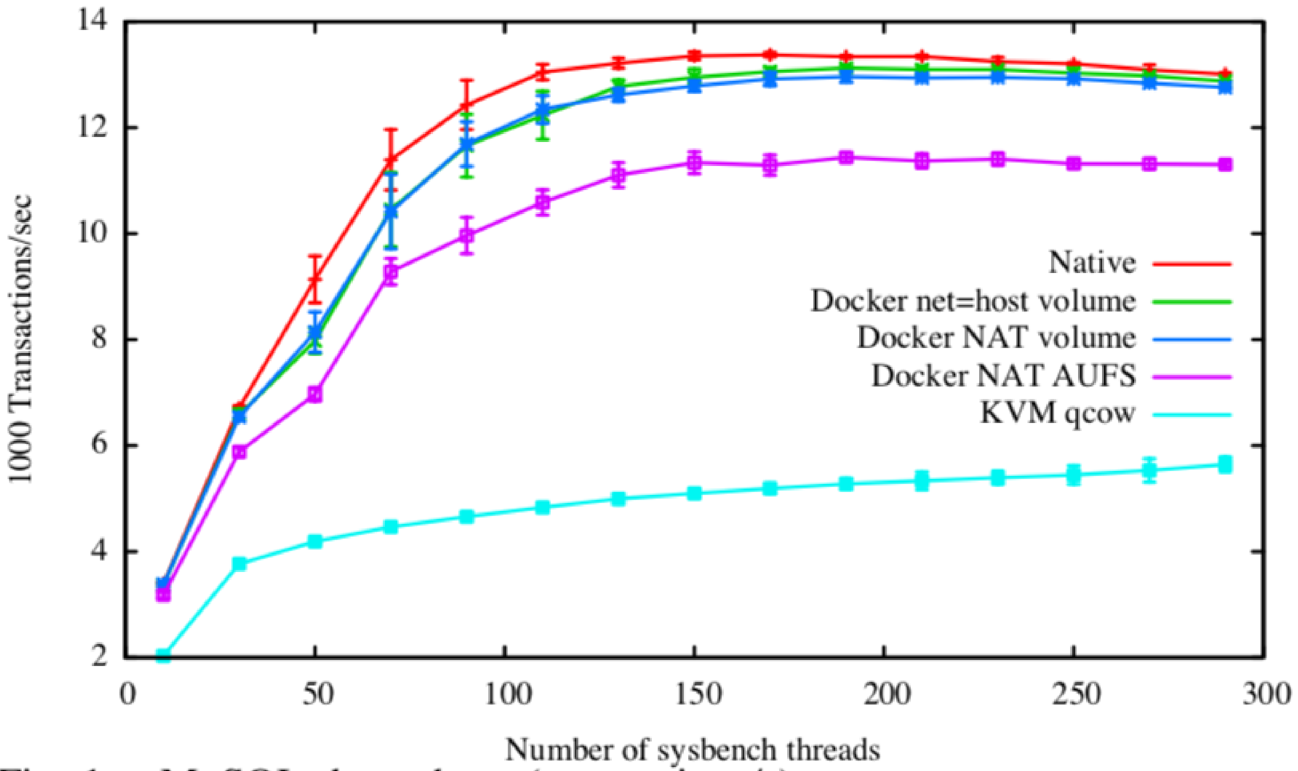
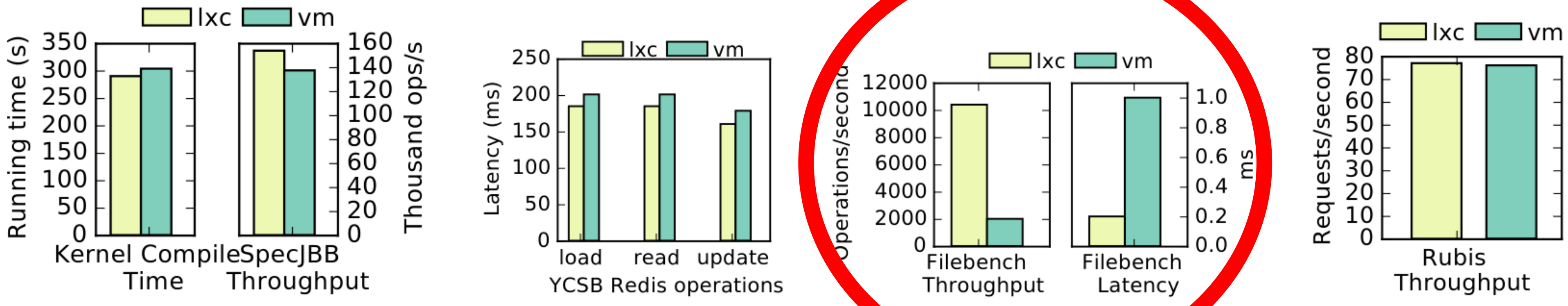


Fig. 1. MySQL throughput (transactions/s) vs. concurrency.

Image source: W. Felter et al. 2015. An Updated Performance Comparison of Virtual Machines and Linux Containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE.

Hypervisors vs. containers (3)

- Another perspective:



(a) CPU intensive (b) Memory intensive (c) Disk intensive (d) Network intensive

Figure 4: Performance overhead of KVM is negligible for our CPU and memory intensive workloads, but high in case of I/O intensive applications. Unlike CPU and memory operations, I/O operations go through the hypervisor—contributing to their high overhead.

Image source: Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and Y. C. Tay. 2016. Containers and Virtual Machines at Scale: A Comparative Study. In Proceedings of the 17th International Middleware Conference (Middleware '16). Association for Computing Machinery, New York, NY, USA, Article 1, 1–13.

- This paper also discusses CPU interference on containers and VMs.

Container Runtime vs. Engine

- runC, LXC (or similar) give you the ability to create a container and start a process inside.
  - Low-level Container Runtime
- But we need more than that:
  - Set up necessary networking,
  - expose ports,
  - mount data volumes within the containers,
  - tools to create, publish, maintain container images.
  - Sometimes referred to as Container Engine. The naming is a little convoluted at the moment.

Docker

- This is where, for instance, Docker comes in (https://docker.io).
- What is it all about?
  - Employs a layered file system to save space when storing images and to save time when creating new container images.
  - Mechanisms to create, download and distribute container images.
- Essentially, a tool to manage containers **locally**, and to support development and creation of containers.
- Container runtime details factored out into containerd and the previously mentioned runC.
- Prior to runC, Docker used LXC.

Container engine/runtime landscape

- Docker uses containerd, which in turn uses runc.
  - Docker is considered a container engine.
  - containerd a high-level container runtime.
  - runc a low-level container runtime.
    - Can run processes in containerized environments, but not much more. Images, networking, etc. need to be handled by a higher layer.
- Kubernetes uses CRI-O, which in turn uses runc.
  - Kubernetes talks to CRI-O using the CRI API: Container Runtime Interface.
- podman uses crun or runc.
  - podman is a drop-in replacement for Docker, and supports building containers, handling images and networking.
  - One could say it is situated in between Docker and containerd/CRI-O.

Standardization efforts

- Work has been done on standardization under OCI: Open Container Initiative.
- There is the OCI image specification, an open specification for container images. Many projects support this now, making images interchangeable between engines/runtimes.
- The OCI runtime specification defines how to run a certain process within a container image.
- In fact, from an OCI image combined with an OCI runtime specification a container can be instantiated.

OS considerations for containers

- Within containers:
  - Special “minimal” images of main Linux distributions (Ubuntu Minimal, CentOS).
  - Or special new distributions for this purpose: Alpine Linux.
- As container host:
  - The usual Linux distributions running dockerd / containerd/ cri-o.
  - Or special distribution developments:
    - CoreOS / Container Linux; automatically updating operating system, solely purposed for hosting containers.
    - Fedora / CentOS Atomic Host.
    - RedHat CoreOS

Active development

- Container hosting is an area of active development.
- Sometimes hard to get your head around, due to various projects being started (and being abandoned).
- Another example of recent developments:
  - Kata Containers: containers, but virtualized for better isolation.
  - Amazon FireCracker: provides microVMs to be used like containers.
  - Both can be considered as low-level container runtimes.

Data Storage and Networking

- So far, we have discussed virtualized execution environments.
  - Virtual machines: Creating efficient and isolated duplicates of real machines.
  - Containers: Isolated execution environments for applications.
- Virtual Machines also need data storage and networking.
  - A disk and network interface need to be attached.
  - These need to be virtualized as well to complete decoupling from physical hardware.

Data Storage

- Different things to store:
  - Virtual machine disks (block storage)
    - Snapshots of such disks, for e.g. backup purposes.
  - Photos uploaded to the cloud by a user (application data)
  - Virtual machine templates (machine images)
  - Containers
  - Data of a cloud-enabled relational database system (app. data)
  - …

Data Storage (2)

- We need to distinguish between:
  - Storage to support machine virtualization (so in fact IaaS)
  - Storage that would be exposed by various cloud APIs (photos, database records)
- We focus on the former.
- However, the distinction is sometimes a little fuzzy.

IaaS Data Storage

What do we need to store?

- VM disk images of active/suspended virtual machines
  - Representations of block devices
- Container images
- VM image templates

Billing: typically GB/month, higher price for SSD, IOPS free.

IaaS Data Storage (2)

How can we store this?

- **Image files:** we can store all contents of a block device within a single (large) file.
  - File is stored on the file system, either local or remote (e.g. NFS).
- **Block storage**: contents of a block device are stored within a remote block device.
- **Object:** store image file as object

IaaS Data Storage (3)

Where do we store this?

- Two choices:
  - On VM host nodes
  - Off VM host nodes (and thus on a dedicated storage system)
- (Dis)advantages can be named for both.
  - On host node: potentially better performance
  - Off host node: better decoupling; faster/easier migration and scaling.

IaaS Data Storage (4)

OpenStack lists three options:

- Off compute node – shared file system
  - Fully dedicated and separate storage system
- On compute node – shared file system
  - Compute nodes are used within the shared storage system
- On compute node – local file system
  - Potential data loss on compute node failure
  - Complicates live migration

https://docs.openstack.org/arch-design/design-compute/design-compute-storage.html

Basic IaaS Architecture Overview

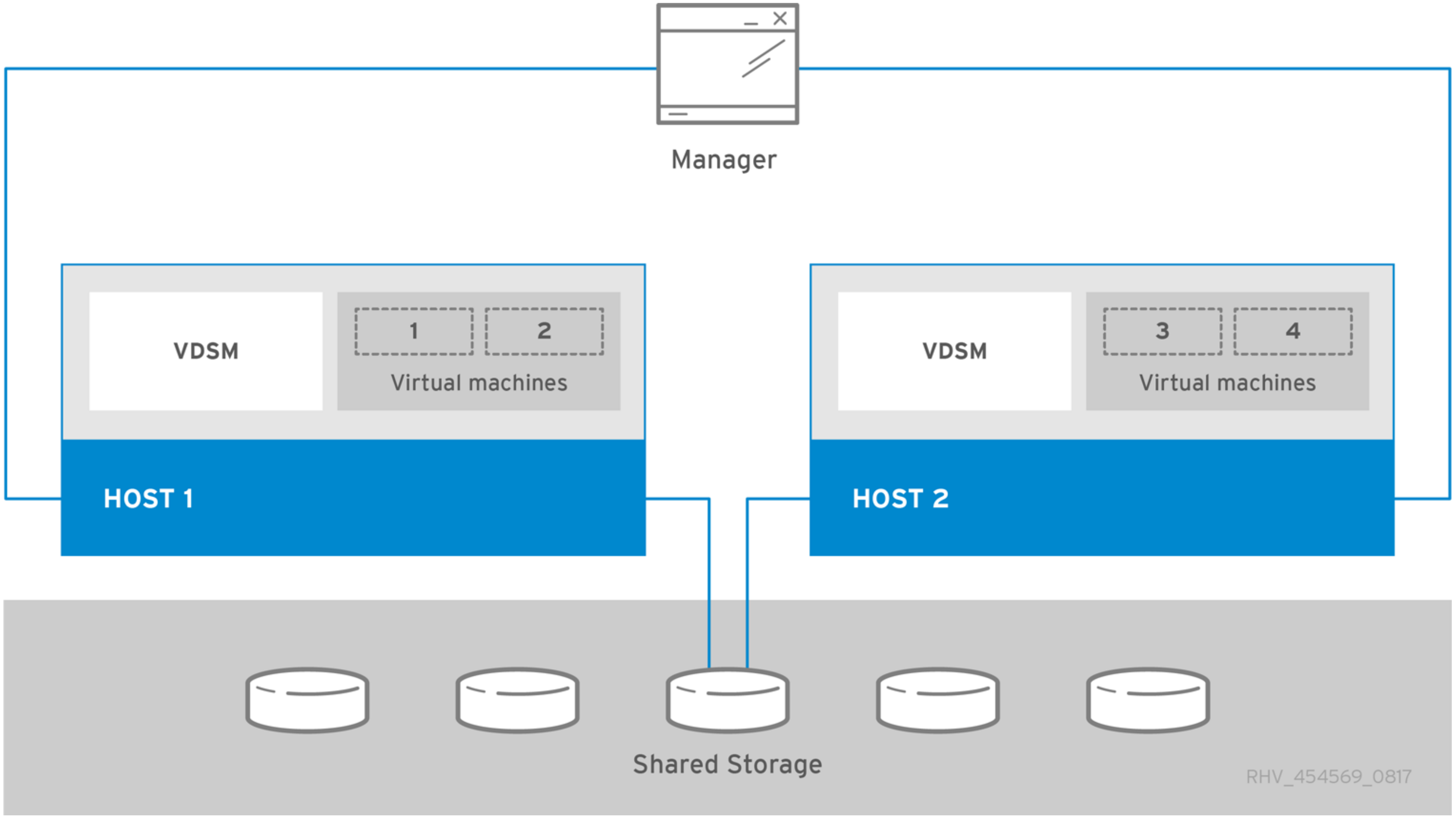


Image source (CC-BY-SA 3.0): https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html/product_guide/introduction

Stateless vs. stateful

- Is local storage ever useful?
- Consider stateless vs. stateful instances.
  - Stateless instances do not store any state locally. This is all stored elsewhere (database, object store, etc.)
  - So on crash, you simply restart elsewhere. No data loss.
  - Also, don't bother about live migration, just restart elsewhere.
  - Many containers are designed to be stateless.

Stateless vs. stateful (2)

- Stateful instances:
  - (Classical) relational database server
  - Virtual Private Server (VPS)
  - Server accounting website sessions, if session data is not stored on a shared medium.
- Can also have a stateless instance writing state to a shared file system
  - Often done with containers: the container instance is stateless, but writes data to a shared file system mounted in the container.

IaaS Data Storage Implementation

- NFS: shared file-based storage
  - Hard to scale to (very) large clusters
- SAN exposing iSCSI: shared block-based storage
  - SAN: Storage Area Network
  - FibreChannel: very high-end, but very expensive; enterprise networks only.
  - Could also be built from commodity components and open source software.

IaaS Data Storage Implementation (2)

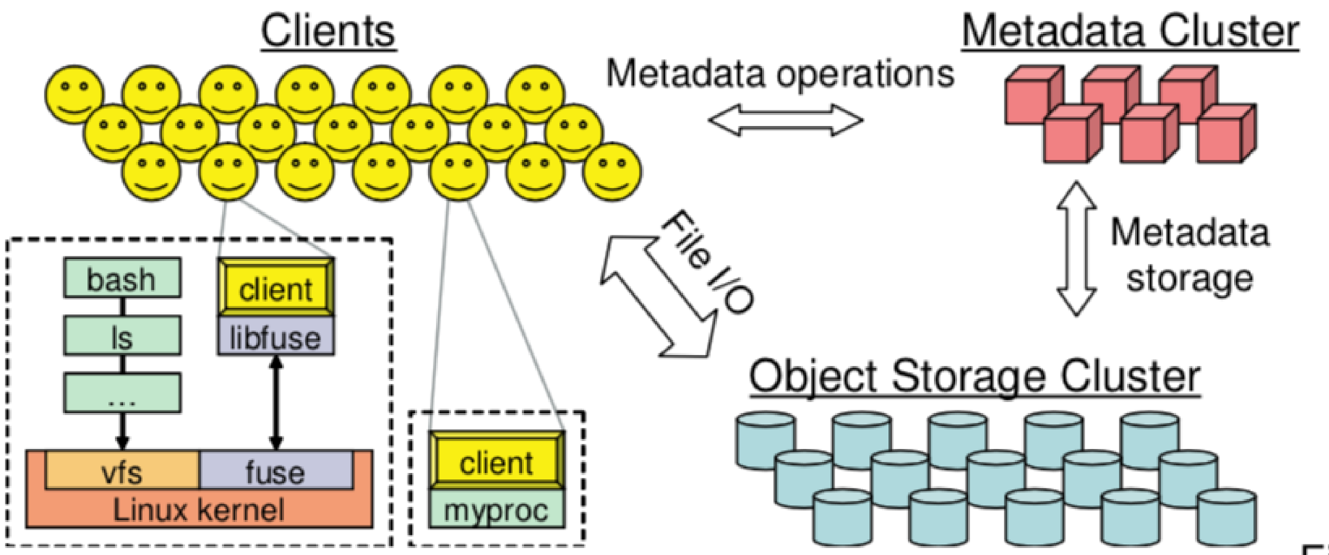
- Distributed file system
  - File system of which contents are stored in an array of different nodes.
  - Built-in replication, data migration, etc.
  - Disks can be located in dedicated servers (off compute nodes), or be spread out in compute nodes over the data center (on compute nodes).
  - Need high-speed and low-latency network between nodes that host disks.
  - Aggregated bandwidth of the file system can be enormous!

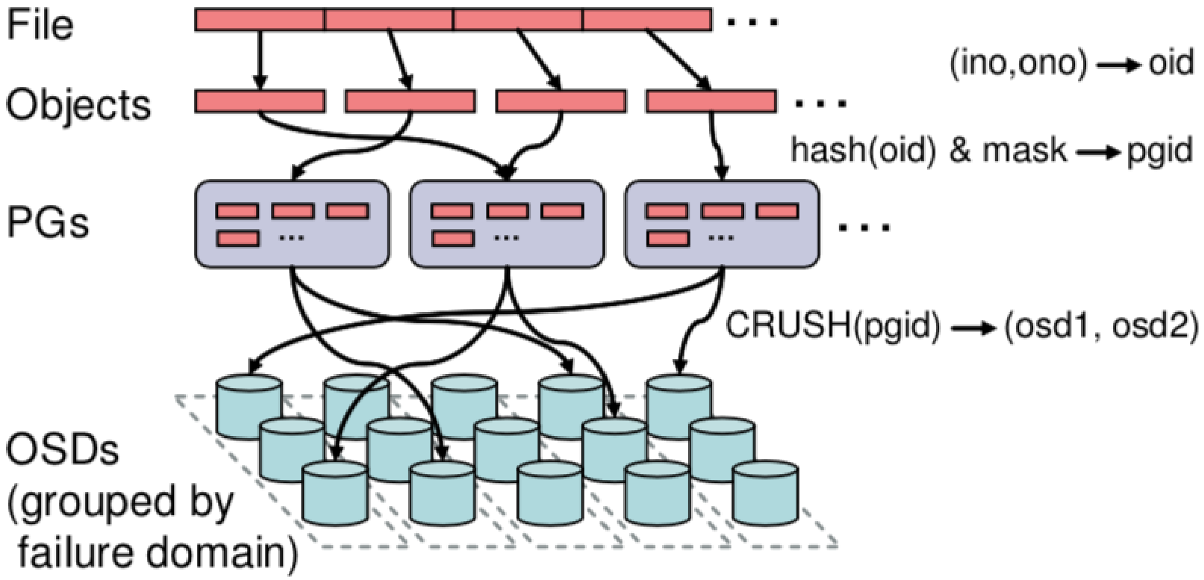
- Examples:
  - Open source: Ceph, GlusterFS, Lustre.
  - Not public: Google GFS and several others.

Ceph

- Started as an object store based on RADOS
  - RADOS: Reliable Autonomic Distributed Object Store
- Ceph can also expose block devices (RBD) as well as file systems (CephFS).
  - E.g.: block devices can be used for virtual disks, objects for templates (images).
- Used by several OpenStack deployments and also at CERN (65 PB).

Anatomy of a Distributed File System





Images from: S. Weil et al. 2006. Ceph: A Scalable, High-Performance Distributed File System. OSDI 2006.

Universiteit Leiden