NYU POLYTECHNIC SCHOOL OF ENGINEERING

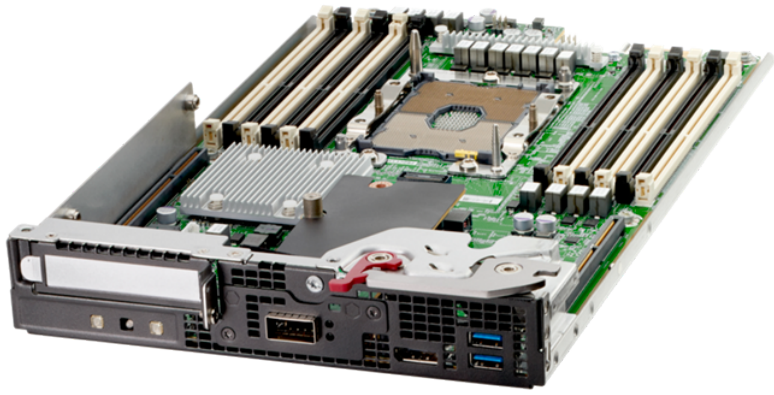
Virtualization and Parallel Programming in Data Centers

Part I

SERVER VIRTUALIZATION

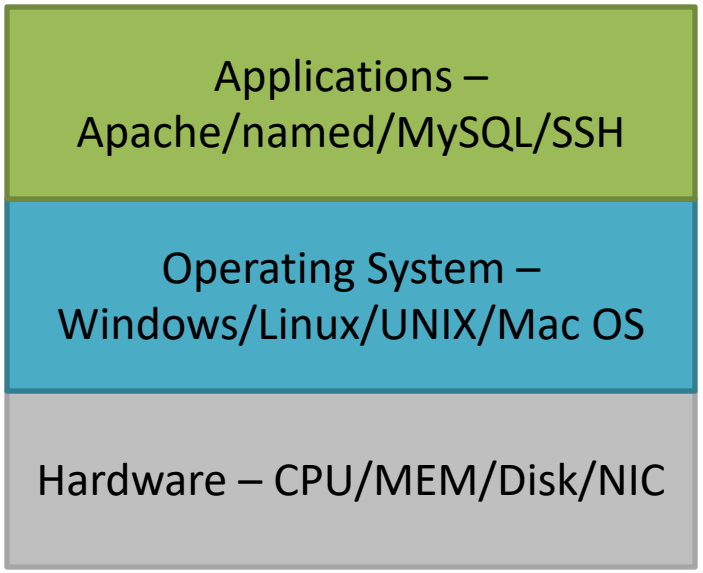
High Performance Servers

- HPE ProLiant e910 Server Blade is certified for extended thermal environments of 0° to 55° C within the HPE Edgeline EL8000 chassis.
- Each server blade can be configured with up to 24 M.2 NVMe SSD (M.2 form factor Non-Volatile Memory Express Solid State Drive, Video link)
- The server is based on Intel® Xeon® Scalable processors to give increased compute power and speed, e.g., Gold 6212U has 24 cores running at 2.4 GHz, 37.5 MB L3 cache, 2933 MT/sec for DDR4 (Double Data Rate 4) memory, up to 1TB external memory, 165 W power consumption.



A Typical Server

- Software, OS, data all tied to one machine
- Hardware failure?
- Network failure?
- Routine maintenance?
- Hardware upgrade?



All cause **service disruption**.

In case of failure, we want to be able to **migrate** applications to **other physical machines**, transparent to the users!

A Typical Server

- Modern hardware packs a lot of resources on a single machine
- But most of the times, it is desired to run single application per machine
  - Isolation
  - Operability
- Too many resources for a single application!

How to **fully** utilize resources?

How to partition physical machines into isolated smaller machines for each application?

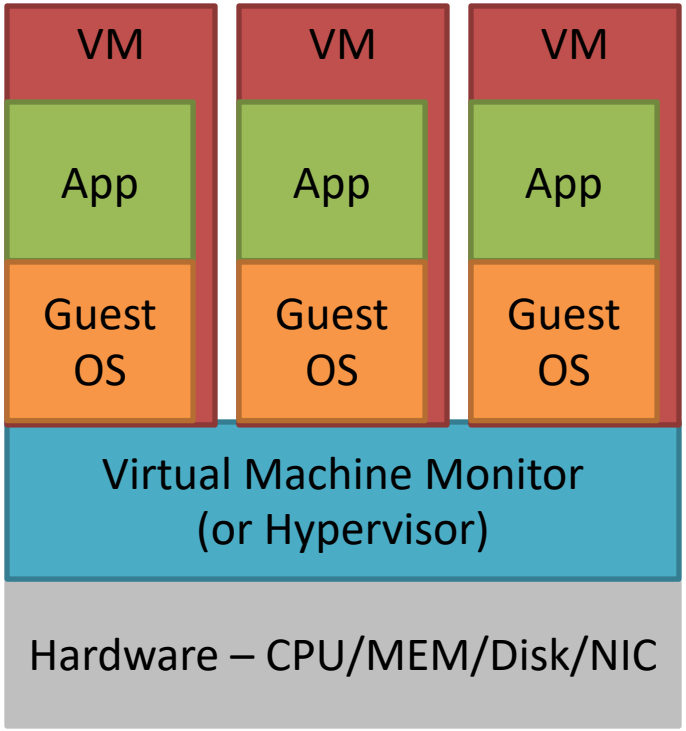
Definition

- Virtualization is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources*
- It is the process by which one computer hosts the appearance of many computers.
- Virtualization is used to improve IT throughput and costs by using physical resources as a pool from which virtual resources can be allocated.

*VMWare white paper, *Virtualization Overview*

Server Virtualization

- Run Virtual Machines (VM) using Hypervisor/Virtual Machine Monitor (VMM) to achieve:
  - Fidelity
    - The VMs should be identical to physical hardware
  - Isolation or Safety
    - A VM cannot manipulate other VMs resources/operation
  - Performance
    - Little to no performance penalty for running VMs compared to physical hardware



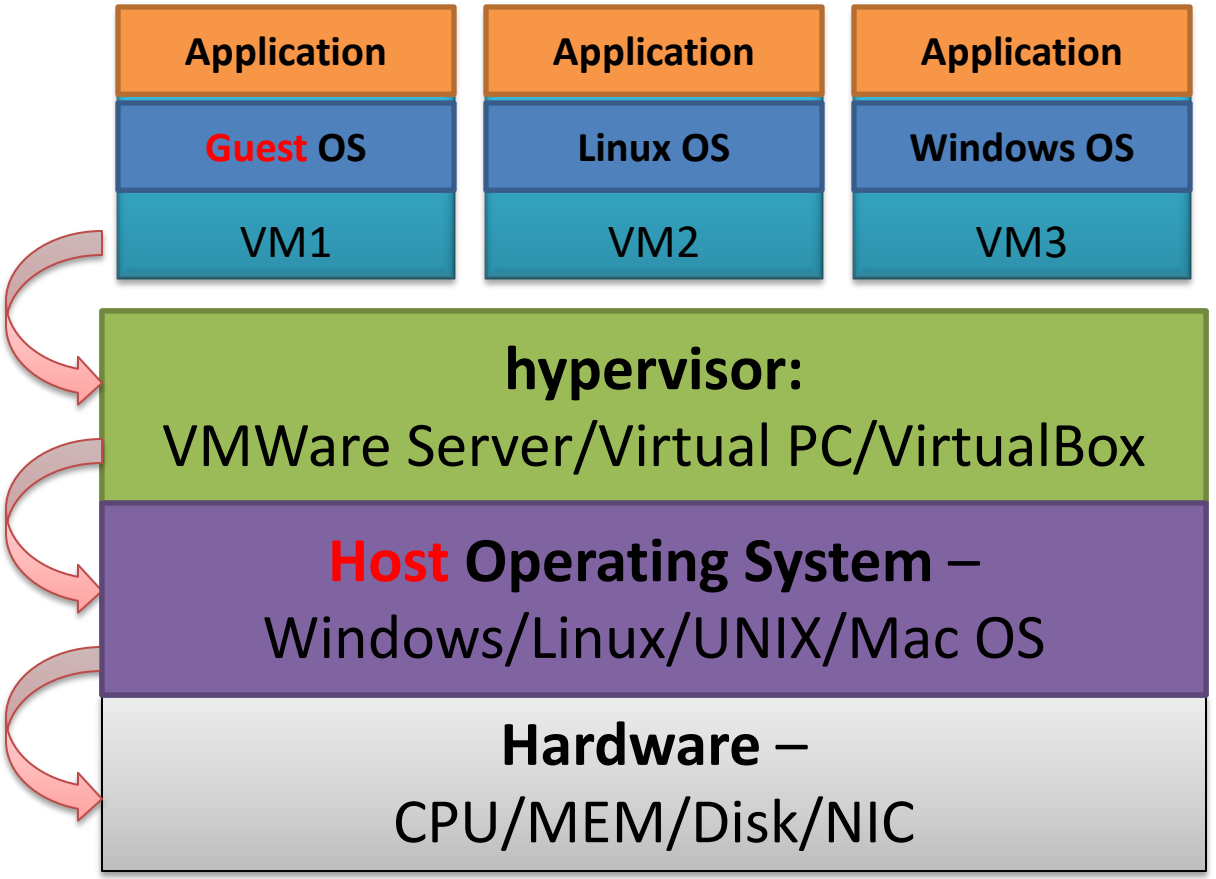
Hypervisor

- A hypervisor, a.k.a. a virtual machine manager/monitor (VMM), or virtualization manager, is a program that allows multiple operating systems to share a single hardware host.
- Each guest operating system appears to have the host's processor, memory, and other resources all to itself.
- The hypervisor is actually controlling the host processor and resources, allocating what is needed to each operating system in turn (i.e., scheduling), and making sure that the guest operating systems (i.e., VMs) cannot disrupt each other.

Two Types of Hypervisor

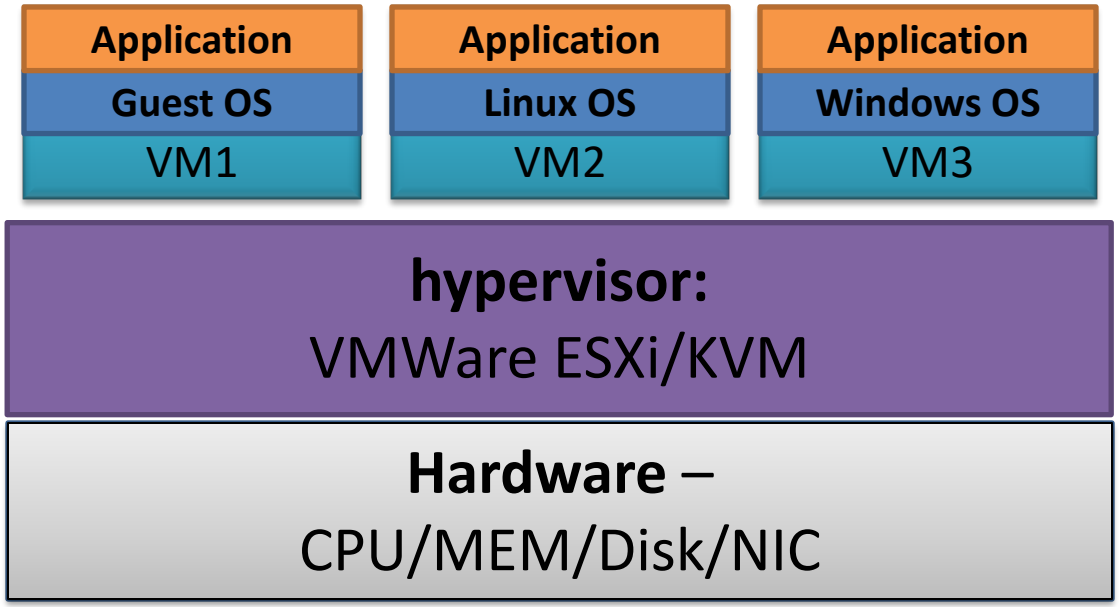
- There are two types of hypervisors: Type 1 and Type 2.
- Type 2 hypervisors (also called hosted hypervisor) support guest virtual machines by coordinating calls for CPU, memory, disk, network and other resources through the physical host's operating system. This makes it easy for an end user to run a virtual machine on a personal computing device. Examples of this type of hypervisor include VMware Fusion, Oracle Virtual Box, Oracle VM for x86, Solaris Zones, Parallels and VMware Workstation.
- In contrast, a Type 1 hypervisor (also called a bare metal hypervisor) is installed directly on physical host server hardware just like an operating system. Type 1 hypervisors run on dedicated hardware. They require a management console and are used in data centers. Examples of this type of hypervisor include Oracle OVM for SPARC, ESXi, Hyper-V and KVM.
- Regardless of the implementation, VMs and their guest operating systems are typically unaware of which type of hypervisor is implemented, as they interact only with the hypervisor itself.

Type 1: Hosted Hypervisors



- **Disadvantages:**
  Extra layer == overhead
- VM Software consumes CPU and MEM
- **Inefficient/slow**

Type2: Bare-Metal Hypervisors



- The hypervisor (e.g., ESXi) has direct access to the hardware; thus, it is more efficient and achieves higher performance than the hosted architecture.

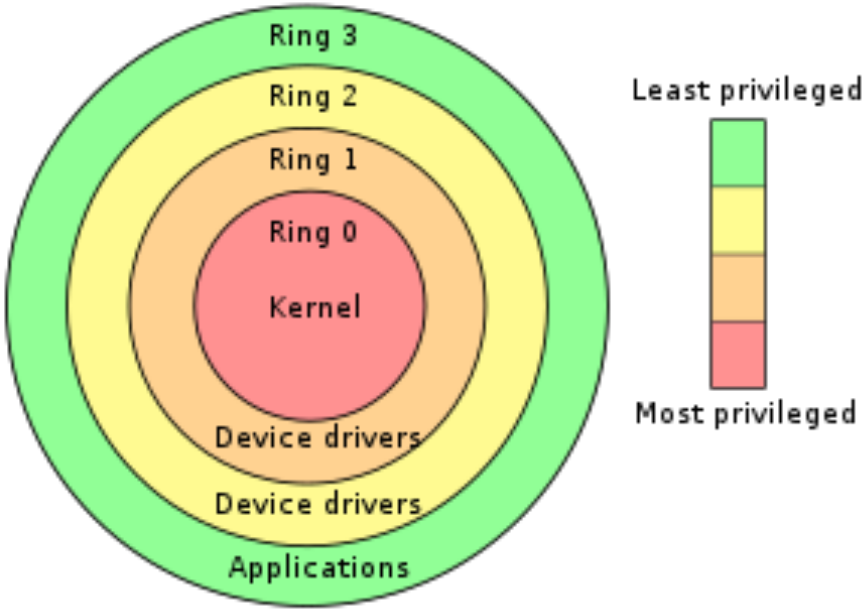
Server Virtualization Techniques

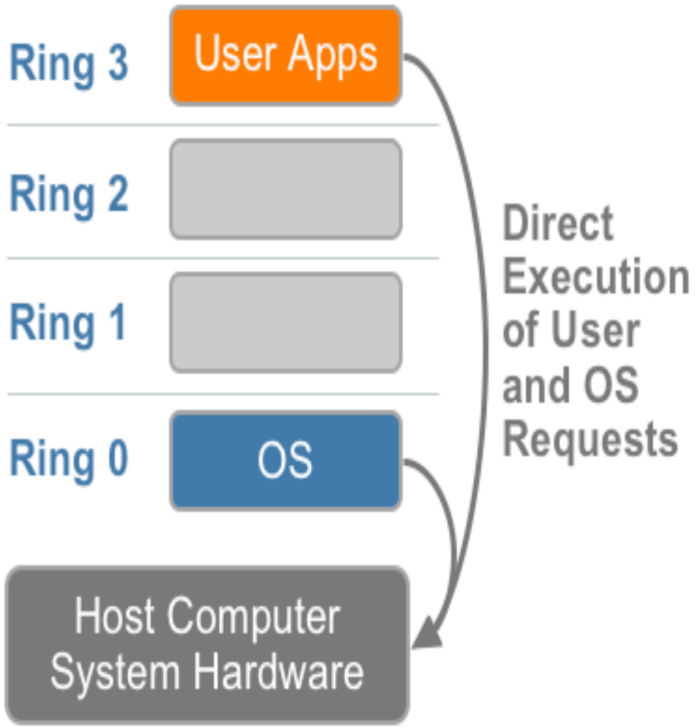
- Full virtualization
  1. Software-assisted
  2. Hardware-assisted

- Paravirtualization
  - **Para:** an English affix of Greek origin that means beside, with, or alongside.

  - Alongside virtualization refers to communication between the guest OS and the hypervisor to improve performance and efficiency

x86 CPU architecture Without Virtualization!





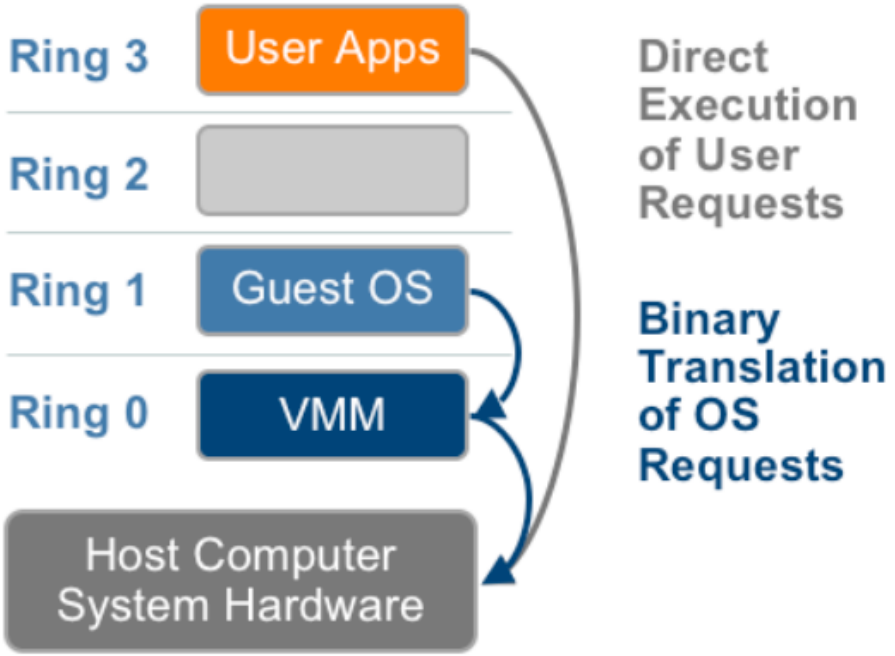
Privilege rings for the x86 available in protected mode

Full Virtualization: Software-assisted

Guest OS doesn't know that it is virtualized!

- It enables hypervisors to run an unmodified guest operating system
- It can have combination of Direct Exc. and Binary translation (BT)
- User application instructions are executed directly.
- Guest OS instructions are binary translated by the VMM and then executed. This is because such instructions may be hardware-related.
- **No modifications to the guest OS**
- No modifications to the hardware.

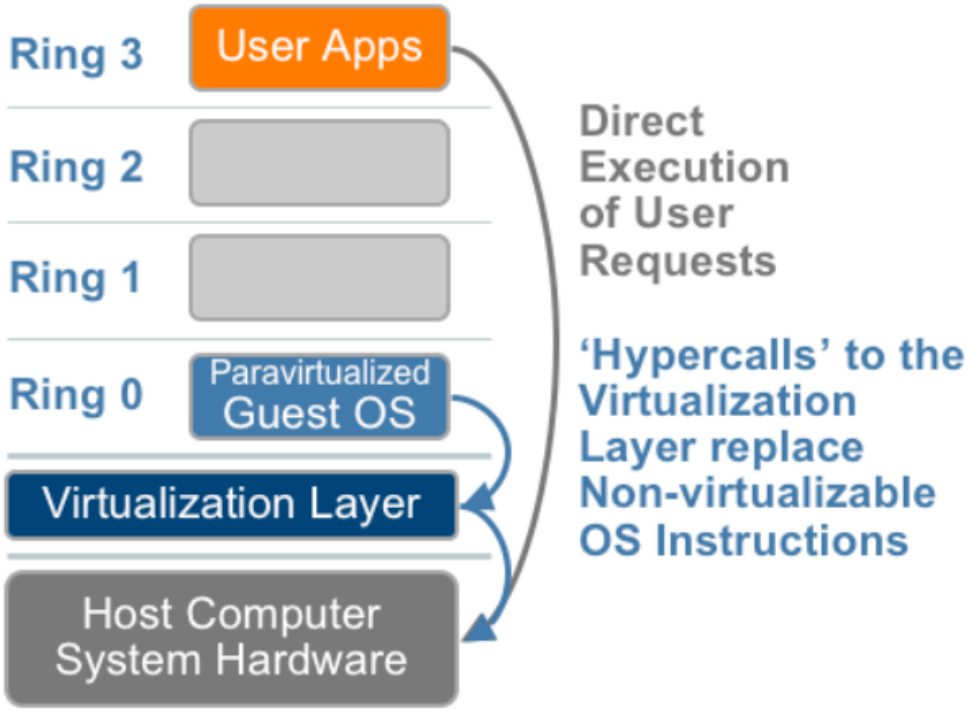
- E.g., VMWare Player, Microsoft virtual server



Paravirtualization

Guest OS knows that it is virtualized!

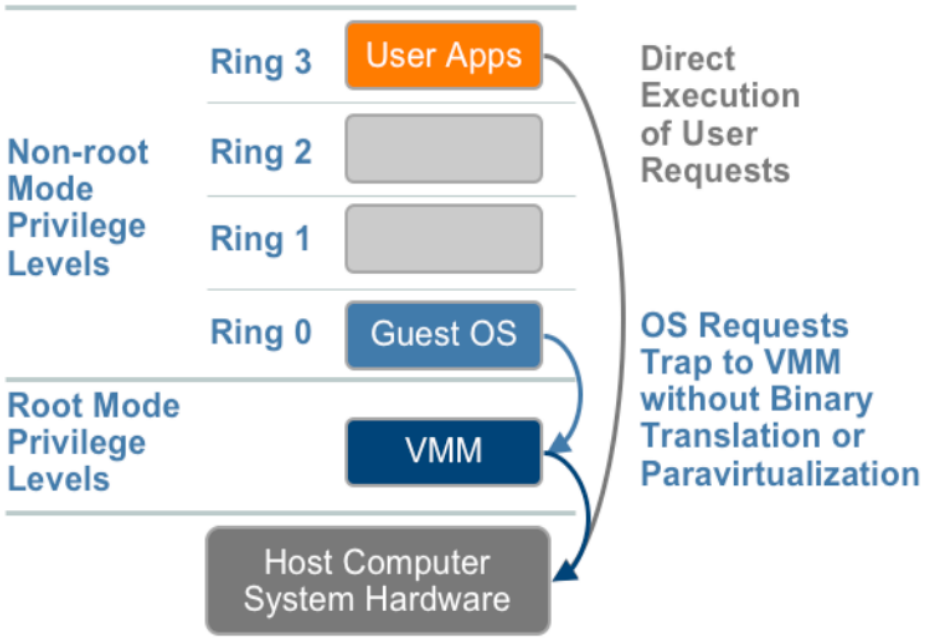
- Paravirtual mode does not require the host computer to support hardware-assisted virtualization technology, but does require the guest operating system to be modified for the virtualization environment.
- Better performance than those requiring full virtualization mode.
- Instructions that cannot be virtualized are replaced with hypercalls that communicate with the underlying hypervisor.
- It reduces virtualization overhead but adds complexity by modifying the guest OS.
- E.g. XEN, VMWare ESXi, VirtualBox



Full Virtualization: Hardware-Assisted

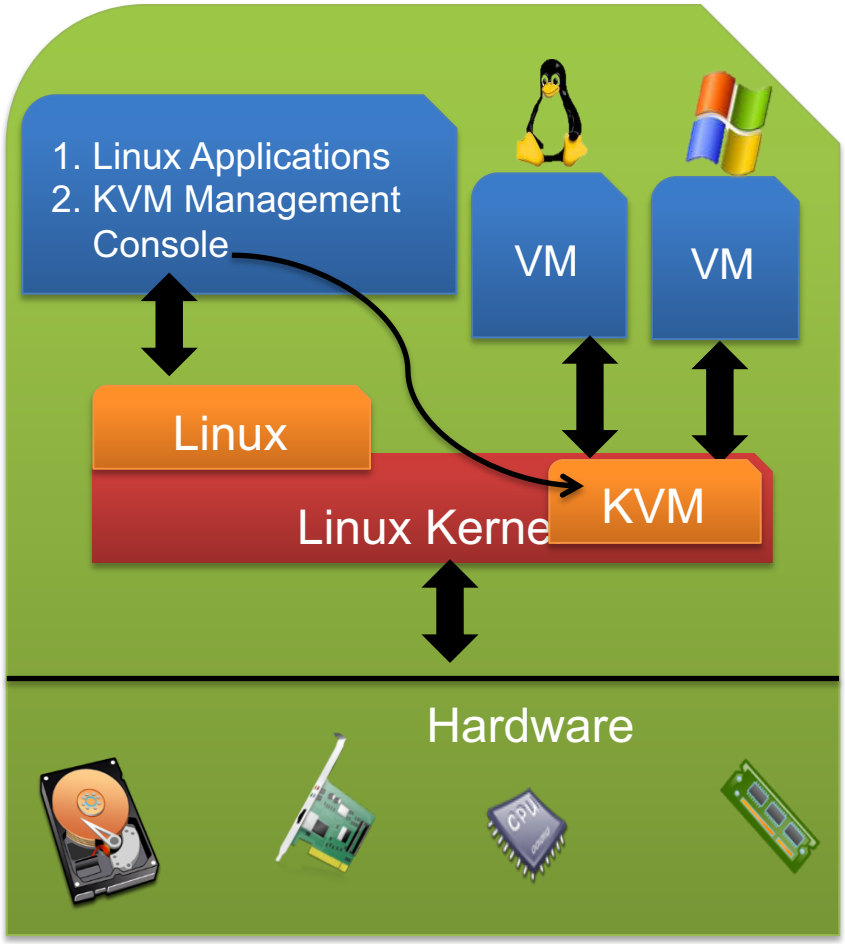
Guest OS doesn't know that it is virtualized!

- **Hardware** provides a new privilege level below Ring0 to run VMM
- Privileged and sensitive calls are set to **automatically** trap to the hypervisor.
- Removes the need for either binary translation or paravirtualization.
- The guest state is stored in Virtual Machine Control Structures (Intel VT-x) or Virtual Machine Control Blocks (AMD-V).
- VMware workstation, XEN, Linux KVM, Microsoft Hyper-V all support hardware assisted virtualization.



UBUNTU KVM

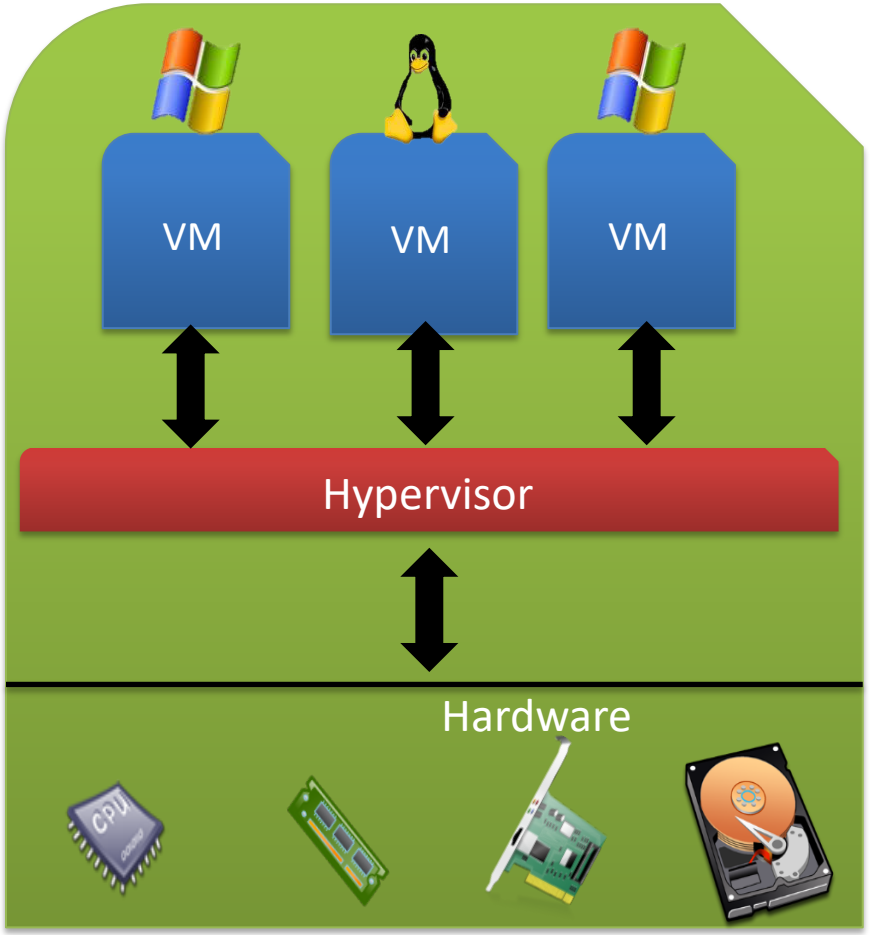
- KVM (Kernel Based VM) converts Linux into a type-1 (bare-metal) hypervisor.
- All hypervisors need some operating system-level components—such as a memory manager, process scheduler, input/output (I/O) stack, device drivers, security manager, a network stack, and more—to run VMs. KVM has all these components because it's part of the Linux kernel.
- Every VM is implemented as a regular Linux process, scheduled by the standard Linux scheduler, with dedicated virtual hardware like a NIC, graphics adapter, CPU(s), memory, and disks.



Architecture of KVM

VMWare ESXi

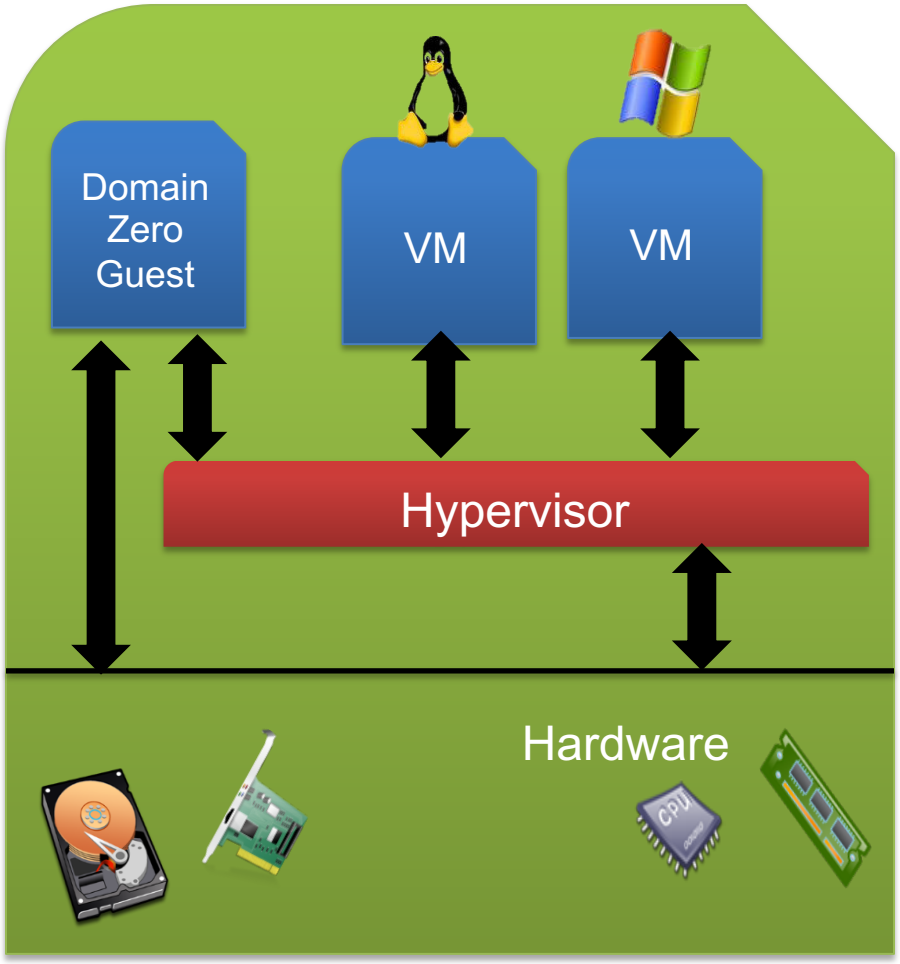
- Type1 bare metal approach.
- ESXi provides a virtualization layer that abstracts the CPU, storage, memory and networking resources of the physical host into multiple virtual machines.
- Applications running in virtual machines can access these resources without direct access to the underlying hardware.
- Takes advantage of support for hardware assisted virtualization for 64-bit OS on Intel processors.



Architecture of VMWare ESXi

CITRIX XEN SERVER

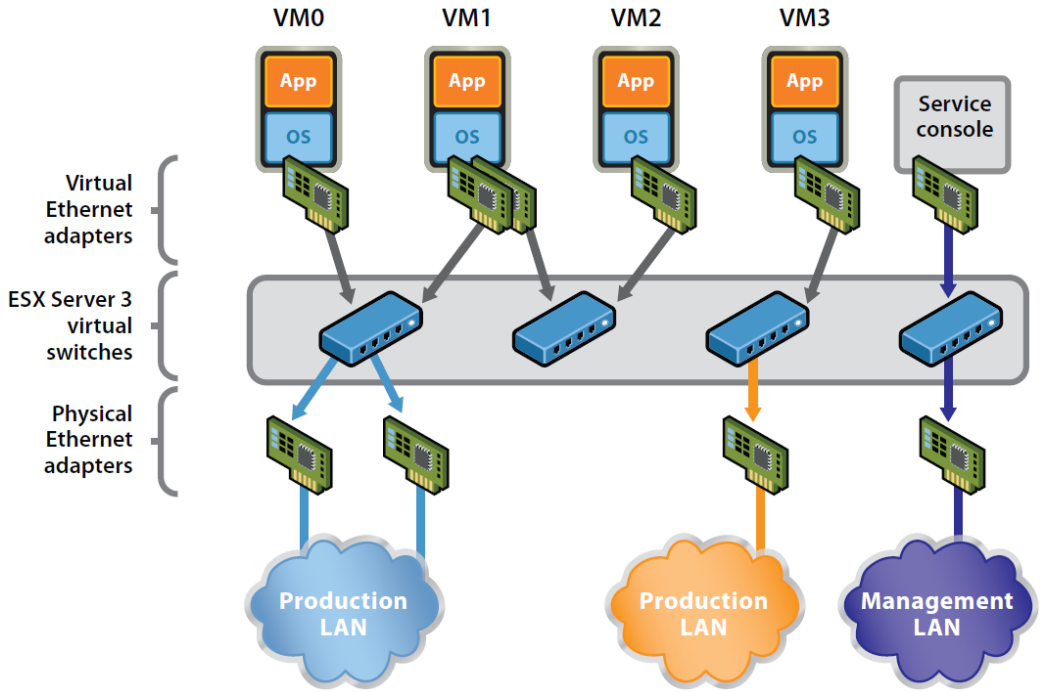
- Open source; type 1 bare metal appr
- Offers both Hardware Assisted Virtualization (HVM) and Para-Virtualization (PV)
- Needs virtualization support in the CPU for HVM and uses the advanced features of modern x86 CPUs (Intel's VT-x or AMD-V).
- Xen loads an initial OS which runs as a privileged guest called “domain 0”.
- The domain 0 OS, typically a Linux or UNIX variant, can talk directly to the system hardware (whereas the other guests cannot) and also talk directly to the hypervisor itself.
- It allocates and maps hardware resources for other guest domains.



Architecture of Xen

NETWORK VIRTUALIZATION

Virtual Networking for VMs



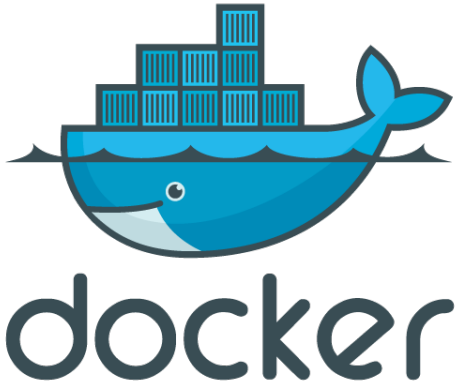
- A VM could have multiple virtual network interface cards (VNICs)
- A physical machine could have multiple NICs
- Flexible binding and switching between VNICs and NICs
- Flexible networking among VMs

Virtual NICs

- A virtual NIC is a L2 device, and it has its own MAC address
- Virtual NICs can be based on paravirtualization or full virtualization
- Some NICs have hardware-assisted virtualization – virtual private queue

Virtual Switches

- Functions
  - L2 forwarding
  - VLAN tag operations (adding/removing/filtering)
  - L2 security, checksum, etc.
  - Flow monitoring, QoS, ACL (access control ilist), etc.
- A virtual switch may not need to perform address learning in case the VNIC information is provided through a special channel (e.g., in VMware ESX server)
- Examples: Cisco Nexus 1000v, VMware vSwitch, Open vSwitch



INTRODUCTION TO DOCKER

http://lsi.vc.ehu.es/pablogn/docencia/ISO/9%20Aislamiento%20de%20Subsistemas%20y%20Contenedores/dockerintronovember-131125185628-phpapp02.pptx

Definitions

- Docker can package an application and its dependencies in a virtual container that can run on any Linux server.
- This enables the application to run in a variety of locations, such as on-premises, in a public cloud, and/or in a private cloud.
- Docker uses the resource isolation features of the Linux kernel and a union-capable file system to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.
- Because Docker containers are lightweight, a single server or virtual machine can run several containers simultaneously.
- A 2018 analysis found that a typical Docker use case involves running 8 containers per host, and that a quarter of analyzed organizations run 18 or more per host.

The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

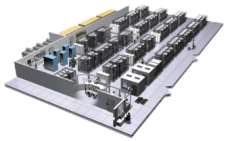
Customer Data Center

Public Cloud

Disaster recovery

Production Servers

Production Cluster



Contributor's laptop

Can I migrate smoothly and quickly?

The Matrix From Hell

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Cargo Transport Pre-1960

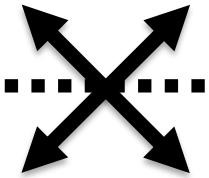
Multiplicity of Goods







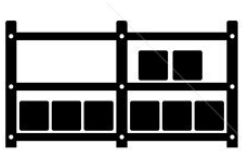
Do I worry about how goods interact (e.g. coffee beans next to spices)



Multipilicity of methods for transporting/storing

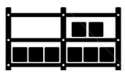




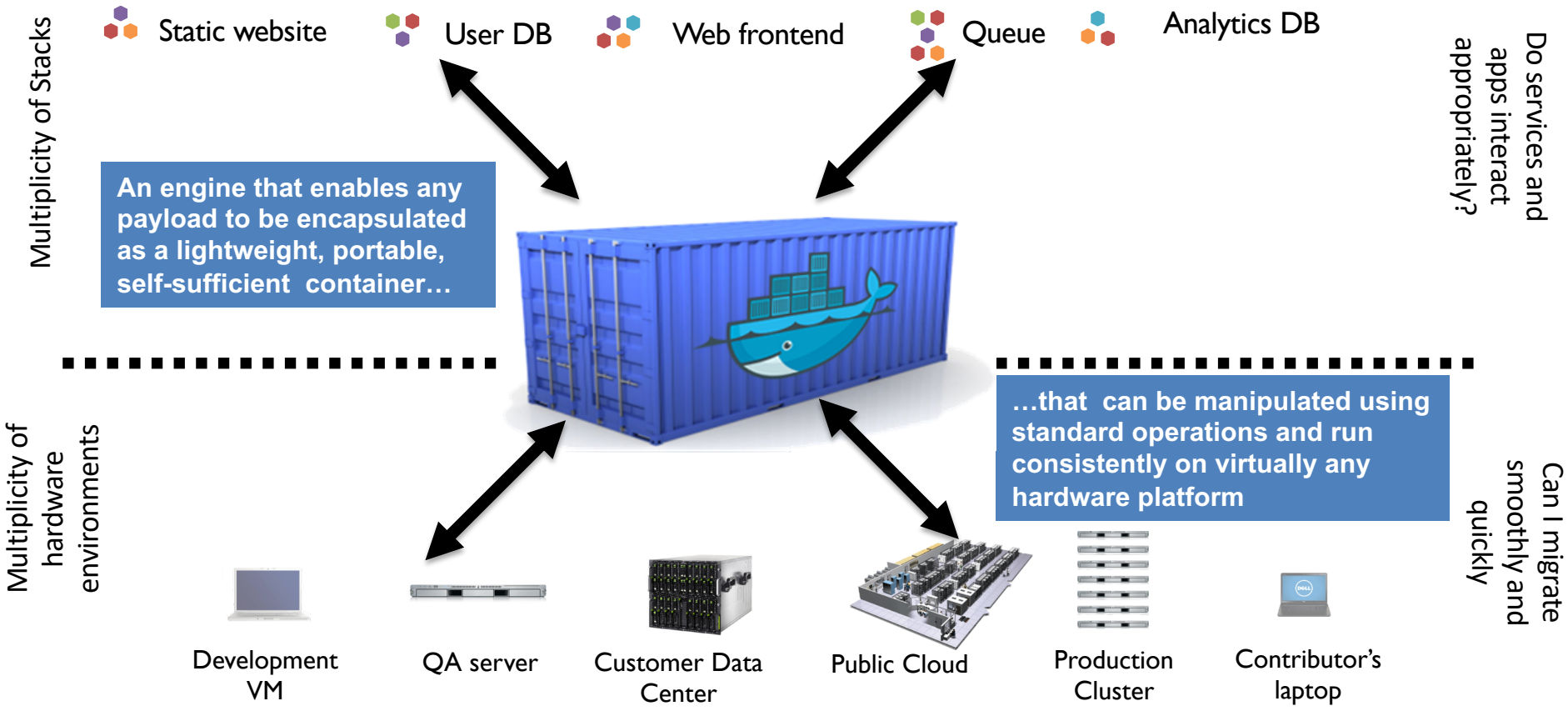

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Also a matrix from hell

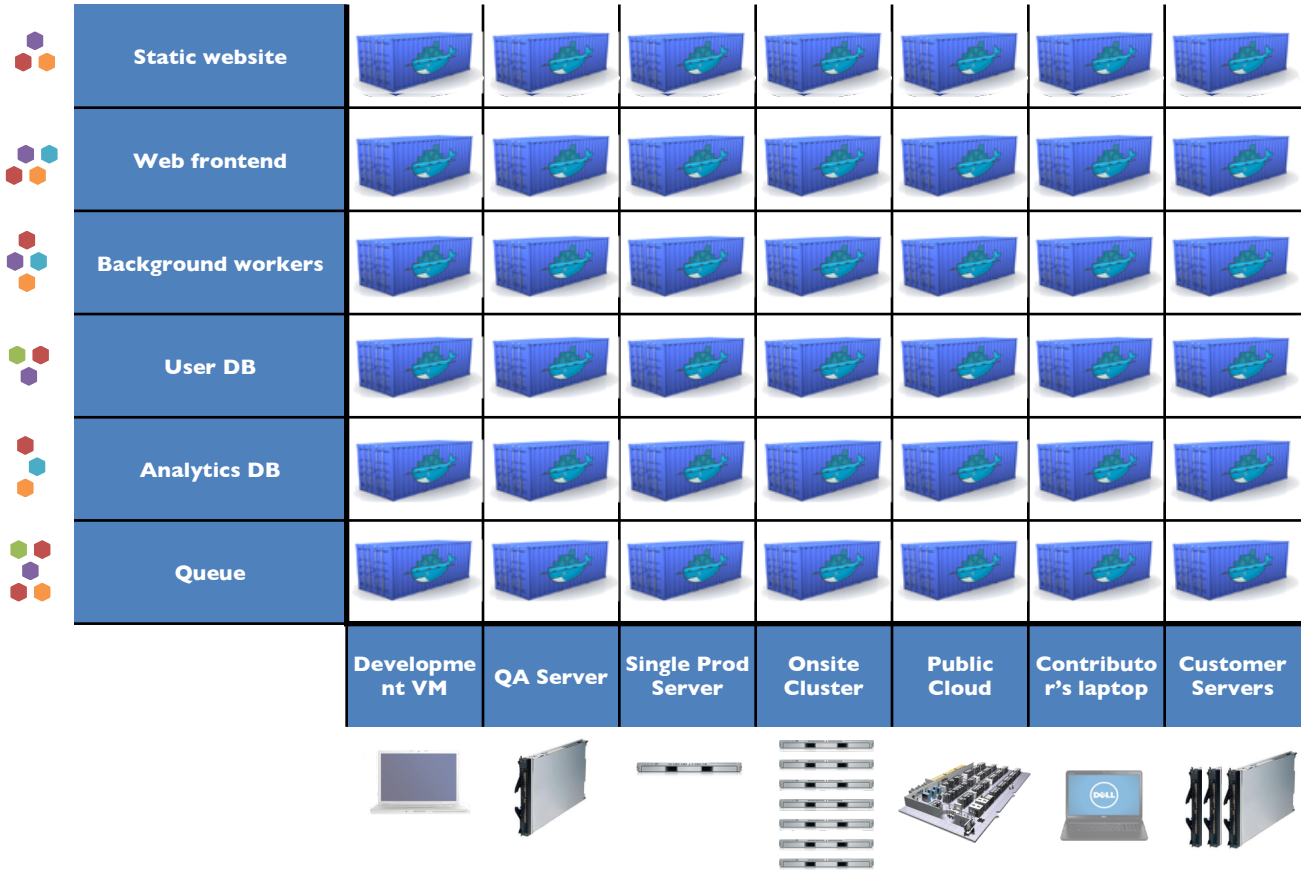
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Docker is a shipping container system for code



Docker eliminates the matrix from Hell



Why Developers Care

Build once…(finally) run anywhere

- A clean and portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
- Automate testing, integration, packaging…anything you can script
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
- Cheap, zero-penalty containers to deploy services
- A VM without the overhead of a VM
- Instant replay and reset of image snapshots

Why Devops* Cares?

Configure once…run anything

- Make the entire lifecycle more efficient, consistent, and repeatable
- Increase the quality of code produced by developers.
- Eliminate inconsistencies between development, test, production, and customer environments
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems
- Because the containers are so lightweight, significant performance, costs, deployment, and portability issues are normally associated with VMs

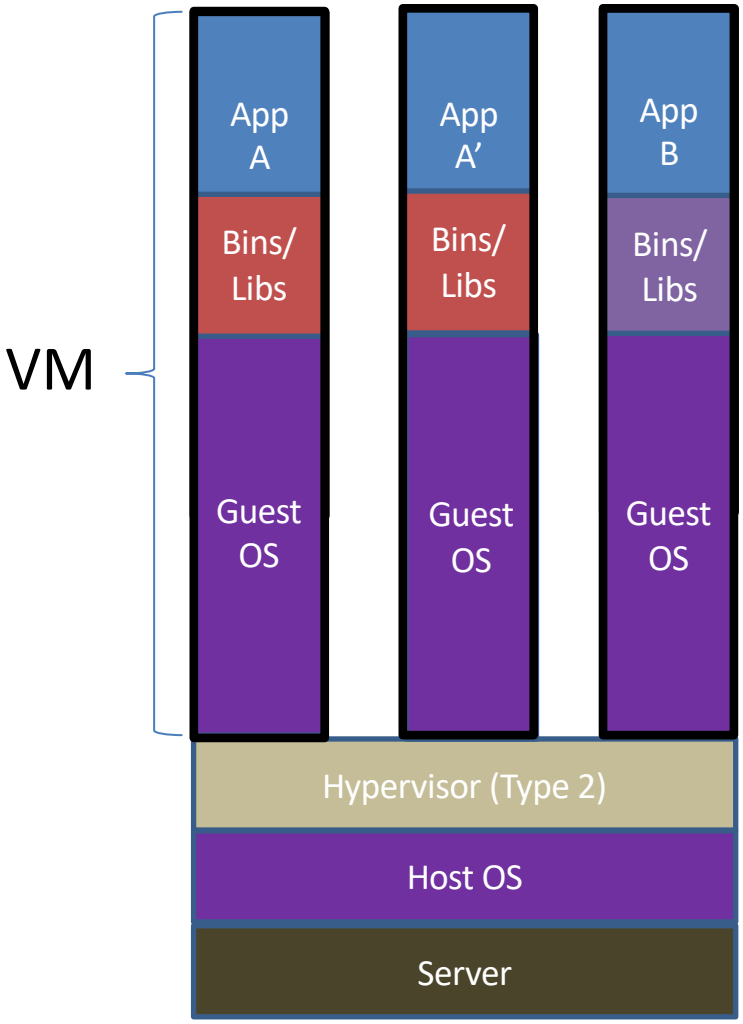
- ***DevOps** (development and operations) is the combination of practices and tools that increases an organization's ability to deliver applications and services at high velocity

More Technical Details

- Run everywhere
  - Regardless of kernel version
  - Regardless of host
  - Physical or virtual, cloud or not
- Run anything
  - If it can run on the host, it can run in the container
  - i.e. if it can run on a Linux kernel, it can run

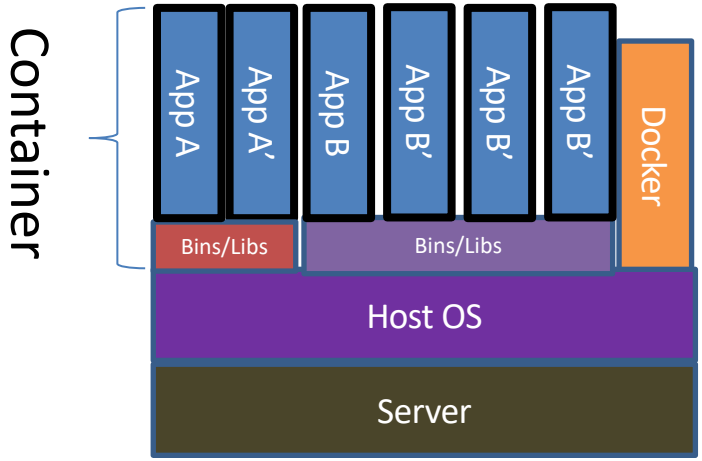
- High Level
  - It's a lightweight VM
  - Own process space
  - Own network interface
  - Can run stuff as root
  - Can have its own /sbin/init
- Low Level
  - Can also *not* have its own /sbin/init
  - Share kernel with host

Containers vs. VMs



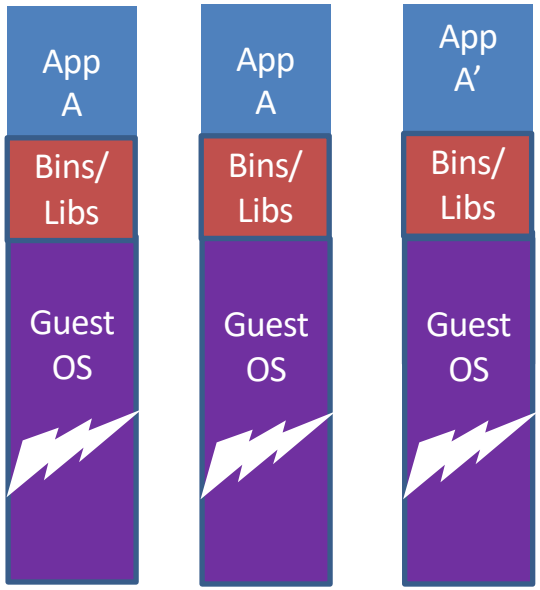
Containers are isolated, but share OS and, where appropriate, bins/libraries

…result is significantly faster deployment, much less overhead, easier migration, faster restart



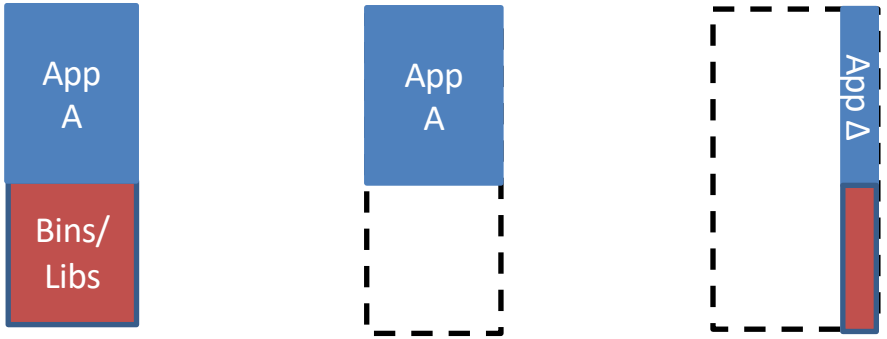
Why are Docker containers lightweight?

VMs



Every app, every copy of an app, and every slight modification of the app requires a new VM

Containers



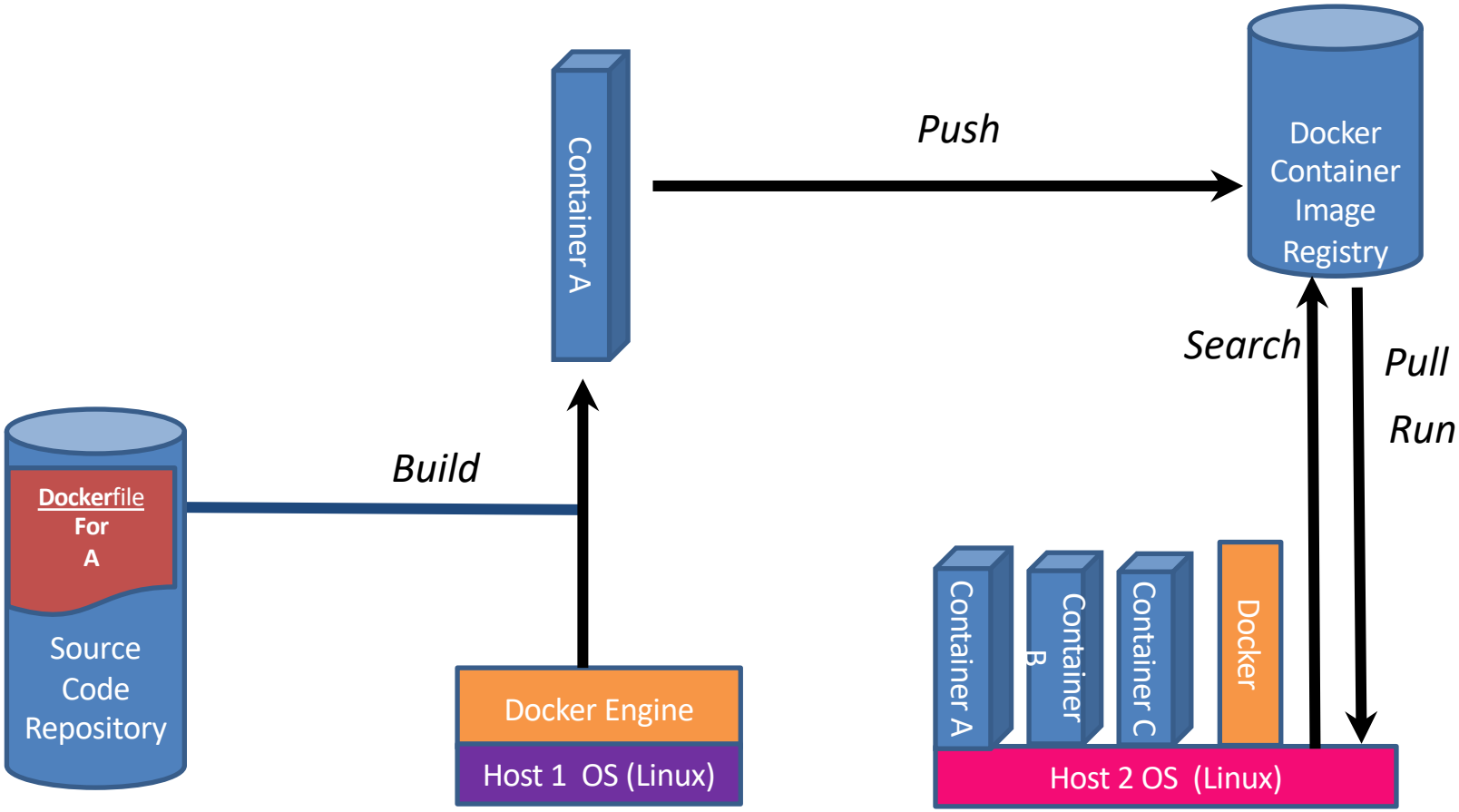
Original App (No OS to take up space, resources, or require restart)

Copy of App
No OS.
Can Share bins/libs

Modified App

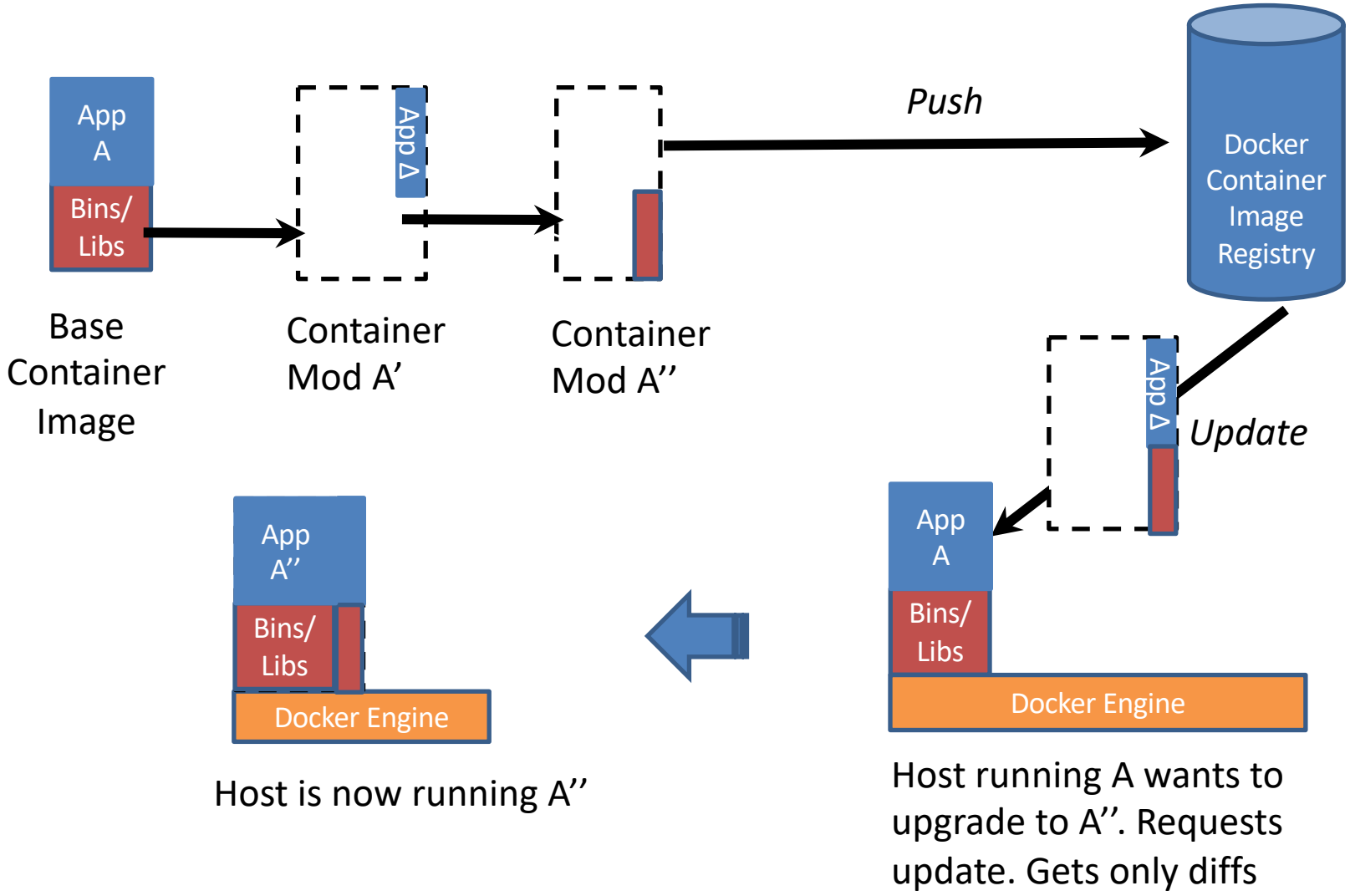
Copy on write capabilities allow us to only save the diffs between container A and container A'

What are the basics of the Docker system?



A container is **running instance** of a **Docker image** with top writable layer. A **Container** runs the actual application. A **container** includes an application and all of its dependencies.

Changes and Updates



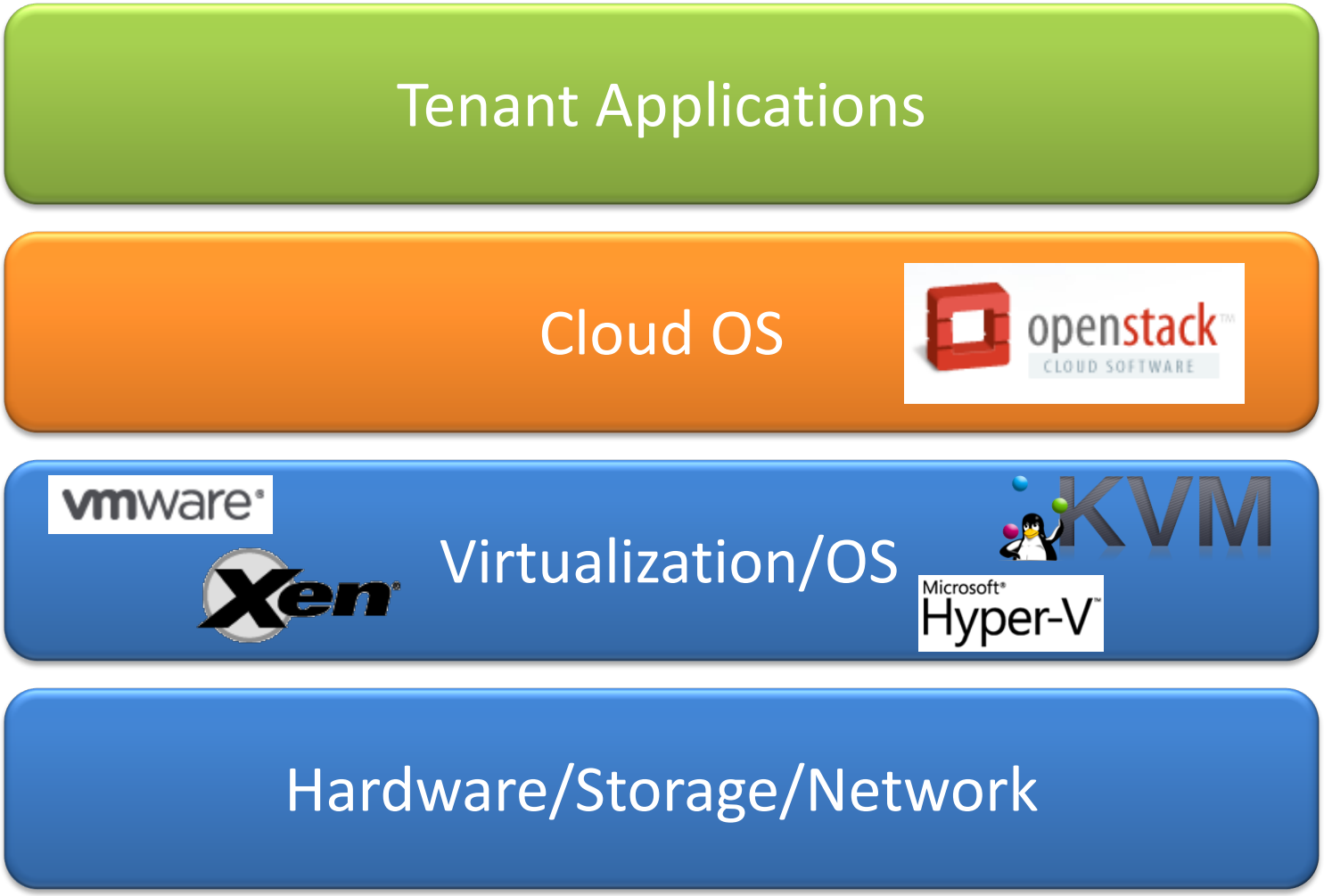
CLOUD OS!

Cloud Computing

- Cloud Computing ‘s characteristics:
  1. Scale with increasing demand
  2. Accessible anytime
  3. Automate the management of resources and hide details from users
  4. Accounting: Pay per use

- We need a framework to provide these functions

Cloud Operating System

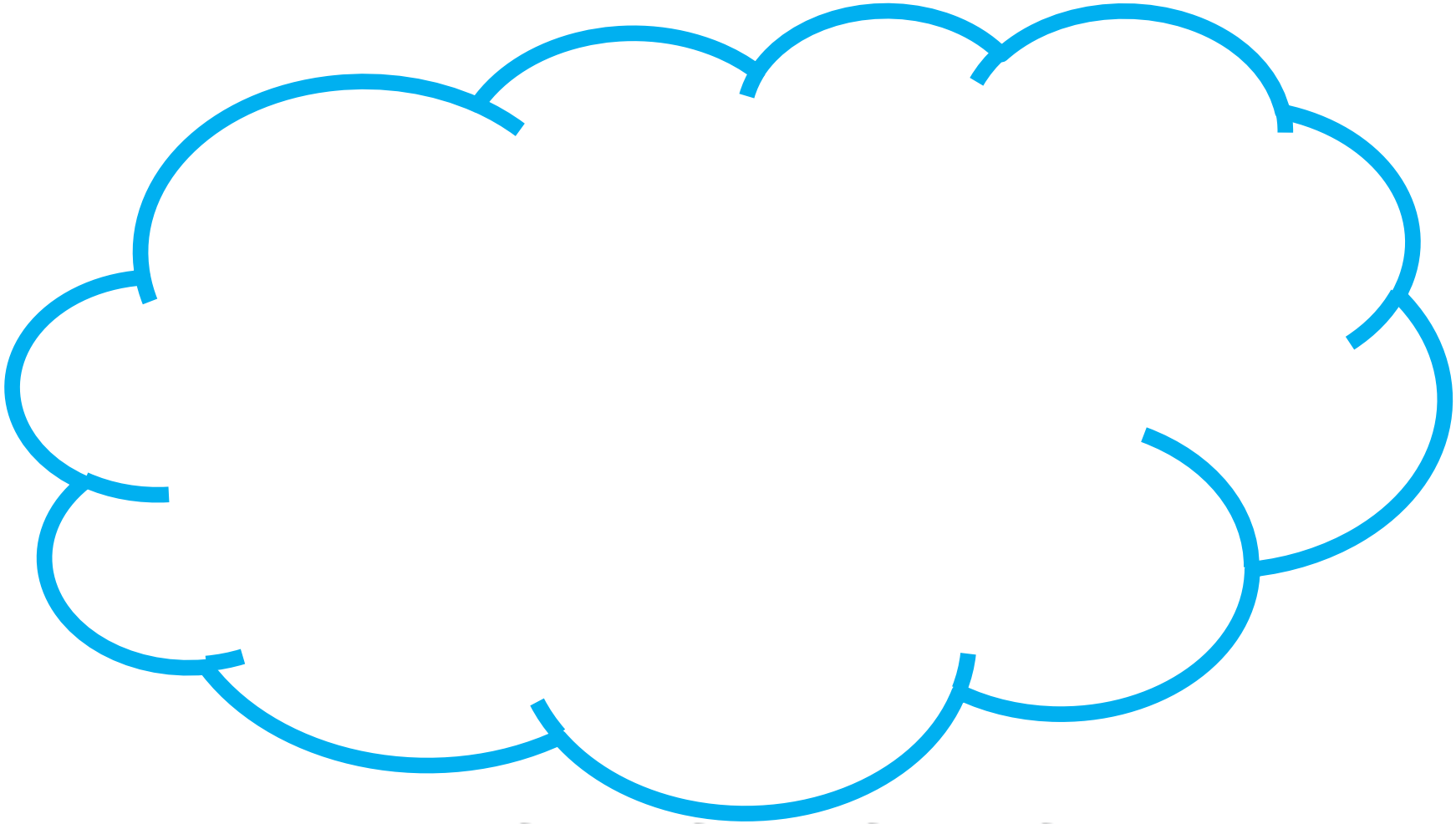


OpenStack Overview

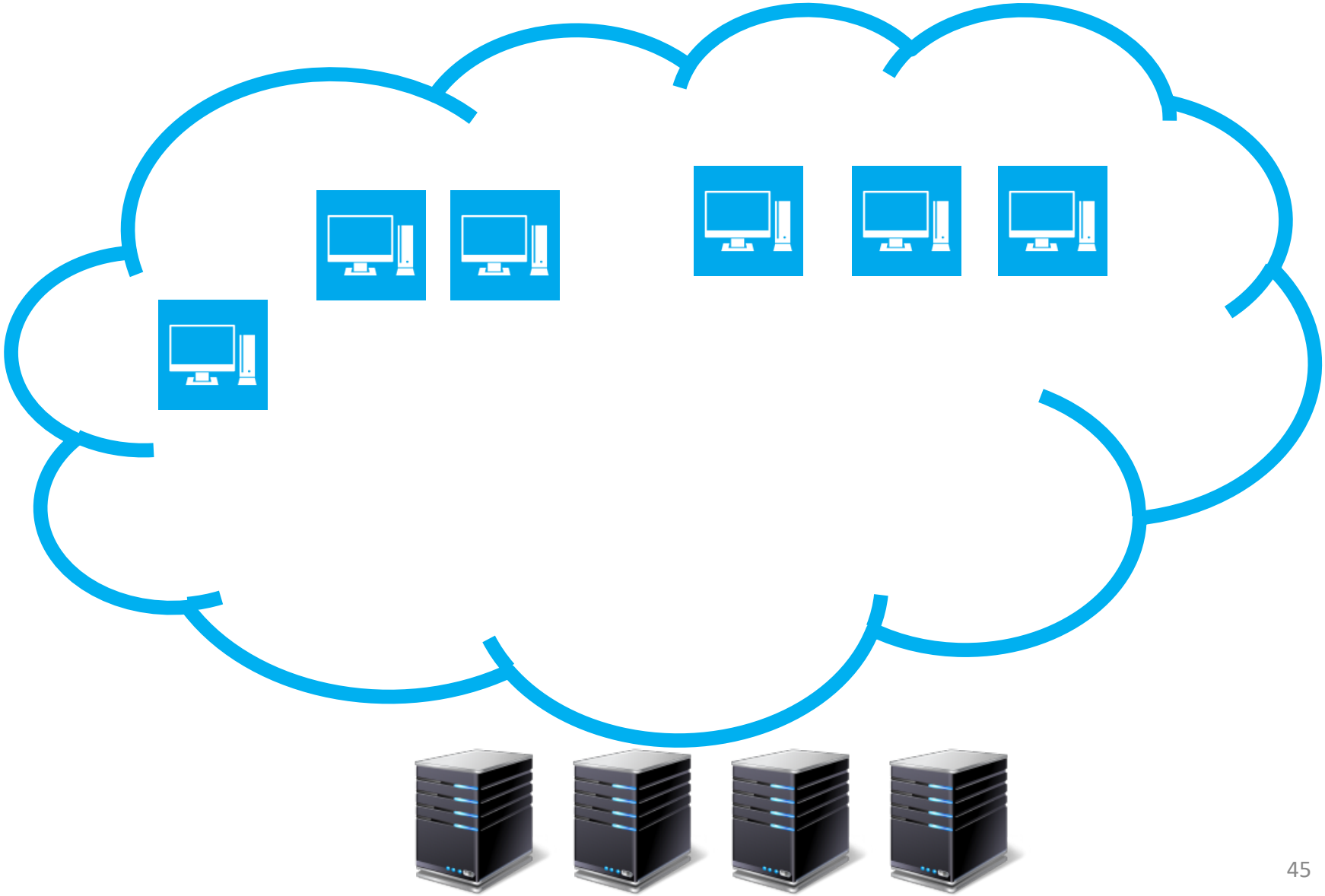
- Cloud OS developed by Rackspace and NASA
- Infrastructure as a Service

- Support Private Cloud and Public Cloud
- Open Source (Apache 2.0 license)
- OpenStack Foundation
- Popular and widely supported

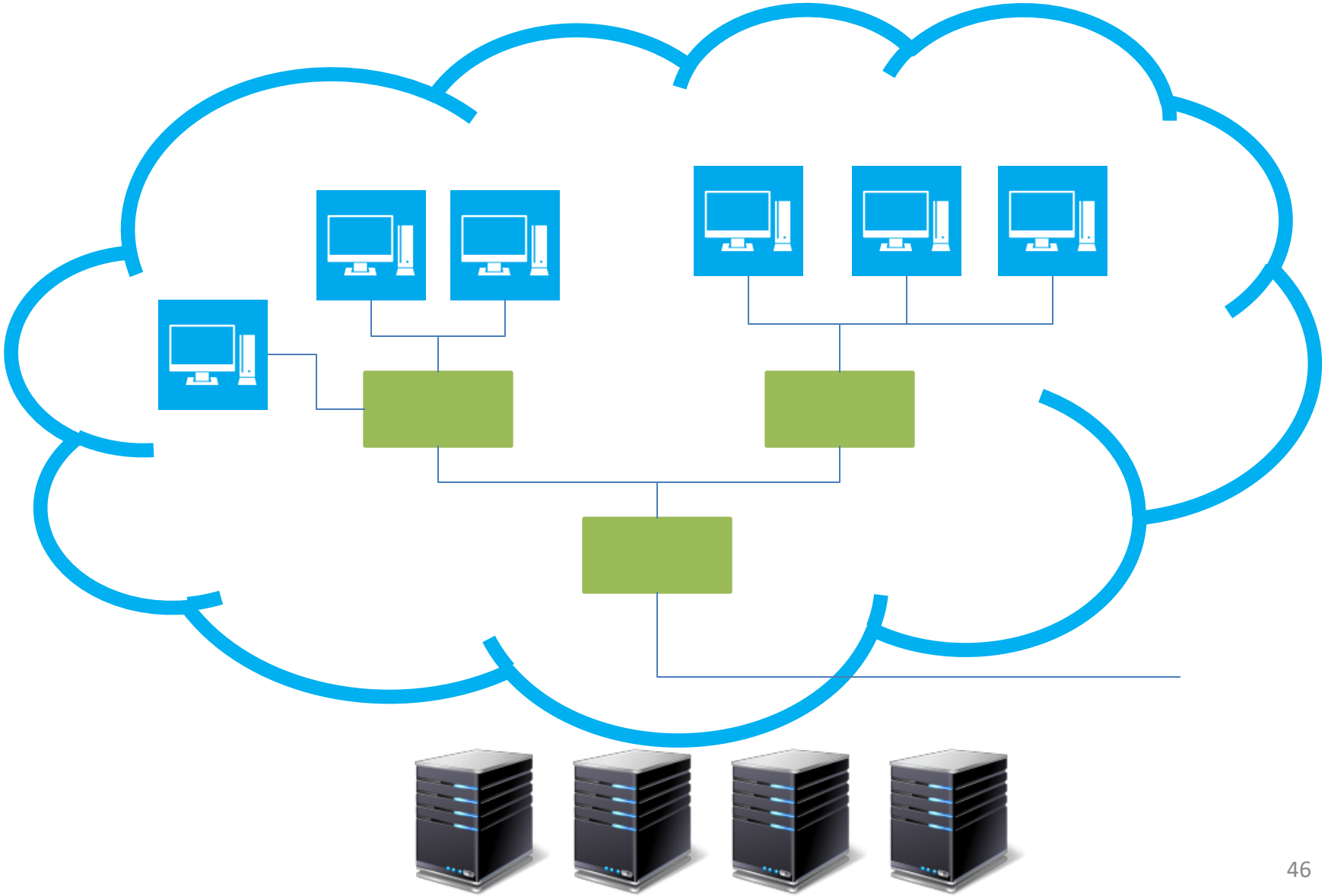
What does OpenStack provide?



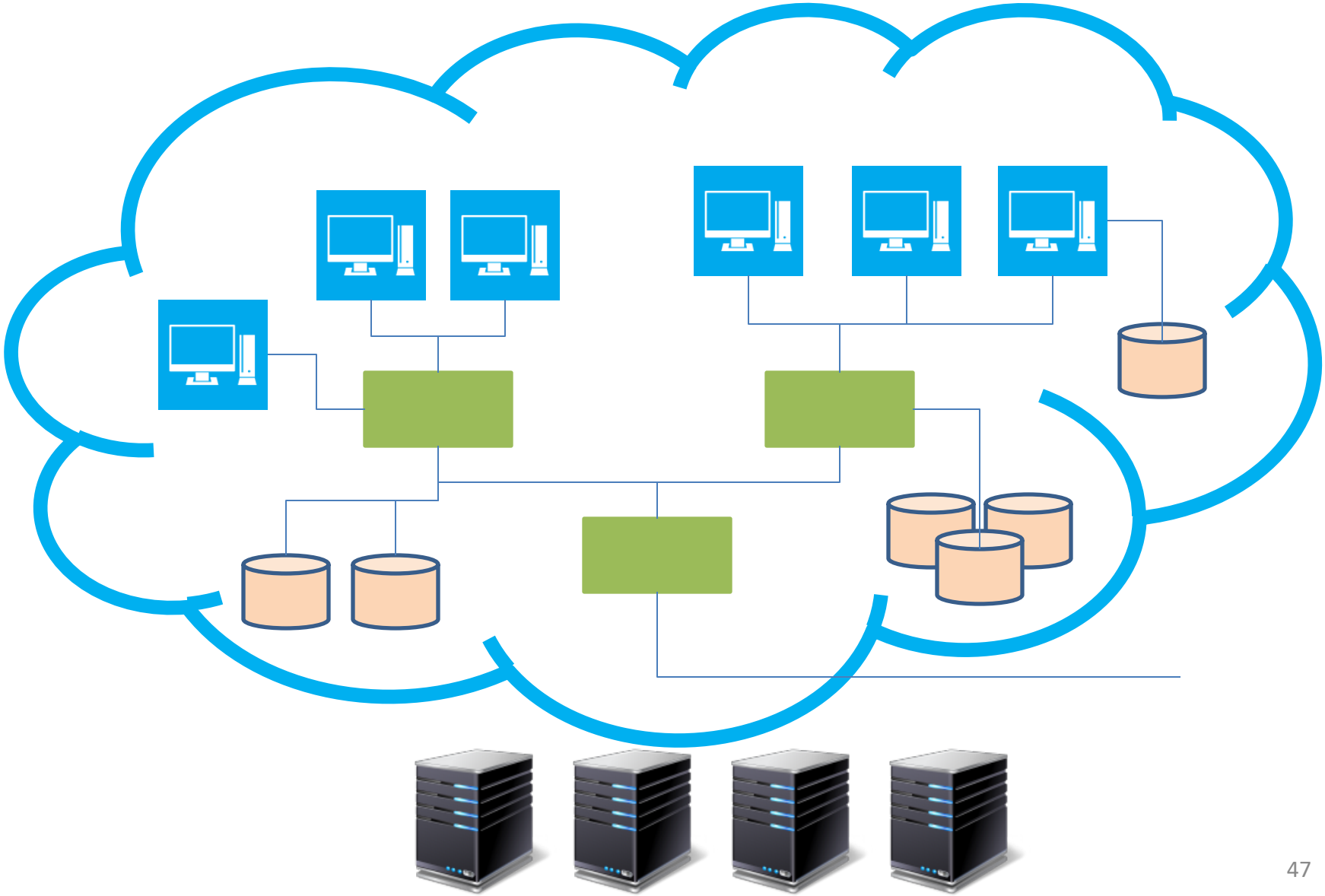
1. manage virtual machines



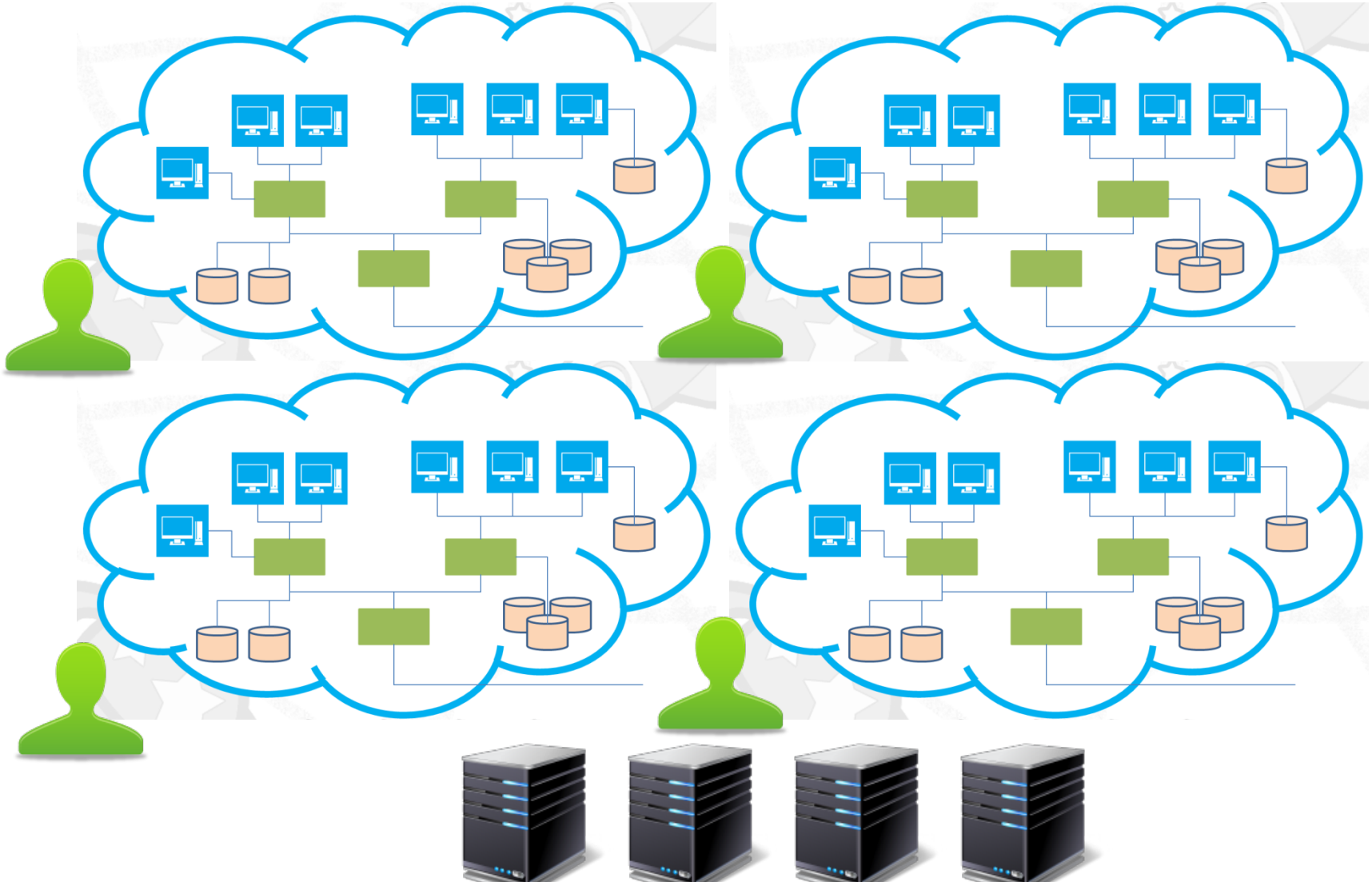
2. manage virtual networks



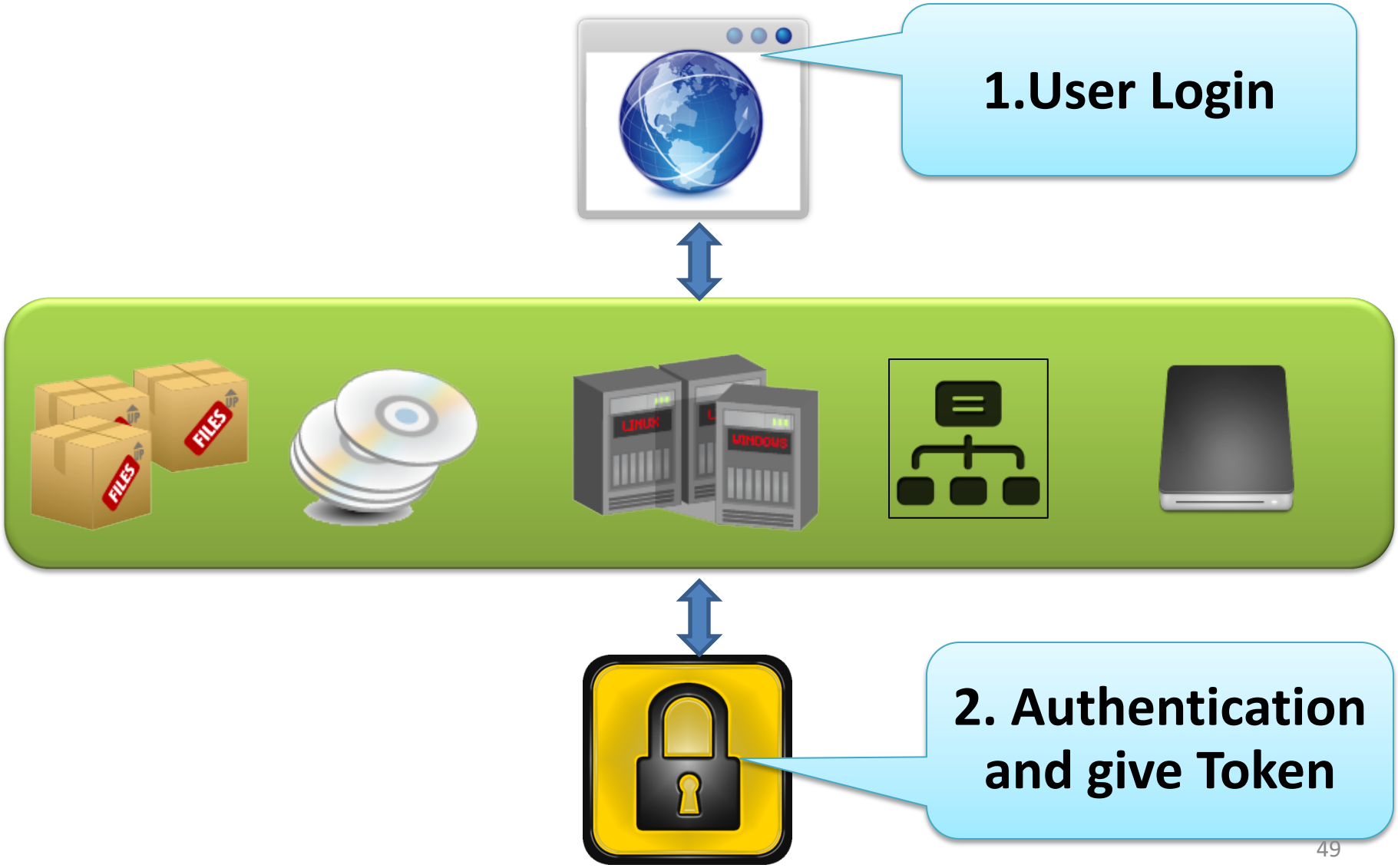
3. manage virtual storages



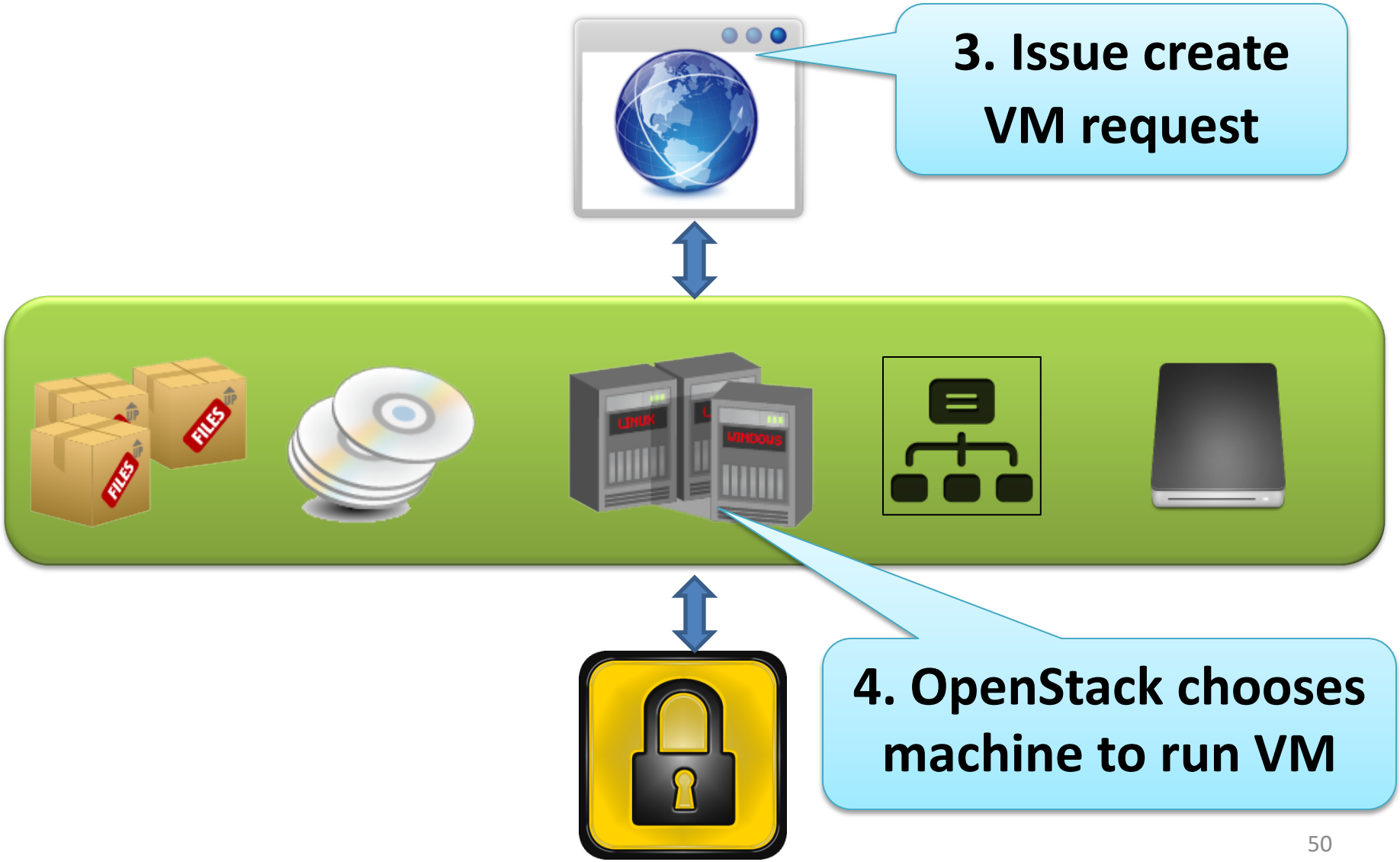
4. Multi-tenants



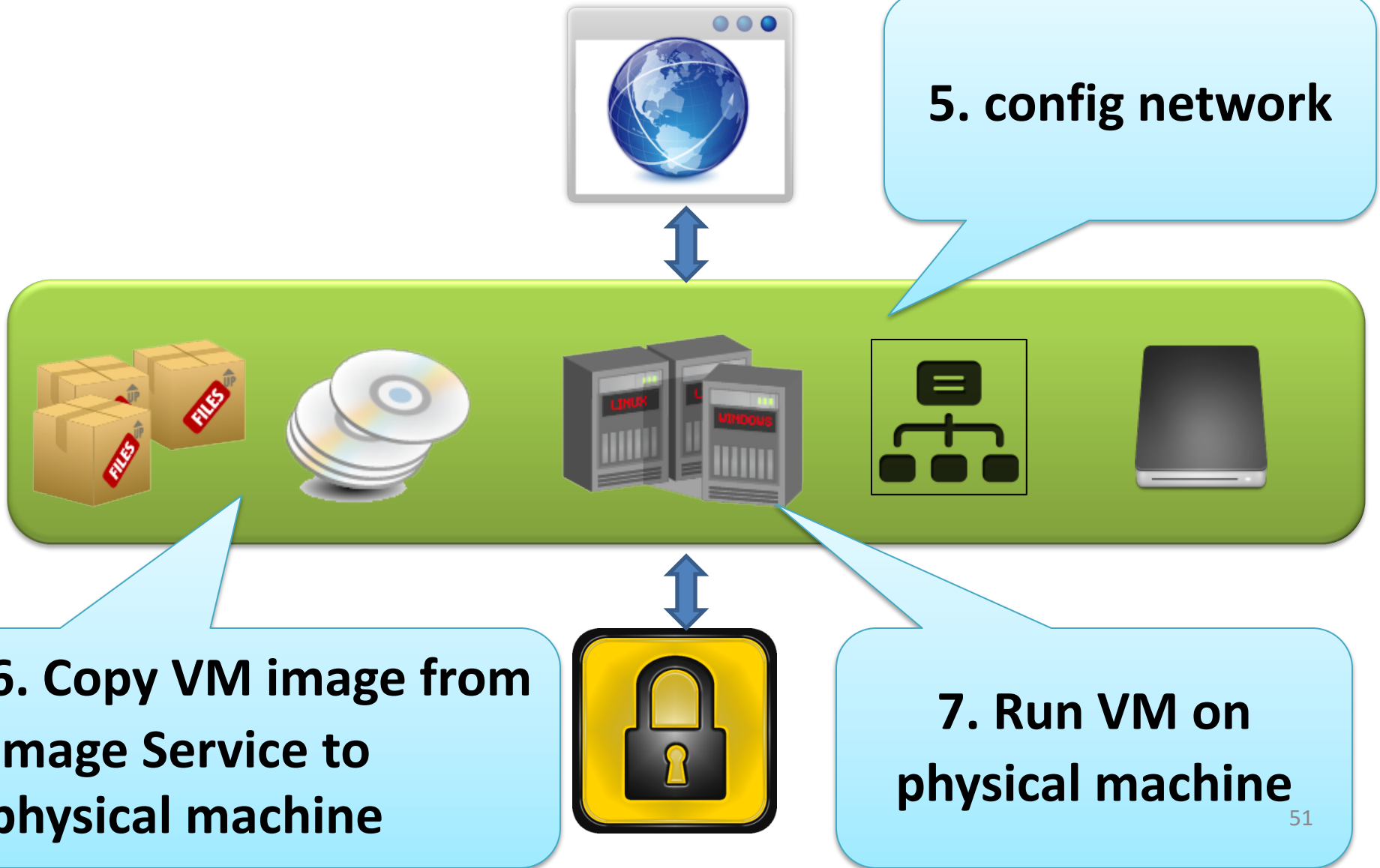
OpenStack Operation

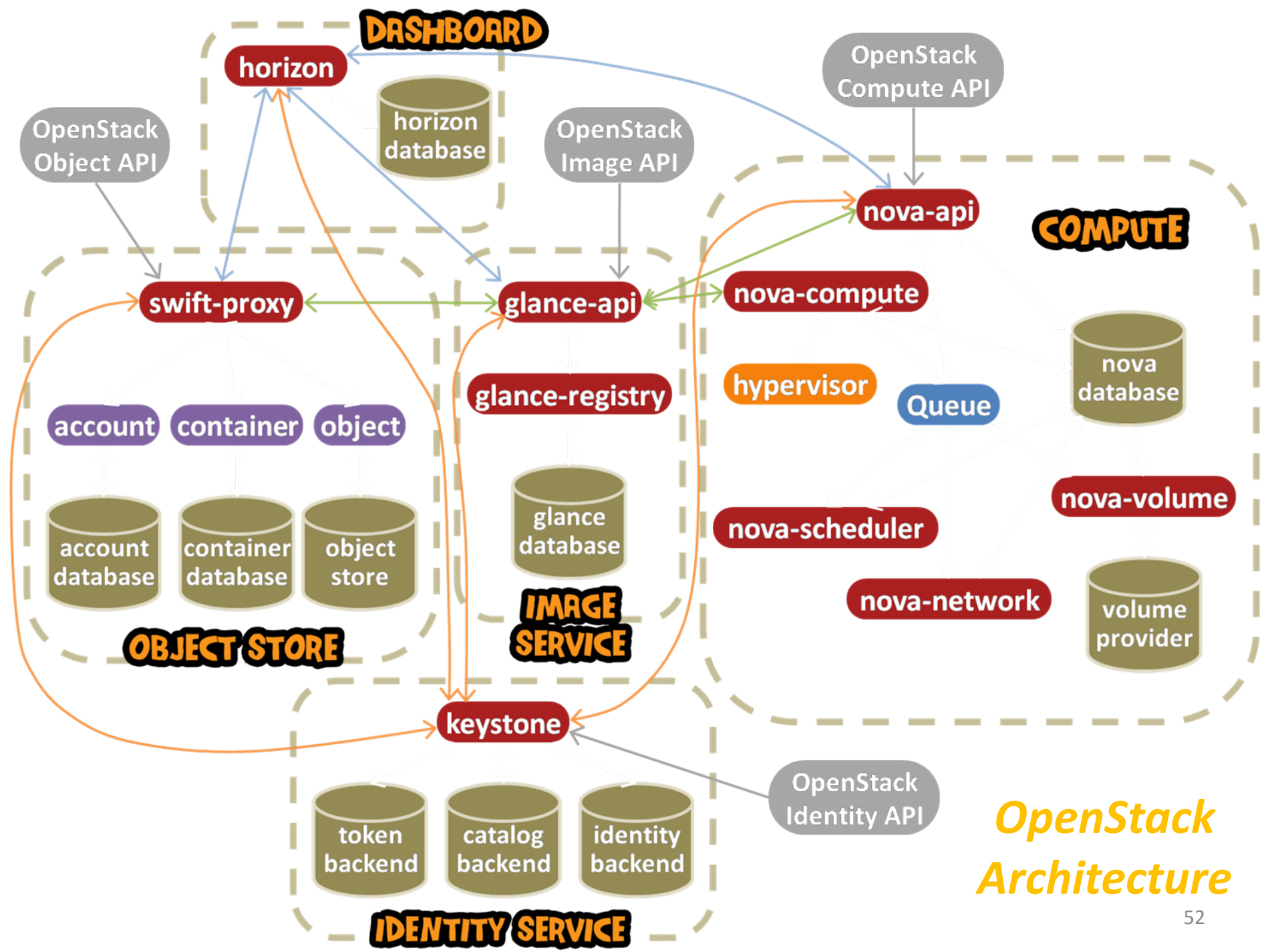


OpenStack Operation



OpenStack Operation





OpenStack Architecture

References

- [HUE] - http://gethue.com/
- [HADOOP] - http://hadoop.apache.org/
- [OOZIE]- http://oozie.apache.org/
- [SOLR]- http://lucene.apache.org/solr/
- [SPARK] - http://spark.apache.org/docs/latest/
- [SPARK_STREAM] - http://spark.apache.org/docs/latest/streaming-programming-guide.html
- [HIVE_WIKI] - https://cwiki.apache.org/confluence/display/Hive/Design
- [BIGTABLE] - *Bigtable: A Distributed Storage System for Structured Data*, Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber - https://research.google.com/archive/bigtable.html
- [HBASE_ARCH] - http://hbase.apache.org/book.html#_architecture
- [OSVIRT_WIKI] - https://en.wikipedia.org/wiki/Operating-system-level_virtualization
- [SYS_PERF] - *Systems Performance: Enterprise and the Cloud*, Brendan Gregg, Prentice Hall, October 2013
- [LXC_WIKI] - https://en.wikipedia.org/wiki/LXC
- [CGRP_WIKI] - https://en.wikipedia.org/wiki/Cgroups
- [NMSP_CGRP] - http://www.netdevconf.org/1.1/proceedings/slides/rosen-namespaces-cgroups-lxc.pdf
- [SPARK] – http://spark.apache.org/docs/latest/
- [SPARK_STRM] - http://spark.apache.org/docs/latest/streaming-programming-guide.html
- [BORG_OMEGA_KUBE] - *Borg, Omega, and Kubernetes, Lessons learned from three container management systems over a decade,* ACMQUEUE Jan-Feb 2016, BRENDAN BURNS, BRIAN GRANT, DAVID OPPENHEIMER, ERIC BREWER, AND JOHN WILKES, GOOGLE INC.
- [KUBE_IO] – https://kubernetes.io
- [OPEN_STACK_CPT] - https://docs.openstack.org/admin-guide/common/get-started-conceptual-architecture.html
- [GOOG_MAP] - *MapReduce: Simplied Data Processing on Large Clusters,* Jeffrey Dean and Sanjay Ghemawat, Google Inc.*, https://research.google.com/archive/mapreduce-osdi04.pdf*
- [DOCKER_LNX_JNL] - *Docker: Lightweight Linux Containers for Consistent Development and Deployment*, http://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment?page=0,1
- [DOCKER_OVERVIEW] - https://docs.docker.com/engine/understanding-docker/
- [CONTAINER_SAAS] - *Towards a container-based architecture for multi-tenant SaaS applications,* Eddy Truyen, Dimitri Van Landuyt, Vincent Reniers, Ansar Rafique, Bert Lagaisse, Wouter Joosen, iMinds-DistriNet, KU Leuven, ARM 2016
- [KUBE_MULTI] - https://github.com/kubernetes/kube-deploy/tree/master/docker-multinode
- [POPEK_GOLDBERG] - Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." *Communications of the ACM* 17.7 (1974): 412-421.