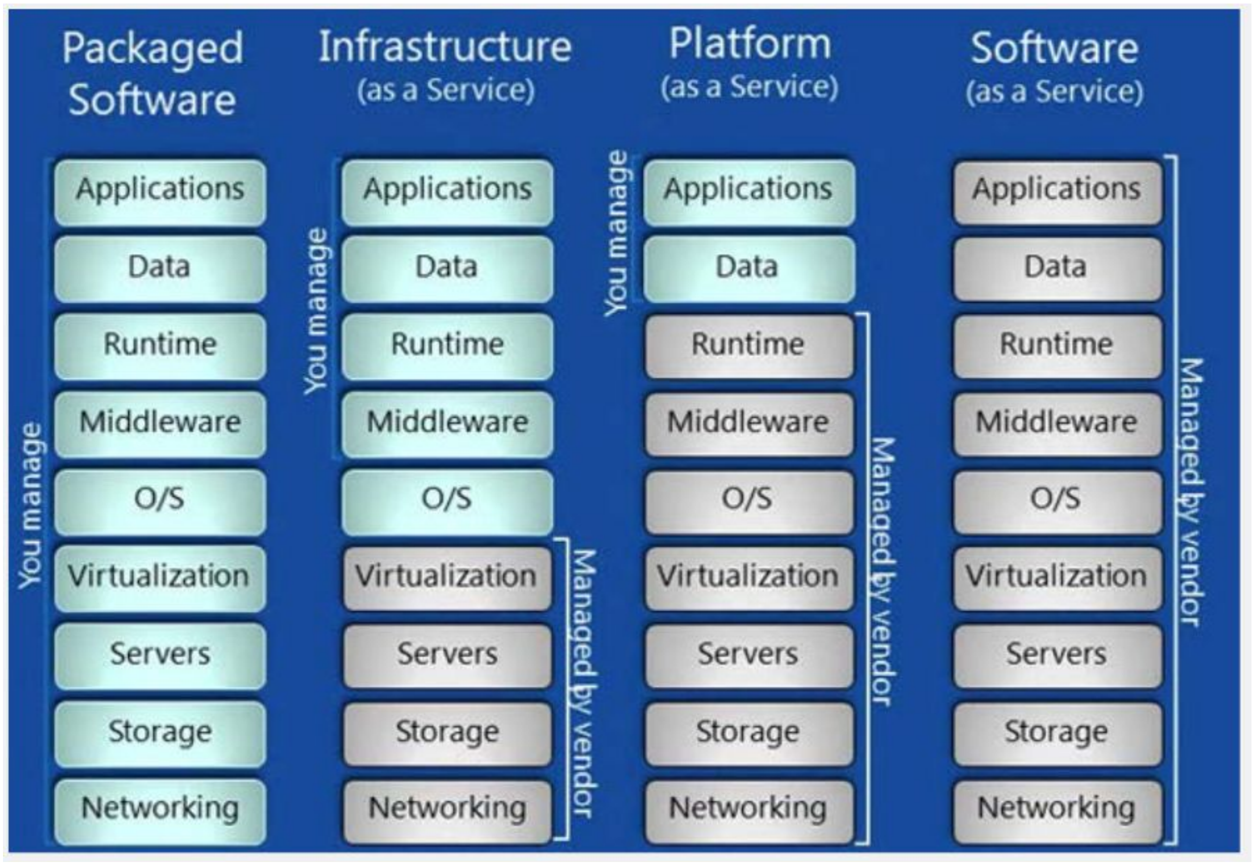
Objective

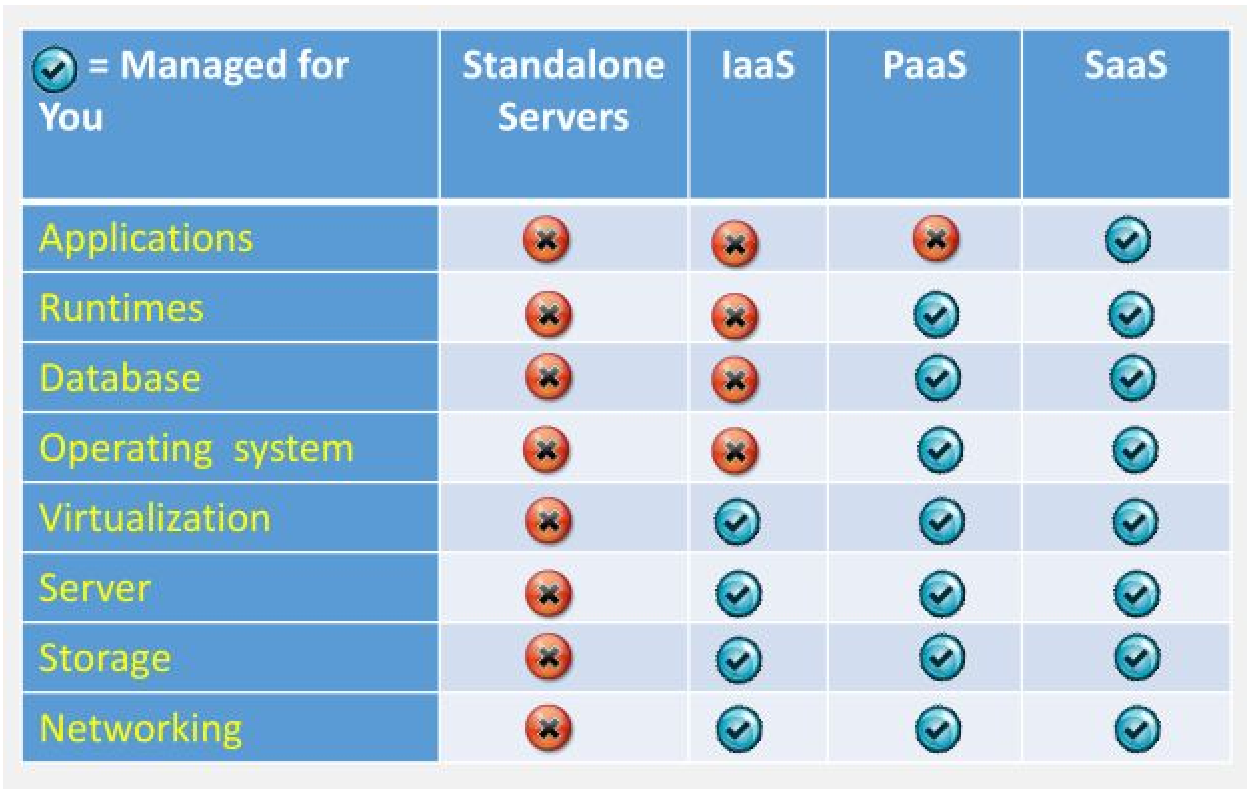
- Next-generation cloud technologies
  Learn about Cloud and container technologies like Docker, CoreOS, Cloud Foundry, Kubernetes and OpenStack, as well as the tooling around them.
- Scalable and performant compute, storage and network solutions
  Get an overview of software defined storage and software defined networking solutions.
- Solutions employed by companies to meet their business demands
- Study up on DevOps and Continuous Integration practices, as well as the deployment tools available to architects to meet and exceed their business goals.

Cloud computing

"Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

- Infrastructure as a Service (IaaS): basic compute and storage resources
  - Rent processing, storage, network
  - Capacity and other fundamental computing resources
  - Provide each user with their own virtual machine,
  - Create a clear separation of resources
  - EC2, S3, DigitalOcean, Linode, Rackspace, Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)
- Platform as a Service (PaaS): cloud application infrastructure
  - Deploy customer-created applications to cloud
  - Offers an application development platform that automatically scale with demand
  - Suitable for multi-tenancy: many users may share the same physical computer and database
  - Risk of “vendor lock-in”: requiring user to develop apps using proprietary interfaces and languages
  - AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos, OpenShift
- Software as a Service (SaaS): cloud applications
  - Use provider’s applications over a network
  - Google Apps, Dropbox, Salesforce, Cisco WebEx, Concur, GoToMeeting





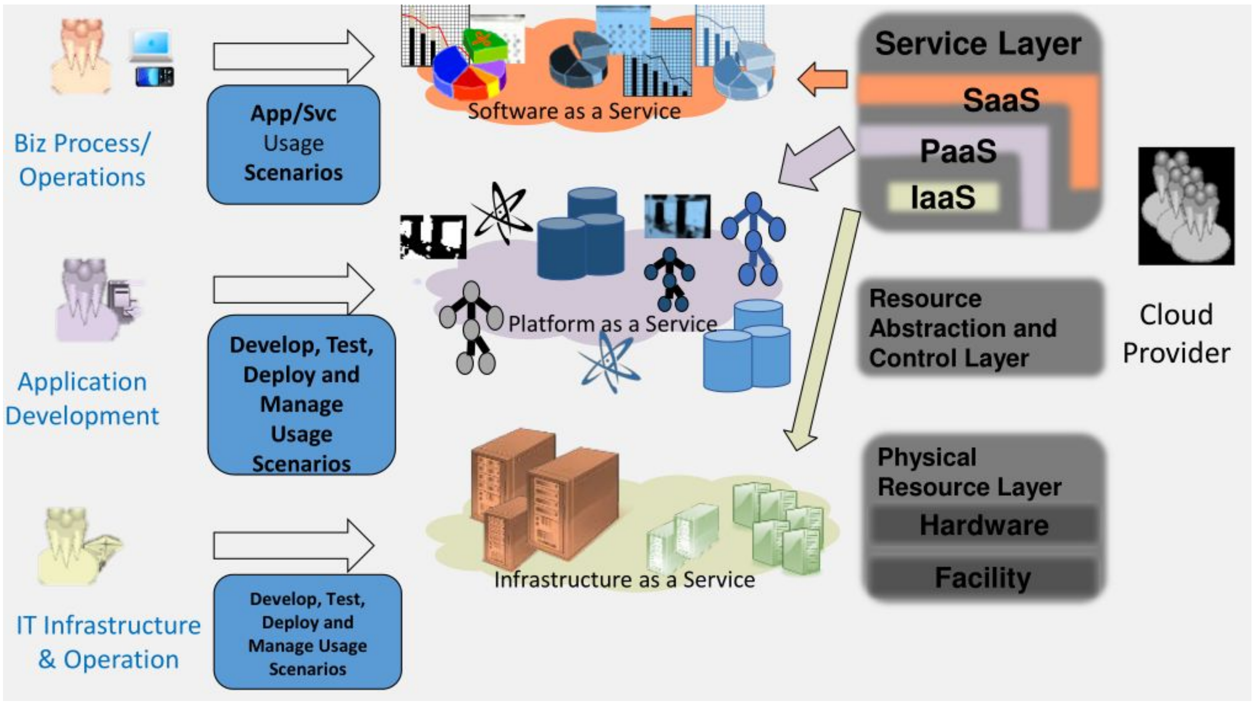
✓ = Managed for You	Standalone Servers	IaaS	PaaS	SaaS
Applications	✗	✗	✗	✓
Runtimes	✗	✗	✓	✓
Database	✗	✗	✓	✓
Operating system	✗	✗	✓	✓
Virtualization	✗	✓	✓	✓
Server	✗	✓	✓	✓
Storage	✗	✓	✓	✓
Networking	✗	✓	✓	✓

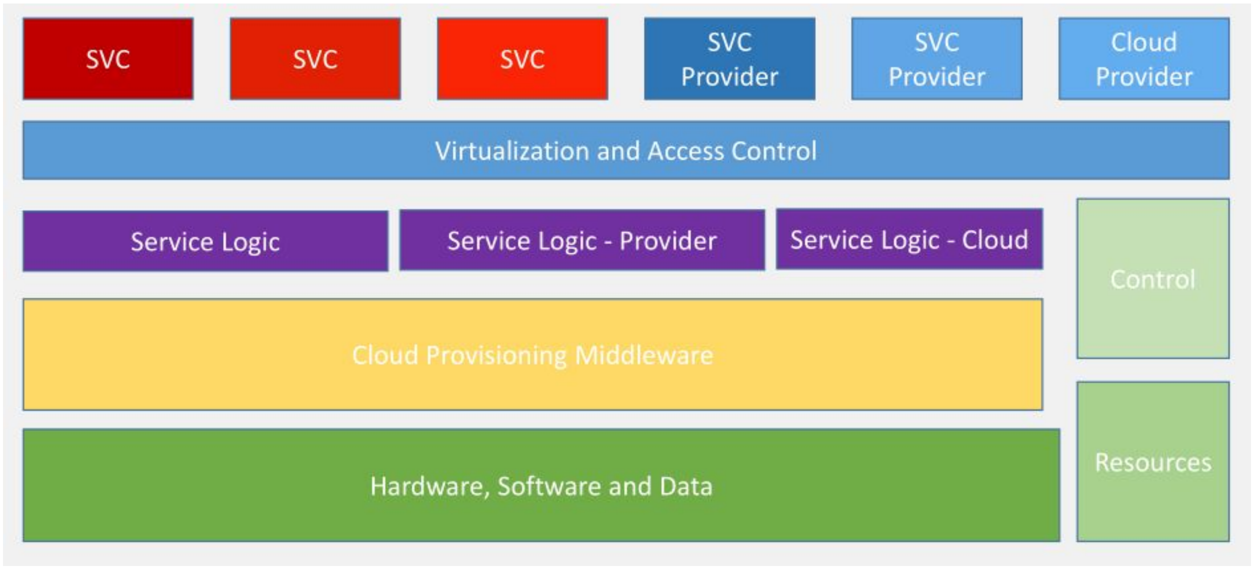
Software defined architecture

- Cloud provides services, service orchestrations, and provisioning
- A cloud may provide IaaS, PaaS, SaaS, and have both internal and external APIs
- The mechanisms and concept of providing services, orchestration and provision is called software defined architecture
- A cloud may contain other software defined entities
    - Software defined network
    - Software defined storage
    - Software defined compute

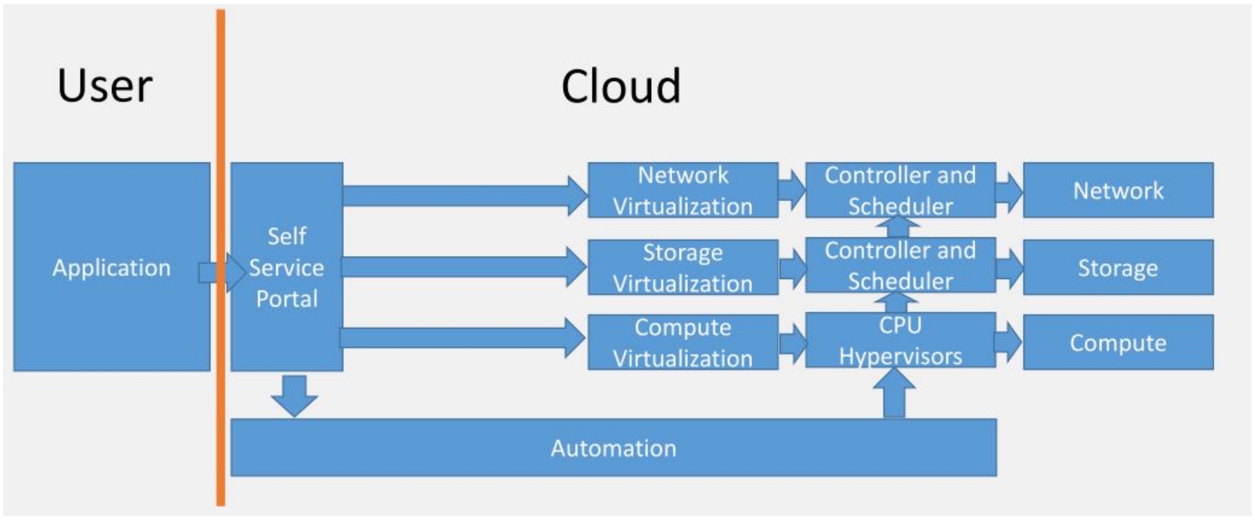
Cloud service orchestration: some mechanism to actually manage those services through the interface into the cloud

- Composing of architecture, tools and processes used by humans to deliver a defined Service
- Stitching of software and hardware components together to deliver a defined Service
- Connecting and automating of workflows when applicable to deliver a defined Service
- Provides: up and down scaling, assurance, billing, workflows


Biz Process/ Operations
App/Svc Usage Scenarios
Software as a Service
Service Layer
SaaS
PaaS
IaaS
Application Development
Develop, Test, Deploy and Manage Usage Scenarios
Platform as a Service
Resource Abstraction and Control Layer
Cloud Provider
IT Infrastructure & Operation
Develop, Test, Deploy and Manage Usage Scenarios
Infrastructure as a Service
Physical Resource Layer
Hardware
Facility



Software defined data center



Provisioning: the signing services to individual machines.

Economical benefits of cloud computing

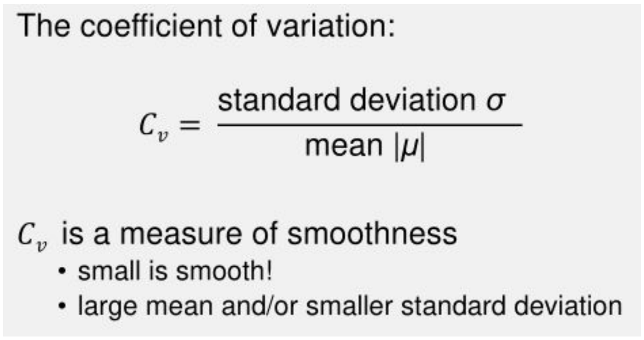
Utility Pricing is good

- when demand varies over time, as is the case of a start-up or a seasonal business
- When Utility Premium is less than ratio of Peak Demand to Average Demand, Cloud computing is beneficial

The Value of Common Infrastructure

- For infrastructure built to peak requirements: Multiplexing demand higher utilization
    - Lower cost per delivered resource than unconsolidated workloads
- For infrastructure built to less than peak: Multiplexing demand: reduce the unserved demand
    - Lower loss of revenue or a Service-Level agreement Space

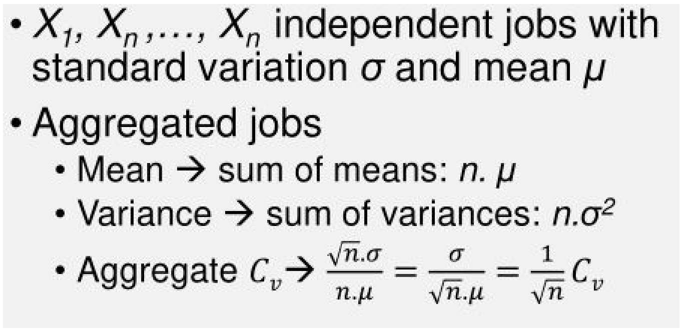
Smoothness

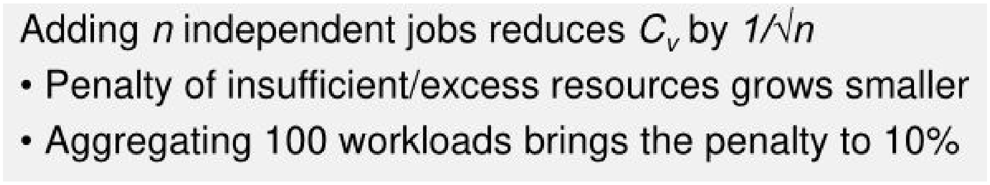

The coefficient of variation:

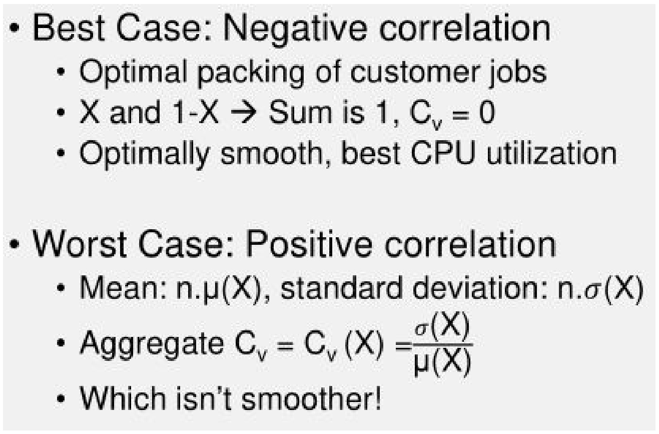
$$C_v = \frac{\text{standard deviation } \sigma}{\text{mean } |\mu|}$$

C_v is a measure of smoothness
- small is smooth!
- large mean and/or smaller standard deviation

A fixed-asset facility servicing highly variable jobs yields low utilization
- Same facility servicing smooth jobs yields high utilization
- Multiplexing jobs with different distributions may reduce the coefficient of variation C_v


- X_1, X_n ,…, X_n independent jobs with standard variation σ and mean μ
- Aggregated jobs
  - Mean → sum of means: $n.\ \mu$
  - Variance → sum of variances: $n.\sigma^2$
  - Aggregate $C_v \rightarrow \frac{\sqrt{n}.\sigma}{n.\mu} = \frac{\sigma}{\sqrt{n}.\mu} = \frac{1}{\sqrt{n}} C_v$


Adding n independent jobs reduces C_v by $1/\sqrt{n}$
- Penalty of insufficient/excess resources grows smaller
- Aggregating 100 workloads brings the penalty to 10%


- Best Case: Negative correlation
  - Optimal packing of customer jobs
  - X and 1-X → Sum is 1, $C_v = 0$
  - Optimally smooth, best CPU utilization
- Worst Case: Positive correlation
  - Mean: n.μ(X), standard deviation: n.σ(X)
  - Aggregate $C_v = C_v(X) = \frac{\sigma(X)}{\mu(X)}$
  - Which isn't smoother!

Negative-correlated jobs
- Private, mid-size, and large-size providers can experience similar statistics of scale

Independent jobs
- Mid-size providers can achieve similar statistical economies to an infinitely large provider

Available data on economy of scale for large providers is mixed
- Use the same COTS computers and components

- Locate near cheap power supplies (everyone can do that)
- Early entrant automation tools (3rd parties take care of it)

MapReduce

It is a core concept in cloud computing over big data.

- Hadoop: Applications that implement it
- PIG: The applications that allow you to take one MapReduce operation and combine it with multiple others
- HDFS: a distributed file store which is reliable, redundant, fault tolerant, and distributed, and we'll be talking about HDFS.
- HBase: Use it with database

Virtualization

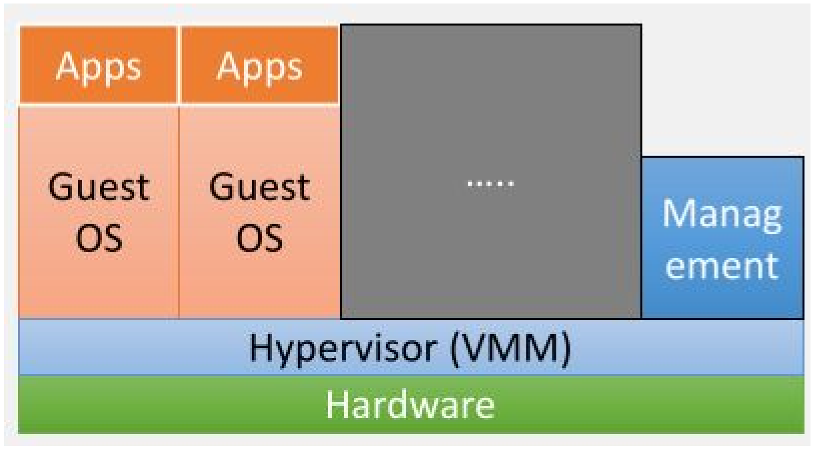
Abstraction is a way to share a physical computer among multiple users.

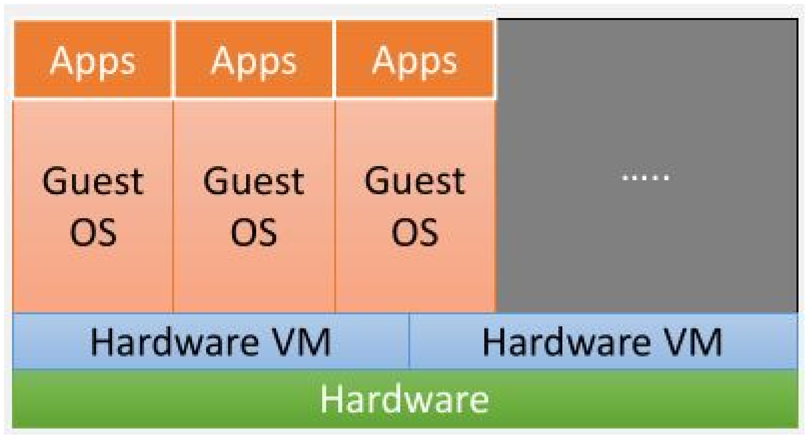
In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, operating systems, storage devices, and computer resources. Virtualization allows distributed computing models without creating dependencies on physical resources.
Clouds are based on virtualization

- Offer services based mainly on virtual machines, remote procedure calls and client/servers
- Provide lots of servers to lots of users

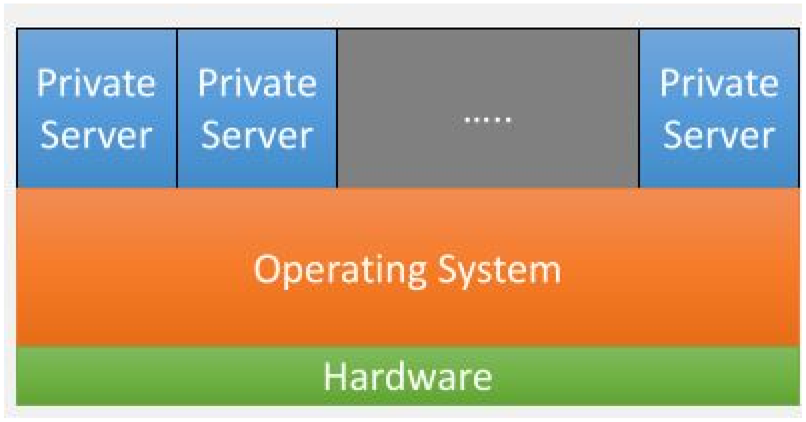
Types of virtualization

- Native / full virtualization:
    - the virtual machine sufficiently simulates hardware to allow an unmodified “guest” OS (one design for the same OS) to be run in isolation.
    - Management software is designed to multiplex those different guest operating systems onto a Hypervisor Virtual Machine Monitor on top of the actual hardware.
    - Example: VirtualBox, virtual PC, VMware, QEMU

- 

- Hardware enabled virtualization:
    - the virtual machine has its own hardware and allows a guest OS to be run in isolation.
    - Needs additional layer in the supervisor state of the computer: Intel VT, AMD-V
    -
    - 

    - Example: VMware Fusion, Parallels Desktop for Mac, Parallels Workstation
- Para-virtualization:
    - the virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by modifying the "guest" OS
    - Example: Xen
- OS-level
    - virtualizing a physical server at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server.

- 

- Example: Containers, Jails, Chroot, Zones, open-VZ

Operating System-Level Virtualization

Hypervisor	**Container**
One real HW, many virtual HWs, many OS'es	One real HW (no virtual HW),
High versatility – can run different OS'es	one kernel, many userspace instances
Lower density, performance, scalability	Higher density, natural page sharing, Dynamic resource allocation
<<Lowers>> are mitigated by new hardware features (such as VT-D)	Native performance: [almost] no overhead

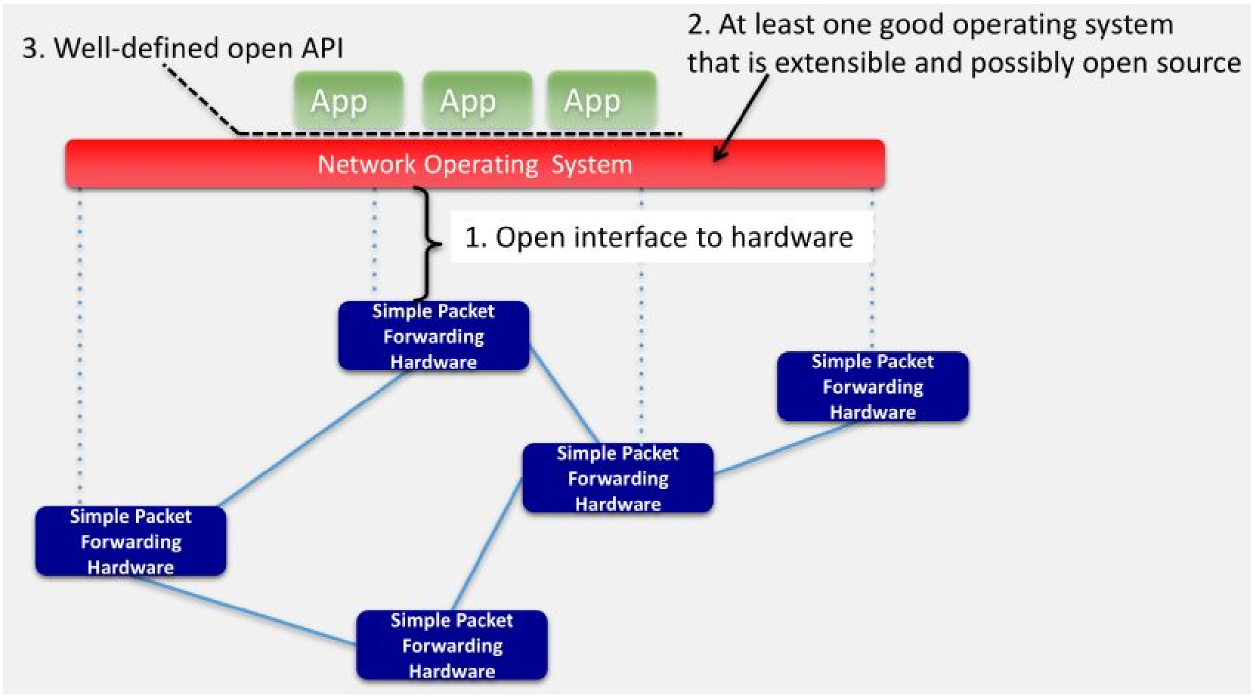
Thinner containers, better performance

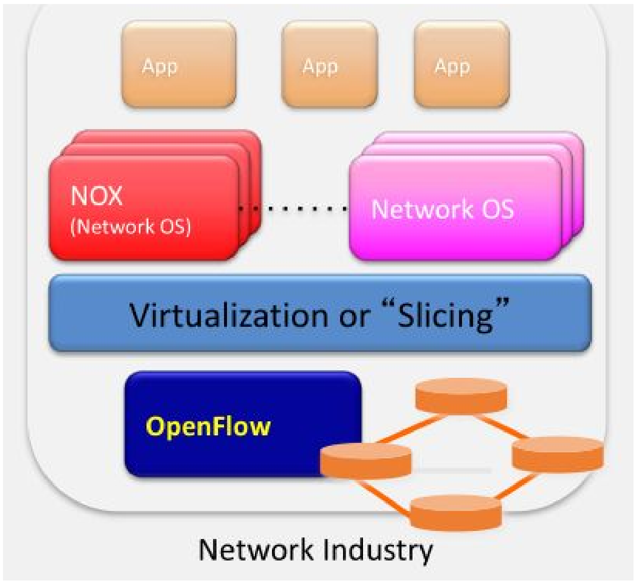
- Containers are more elastic than hypervisor
- Container slicing of the OS is ideally suited to cloud slicing
- Hypervisors' only advantage in IaaS is support for different OS families on one server
- There is a tradeoff between efficiency and isolation

	Hypervisor	container
Multiple kernel	Y	N
Load arbitrary modules	Y	N
Local admin	Y	Y, all
Live migration	Y	Y, OpenVZ

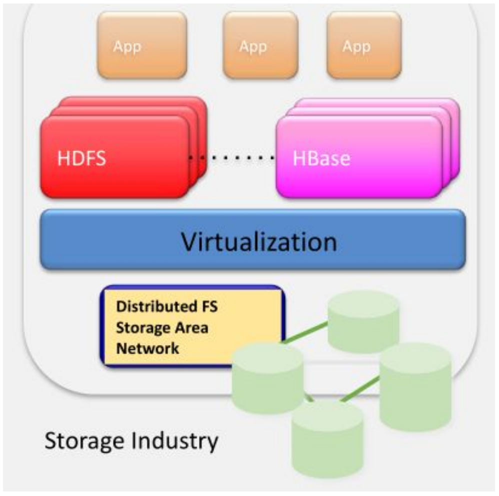
Live system update	N	Y Zap

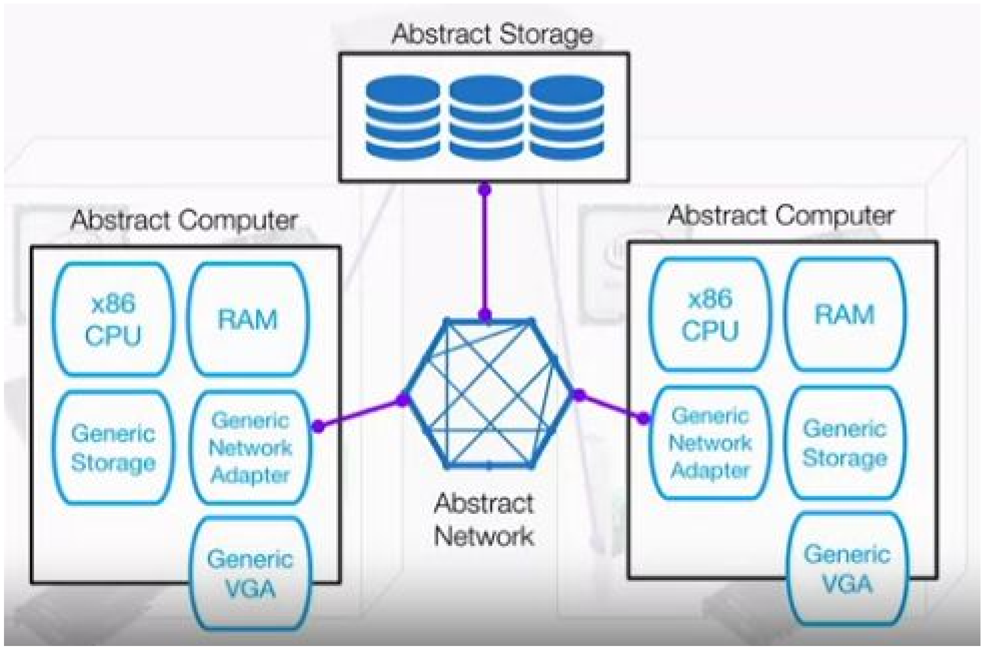
Software defined network


3. Well-defined open API
2. At least one good operating system that is extensible and possibly open source
App
App
App
Network Operating System
1. Open interface to hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware



Virtual storage



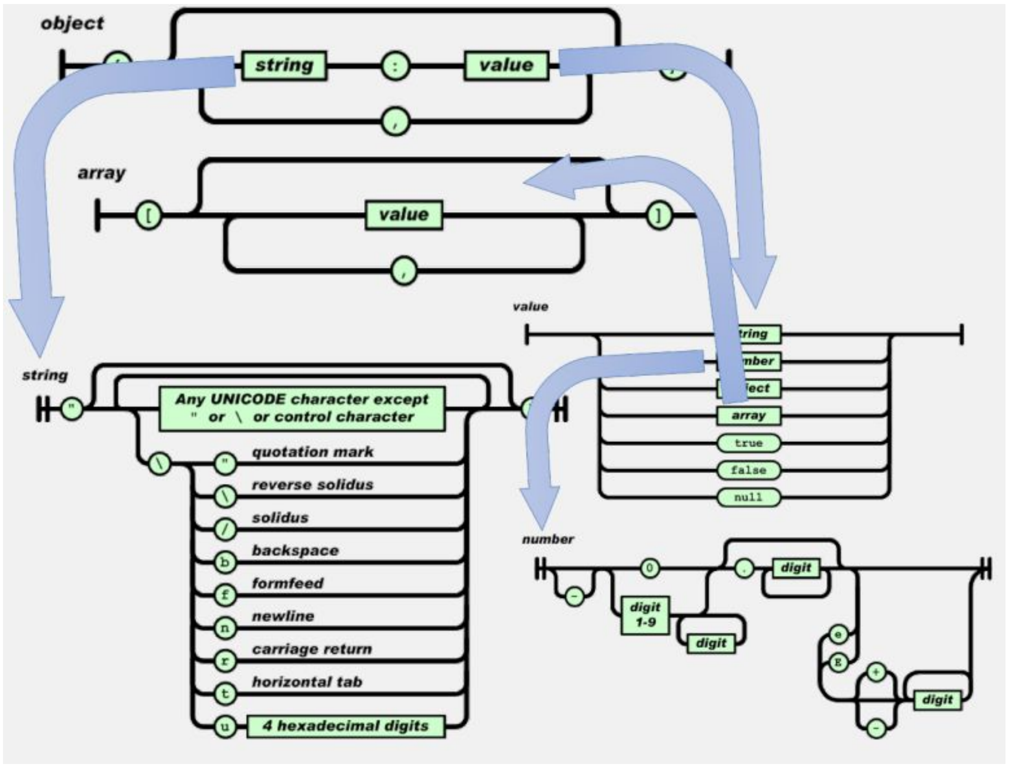


A **hypervisor** or virtual machine monitor (VMM) is computer software, firmware or hardware that creates and runs virtual machines. With Hypervisors, we emulate hardware like CPU, Disk, Network, Memory and install Guest Machines on it. We can create multiple Guest Machines with different Operating Systems on a Hypervisor. Examples are KVM, Xen, VMWare, VirtualBox, Hyper-V.

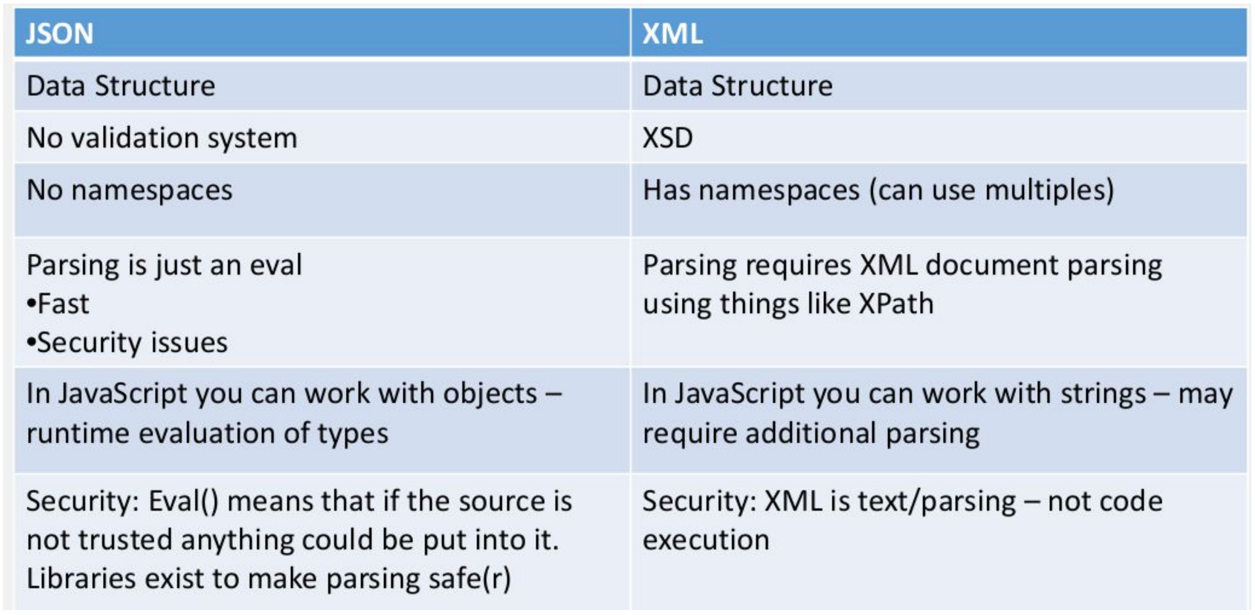
In computing, an **emulator** is hardware or software that enables one computer system (called the host) to behave like another computer system (called the guest).

JSON (JavaScript Object Notation) is a lightweight data interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate." – JSON.org
More importantly: JSON is a subset of JavaScript

JSON structure



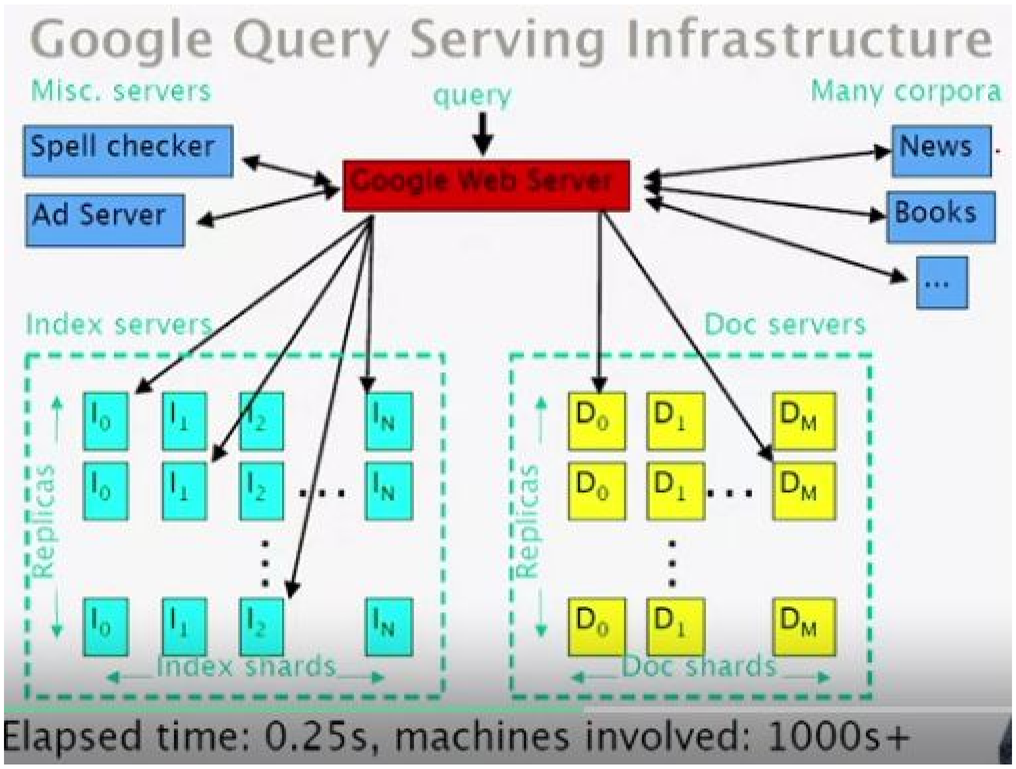
JSON vs XML



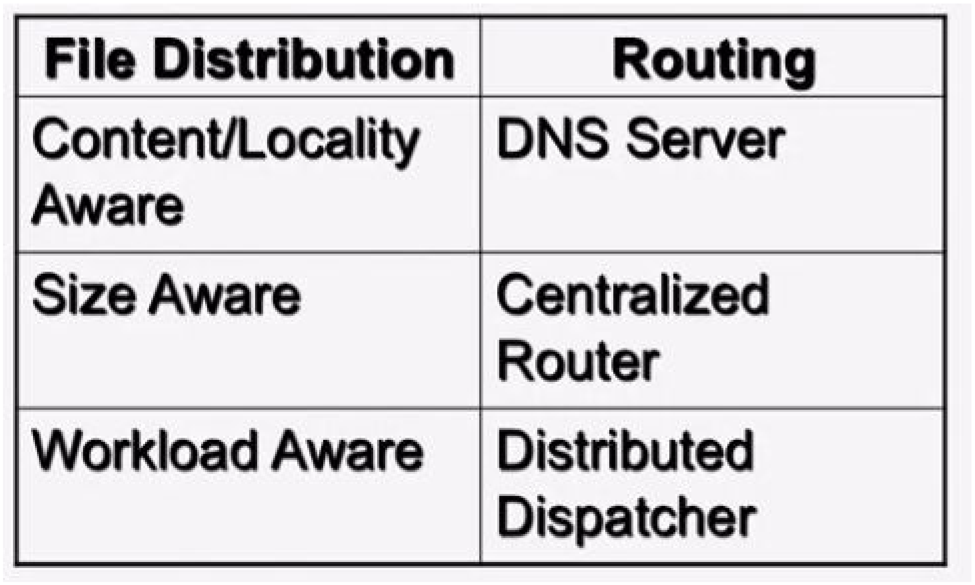
JSON	XML
Data Structure	Data Structure
No validation system	XSD
No namespaces	Has namespaces (can use multiples)
Parsing is just an eval •Fast •Security issues	Parsing requires XML document parsing using things like XPath
In JavaScript you can work with objects – runtime evaluation of types	In JavaScript you can work with strings – may require additional parsing
Security: Eval() means that if the source is not trusted anything could be put into it. Libraries exist to make parsing safe(r)	Security: XML is text/parsing – not code execution

Load balancing

In computing, load balancing refers to the process of distributing a set of tasks over a set of resources (computing units), with the aim of making their overall processing more efficient. Load balancing techniques can optimize the response time for each task, avoiding unevenly overloading compute nodes while other compute nodes are left idle.



Load balancing approaches



File Distribution	Routing
Content/Locality Aware	DNS Server
Size Aware	Centralized Router
Workload Aware	Distributed Dispatcher

Issues

Efficiently processing requests with optimizations for load balancing

- Send and process requests to a web server that has files in cache
- Send and process requests to a web server with the least amount of requests

- Send and process request to a web server determined by the size of the request

Server load balancer

- Gets user to needed resource
    - Server must be available
    - User's "session" must not be broken
        - If user must get to the same resource over and over, the SLB device must ensure that happens (i.e., session persistence)
- In order to do work, SLB must
    - Know servers – IP / port, availability
    - Understand details of some protocols (e.g., FTP, SIP)
- Network Address Translation (NAT)
    - Packets are rewritten as they pass through the SLB device

Reasons to load-balance

- Scale applications / services
- Ease of administration / maintenance
    - Easily and transparently remove physical servers from rotation in order to perform any type of maintenance on that server
- Resource sharing
    - Can run multiple instances of an application / service on a server; could be running on a different port for each instance; can Do load-balance to different port based on data analyzed

SLB Algorithms

Most predominant

- Least connections: Server with fewest number of flows gets the new flow request
- Weighted least connections: Associate a weight / strength for each server and distribute load across server farm based on the weights of all servers in the farm
- Round robin: Round robin through the servers in server farm
- Weighted round robin: Give each server "weight" number of flows in a row; weight is set just like it is in weighted least flows

How SLB Devices Make Decisions

The SLB device can make its load-balancing decisions based on several factors

- Some of these factors can be obtained from the packet headers (i.e., IP address, port numbers)
- Other factors are obtained by looking at the data beyond the network headers. Examples:
    - HTTP cookies
    - HTTP URLs
    - SSL client certificates

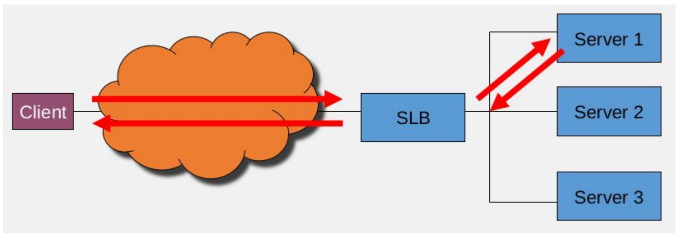
- The decisions can be based strictly on flow counts, or they can be used on knowledge of application
- For some protocols, like FTP, you must have knowledge of protocol to correctly load-balance (i.e., control and data connection must go to same physical server)

When a new flow arrives
Determine whether virtual server exists

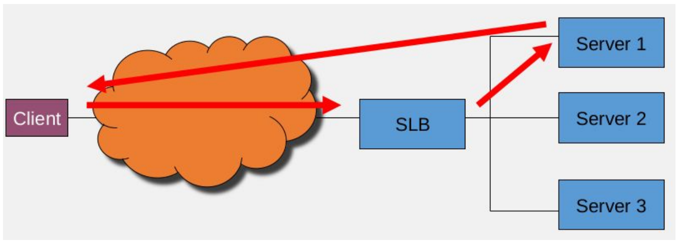
- If so, make sure virtual server has available resources
- If so, then determine level of service needed by that client to that virtual server
    - If virtual machine is configured with particular type of protocol support of session persistence, then do that work
- Pick a real server for that client
    - The determination of real server is based on flow counts and information about the flow
    - In order to do this, the SLB may need to proxy the flow to get all necessary information for determining the real server; this will be based on the services configured for that virtual server
- If not, the packet is bridged to the correct interface based on Layer 2

SLB architecture

- Traditional
    - SLB device sits between the Clients and the Servers being load-balanced
- Distributed
    - SLB device sits off to the side and only receives the packets it needs to, based on flow setup and teardown



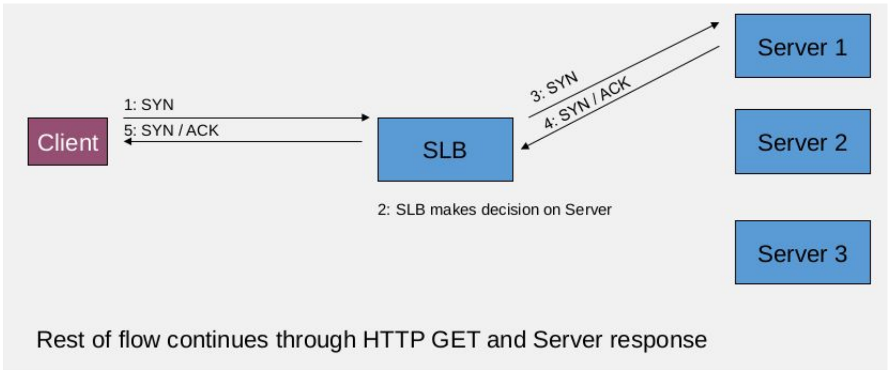
Traditional view with NAT



Traditional view without NAT

Load-balance: layer 3/4

- Look at the destination IP address and port to make a load-balancing decision
- In order to do that, you can determine a real server based on the first packet that arrives

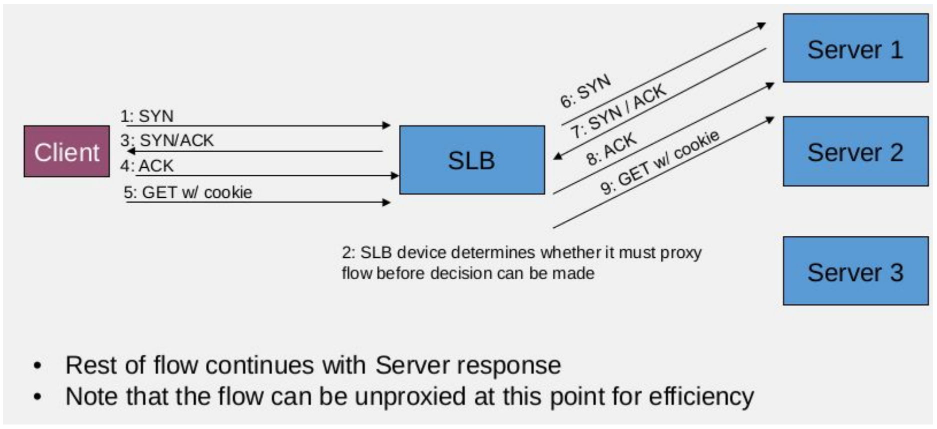

Server 1
Server 2
Server 3
Client
1: SYN
5: SYN / ACK
SLB
3: SYN
4: SYN / ACK
2: SLB makes decision on Server
Rest of flow continues through HTTP GET and Server response

Layer 5

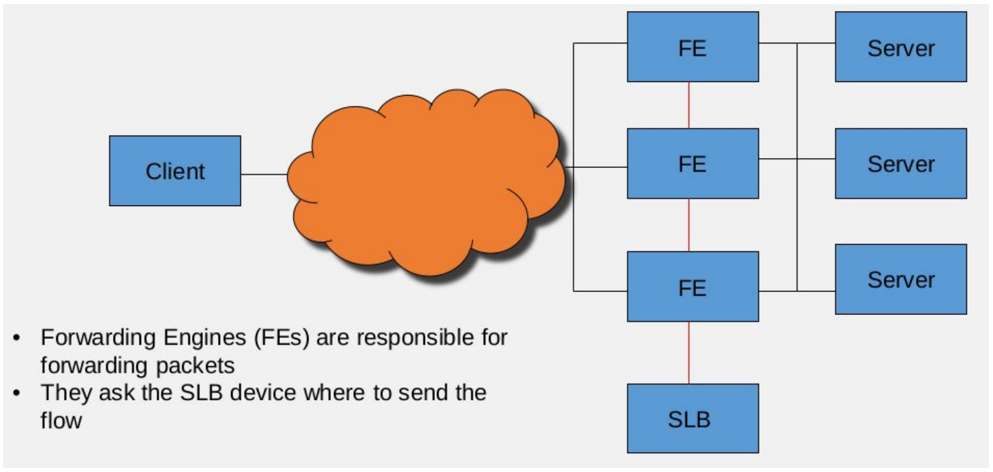
The SLB device must terminate the TCP flow for an amount of time before the SLB decision can be made

- E.g. the cookie value must be sent by the client, which is after the TCP handshake before determining the real server.

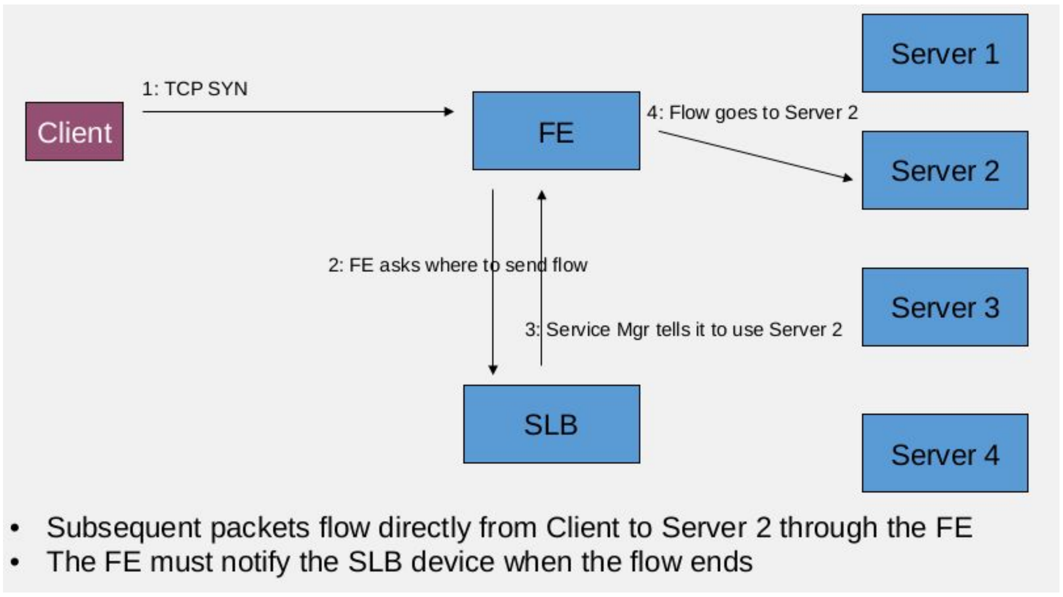
Layer 5+: sample flow


Server 1
Server 2
Server 3
Client
1: SYN
3: SYN/ACK
4: ACK
5: GET w/ cookie
SLB
6: SYN
7: SYN / ACK
8: ACK
9: GET w/ cookie
2: SLB device determines whether it must proxy flow before decision can be made
Rest of flow continues with Server response
Note that the flow can be unproxied at this point for efficiency

Distributed Architecture



- Forwarding Engines (FEs) are responsible for forwarding packets
- They ask the SLB device where to send the flow



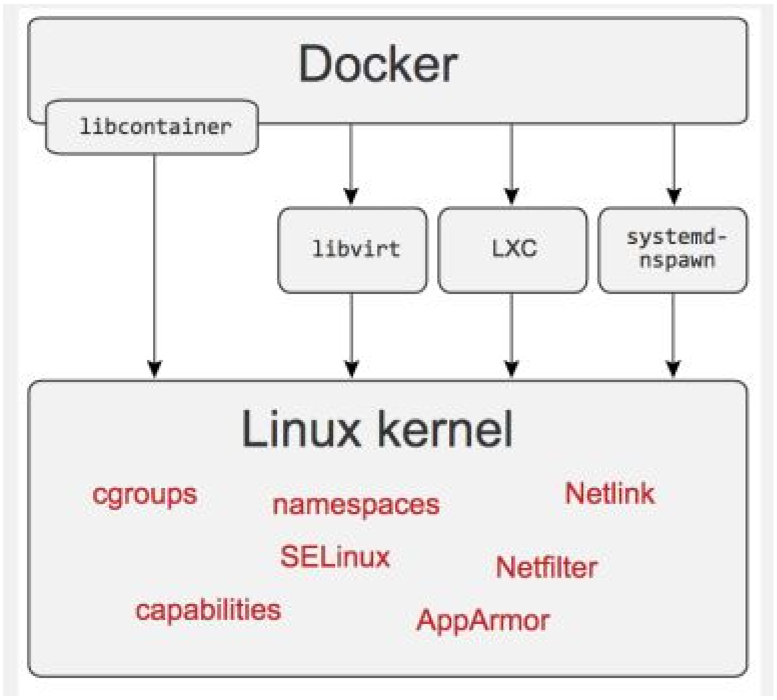
- Subsequent packets flow directly from Client to Server 2 through the FE
- The FE must notify the SLB device when the flow ends

Docker

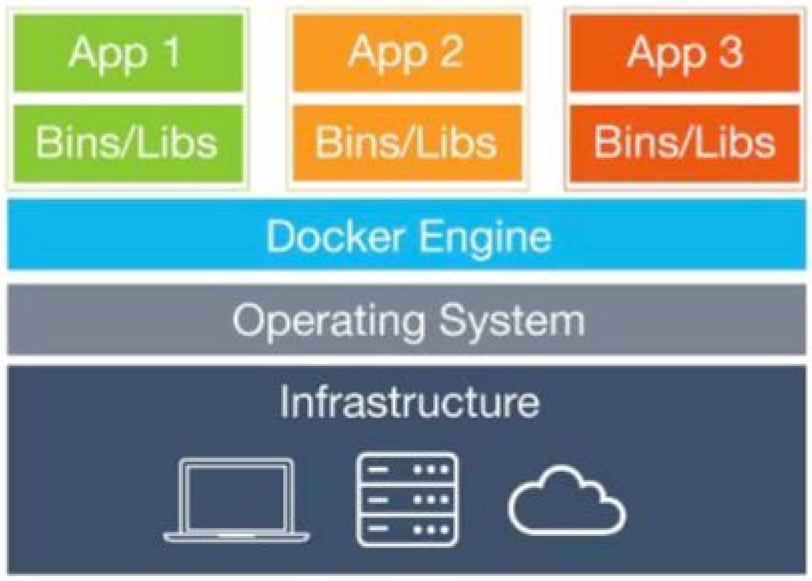
Docker containers wrap up a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries –
anything you can install on a server. This guarantees that the software will always run the same, regardless of its environment it is running in.

- Docker automates the deployment of applications insides software containers
- Additional layer of abstraction and automation of operation system - level virtualization of Linux.

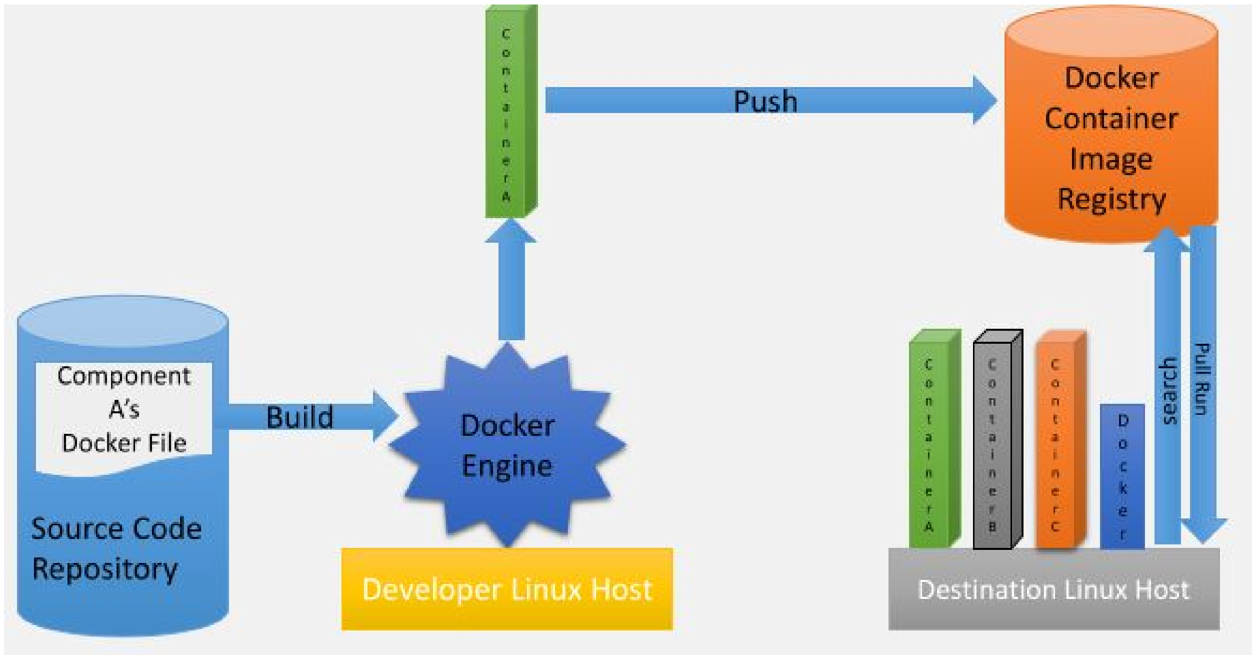
Docker structure



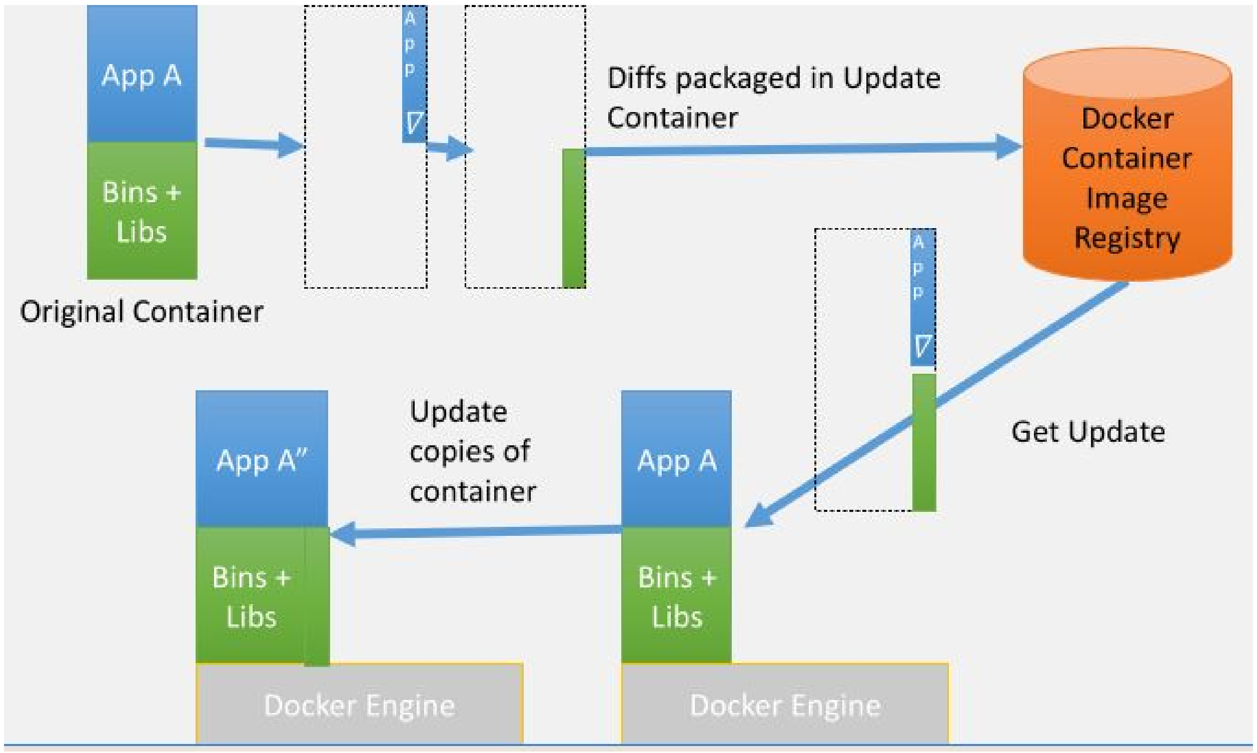
Container built with Docker



Basics of Docker



Changes and updates



When you want to update the docker image, so what you do is to pull down the update, the difference in the apps, difference in the libraries. And the Docker engine applies those engines to give you the new version of your system.

Kubernetes

Kubernetes provides a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts“.

Parts

Loosely coupled building blocks

- Pods
- Unique IP address of a collection of colocated containers
- No collisions on IP port address
- Volume (network disk or local directory) shared by containers in a pod

Labels and Selectors

- Key-value pair (label) that resolves (selector) to a pod or node component or container

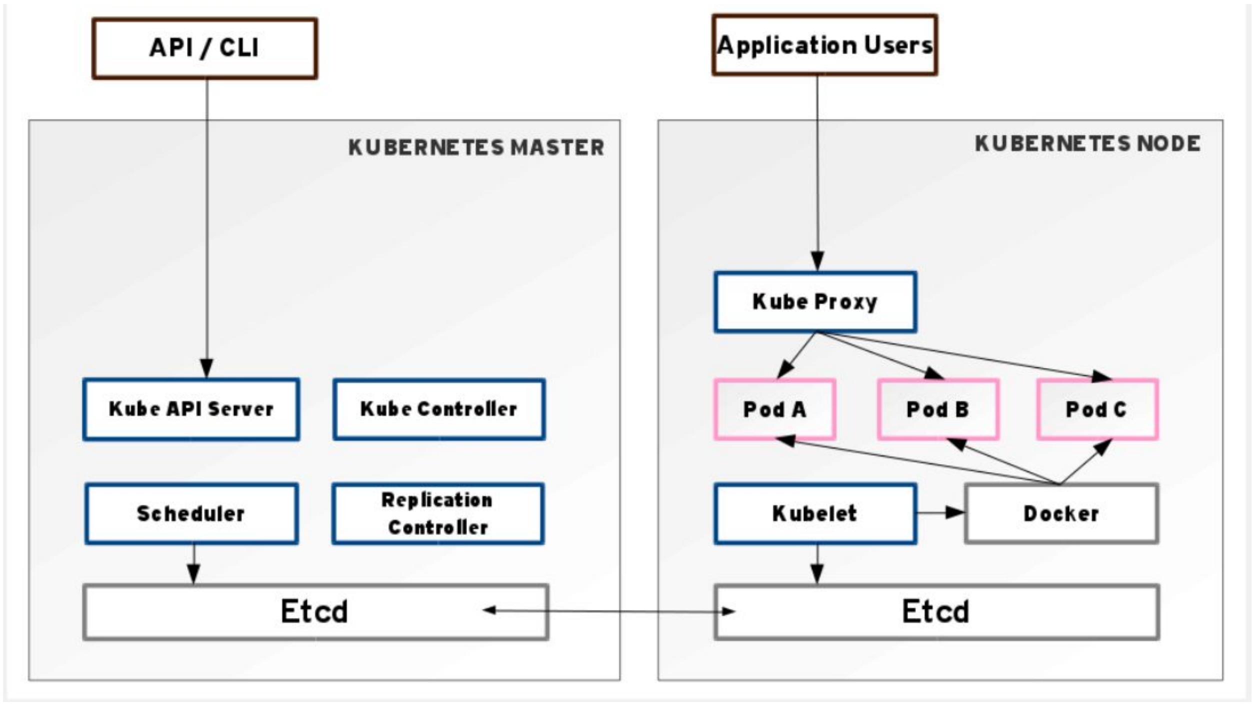
Controllers

- Manages pods: Replication Controller (replacement), Daemon Set Controller (keeps one pod on every machine in a set running), Job Controller (runs “batch” pods to completion)

Services

- Set of pods working together
- Label selector
- Service discovery and request routing – stable IP address and DNS name
- Round Robin load balancer to network IP address

Architecture

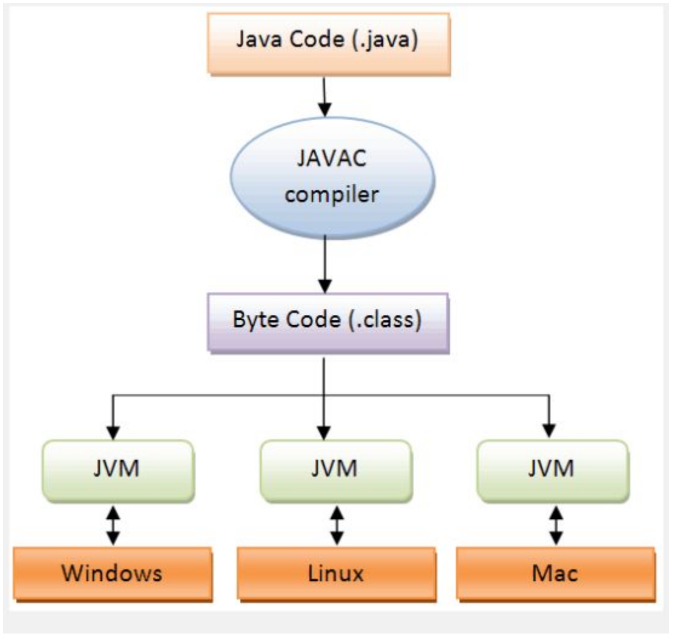


Java virtual machine (JVM)

- Abstract computing machine that enables a computer to run a Java program.
    - Specification: document formally describing a JVM implementation. Having a single specification ensures all implementations are interoperable.
    - Implementation: a computer program meeting the JVM specification requirements.
    - Instance: an implementation running in a process that executes a computer program compiled into Java bytecode.

Java Runtime Environment (JRE) is a software package that contains what is required to run a Java program:

- Java Virtual Machine implementation
- Java Class Library

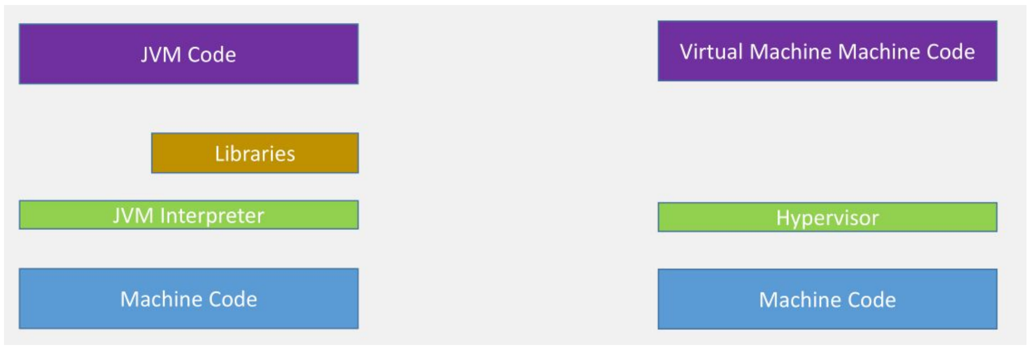


Bytecode verifier

- Branches are always to valid locations
- Data is always initialized and references are always type-safe
- Access to private or package private data and methods is rigidly controlled (at run time on first access.)

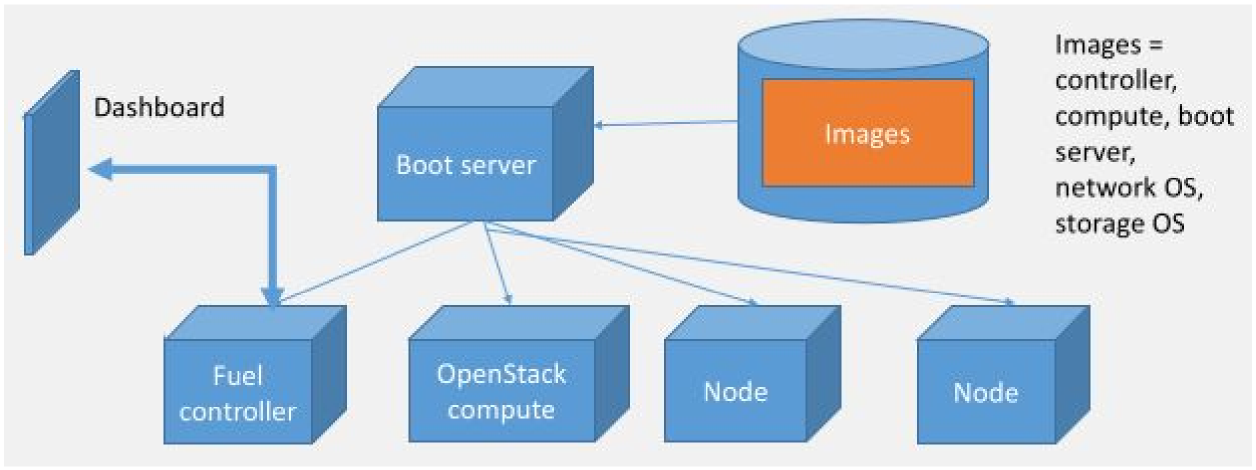
No user program can crash the host machine or otherwise interfere inappropriately with other operations on the host machine
Just-in-time (JIT) compilers can translate much JVM into machine code.
Operates as execution occurs and improves speed.

Comparison with classic VM

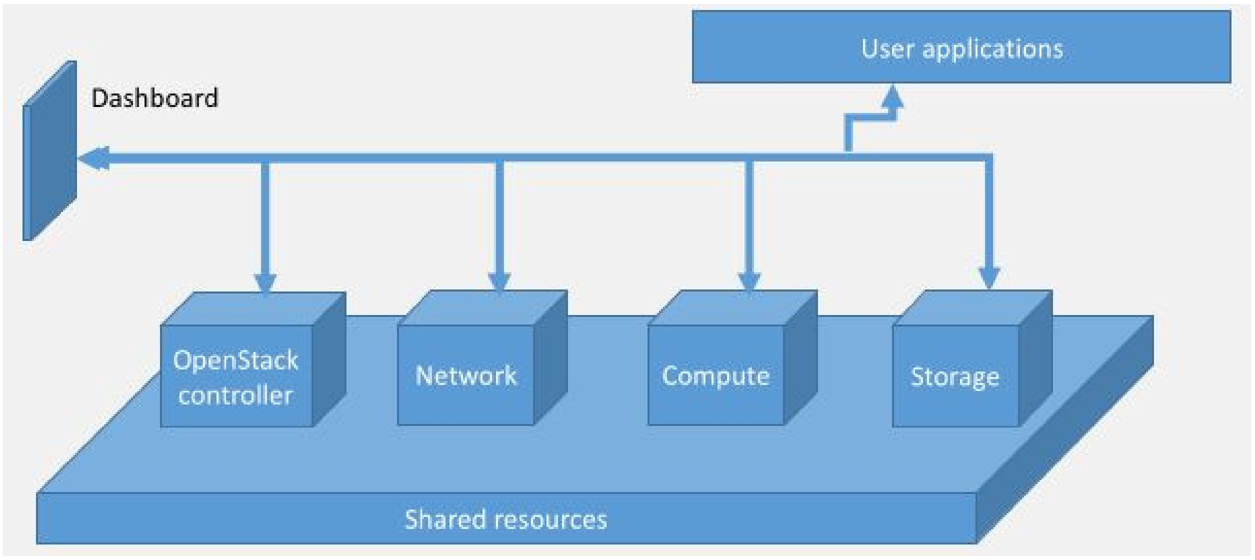


OpenStack

Mirantis Fuel Architecture



OpenStack Architecture



With OpenStack, we can offer a cloud computing platform for public and private clouds. OpenStack was started as a joint project between Rackspace and NASA in 2010. In 2012, a non-profit corporate entity, called the OpenStack Foundation, was formed and it is managing it since then. It is now supported by more than 500 organizations. OpenStack is an open source software platform, which is released under an Apache 2.0 License.

Some of the major OpenStack components are:

- Keystone: Provides Identity, Token, Catalog, and Policy services to projects.
- Nova: Provides on-demand compute resources.
    - Orchestrates large network for virtual machines (VMs)
    - Responsible for VM instance lifecycle, network management, and user
- access control
- Horizon: Provides the Dashboard, which is a web-based user interface to manage the OpenStack service.
- Neutron: Implements the network as a service and provides network capabilities to different OpenStack components.
- Glance: Image service
    - Provides a service where users can upload and discover data assets, like images and metadata.
    - Manages VM disk images
    - Can be a stand-alone service
    - Supports private/public permissions, and can handle a variety of disk
    - image formats
- Swift: Object Storage
    - Provides a highly available, distributed, eventually consistent object/blob store.

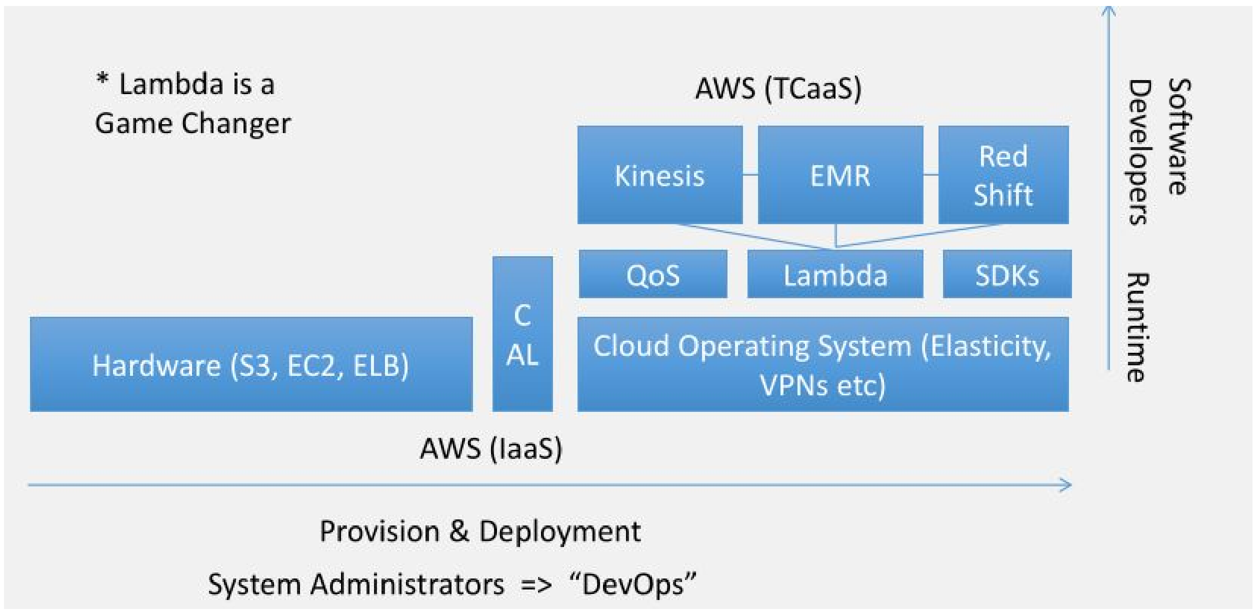
    - Provides scalable, redundant, long-term storage images, data archives, and multimedia
    -
- Cinder: Provides block storage as a service.
- Heat: Provides a service to orchestrate composite cloud applications, using a declarative template format through an OpenStack-native REST API.
- Ceilometer: It is part of the Telemetry project and provides data collection services for billing and other purposes.

Each of the **OpenStack** components is also modular by design. For example, with Nova we can select an underneath hypervisor depending on the requirement, which can be either libvirt (qemu/KVM), Hyper-V, VMware, XenServer, Xen via libvirt.

Some of the benefits of using OpenStack are:
- It is an open source solution.
- It is a cloud computing platform for public and private clouds.
- It offers a flexible, customizable, vendor-neutral environment.
- It provides a high level of security.
- It facilitates automation throughout the stages of the cloud lifecycle.
- By reducing system management overhead and avoiding vendor lock-in, it can be cost-effective.

Serverless architecture

- "Applications where some amount of server-side logic is still written by the application developer but unlike traditional architectures is run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a 3rd party"
- 'Functions as a service / FaaS'



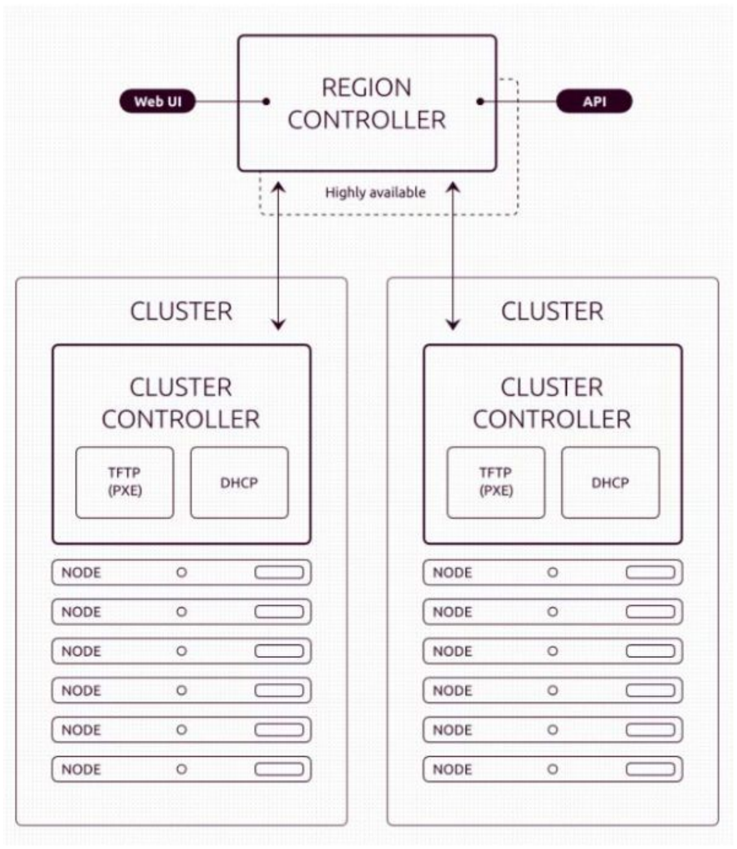
MAAS (Metal as a Service)

Metal as a Service (MaaS) combines the scalability and flexibility of the cloud with the ability to harness the power of physical servers

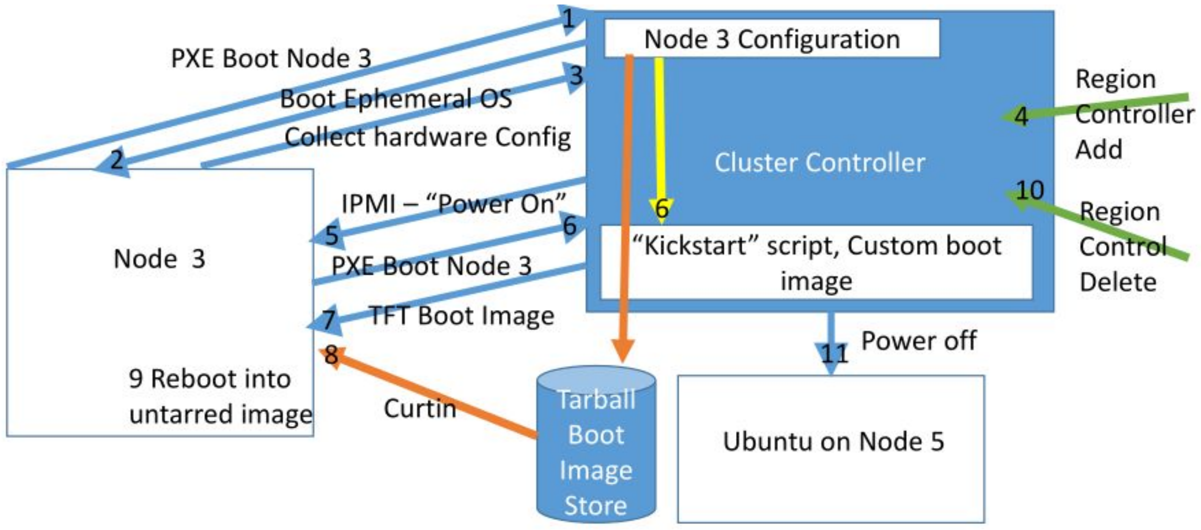
Use cases

- Give me a new server machine
- Install my chosen operating system
- Configure the drives like this
- Configure the network like this
- Use these credentials
- Don't need the server anymore

Components



MaaS Sequence of events: discovery/enlisting, enrolling, commissioning, deploying
Example process:

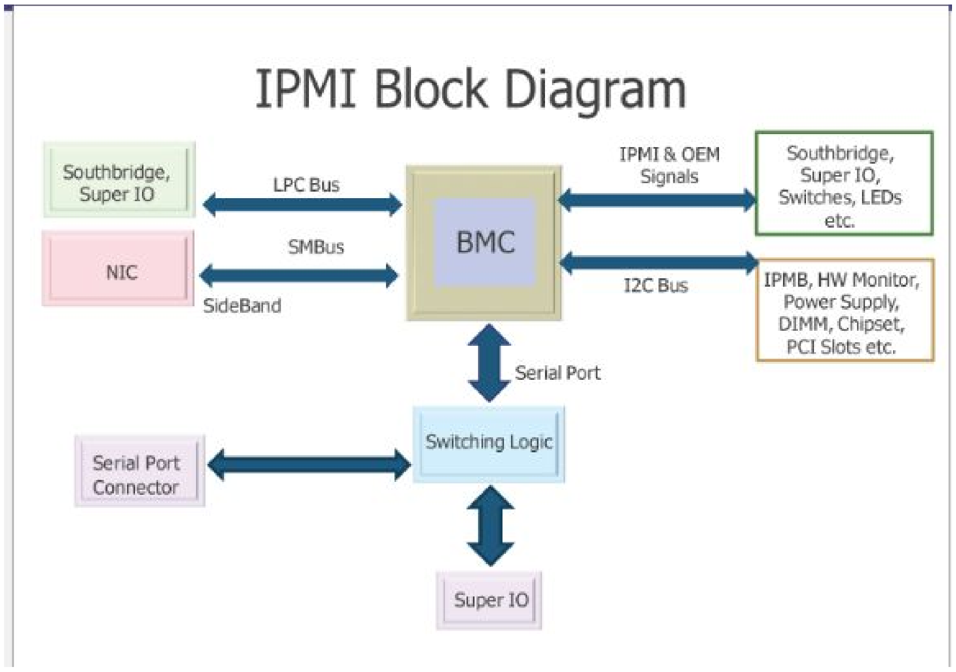


MaaS usage

- Pool of nodes
- Nodes can be allocated to users with user authentication credentials for remote access using queries like memory size and numbers of cores
- Users can choose the operating system and application to run on a node(s)
- When finished with nodes, they can be returned Space to reserved

Juju is an open source application modeling tool developed by Canonical Ltd. Juju focuses on reducing the operation overhead of today's software by facilitating quickly deploying, configuring, scaling, integrating, and performing operational tasks on a wide choice of public and private cloud services along with bare metal servers and local container based deployments.

Intelligent Platform Management Interface (IPMI) hardware



IPMI usage

- before an OS has booted (allowing, for example, the remote monitoring or changing of BIOS settings)
- when the system is powered down
- after OS or system failure – the key characteristic of IPMI compared with in-band system management such as by remote login to the operating system using SSH

Platform as a Service (PaaS)

Google App Engine (GAE)

- It supports multi-tenancy and offers automatic scaling for web applications
- It supports Python, Java, and Go

GAE fameworks and tools

- GAE supports Django web framework and the Grails web app framework
- GAE provides infrastructure tools that enable users to deploy code without worrying about infrastructure challenges such as deployment, failover, or scalability

- However, the GAE infrastructure limits the type of can be run

Security, sandbox

- Applications run in a secure environment
- Isolates applications from hardware and operating system, and imposes security limitations
- For example, application code only runs in response to Space reserved for video requests, and a request handler cannot spawn potentially malicious sub-processes after response has been sent

Storage

- Users of GAE can use App Engine Datastore, Google Cloud SQL, and Google Cloud Storage
- Users can also harness Google's database technology, such as Bigtable

Example uses

- Can take advantage of Google's single sign on feature when users want to access their Gmail or Google docs
- Build Chrome and Android games on GAE
- Google Cloud Endpoints to use / access mobile services
- **Other services**: App engine Map Reduce, Search API, SSL support, Page speed, XMPP API, Memcache API

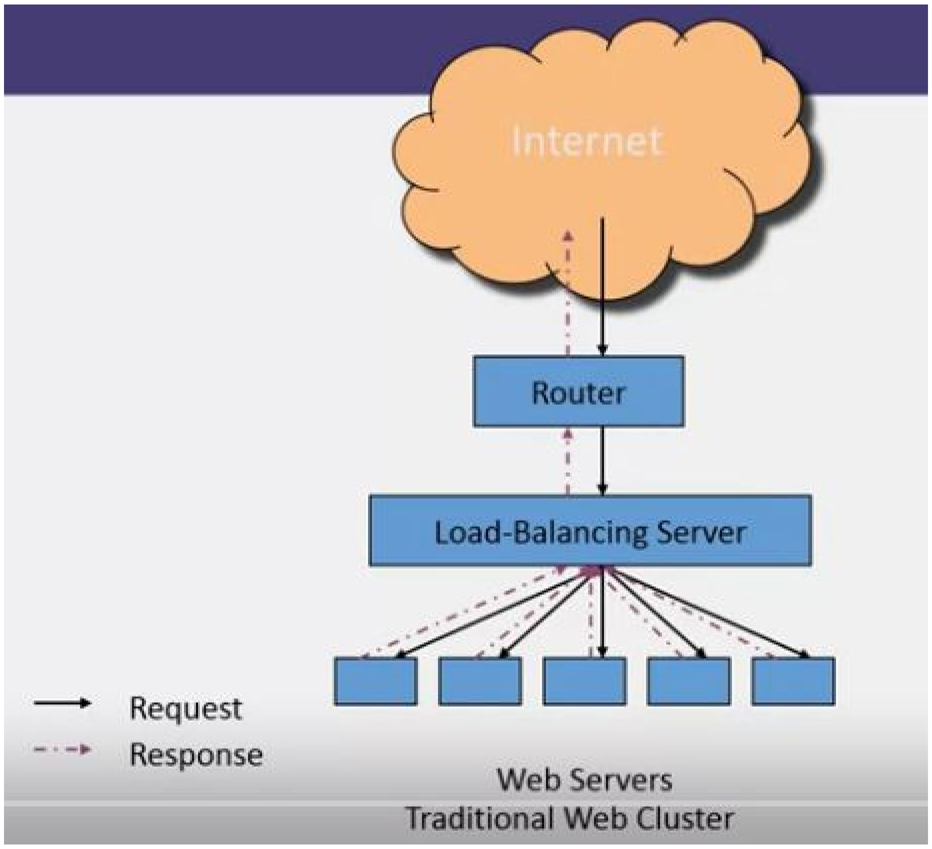
Salesforce Customer Relationship Management (CRM)

Mainly SaaS, PaaS

Web middleware

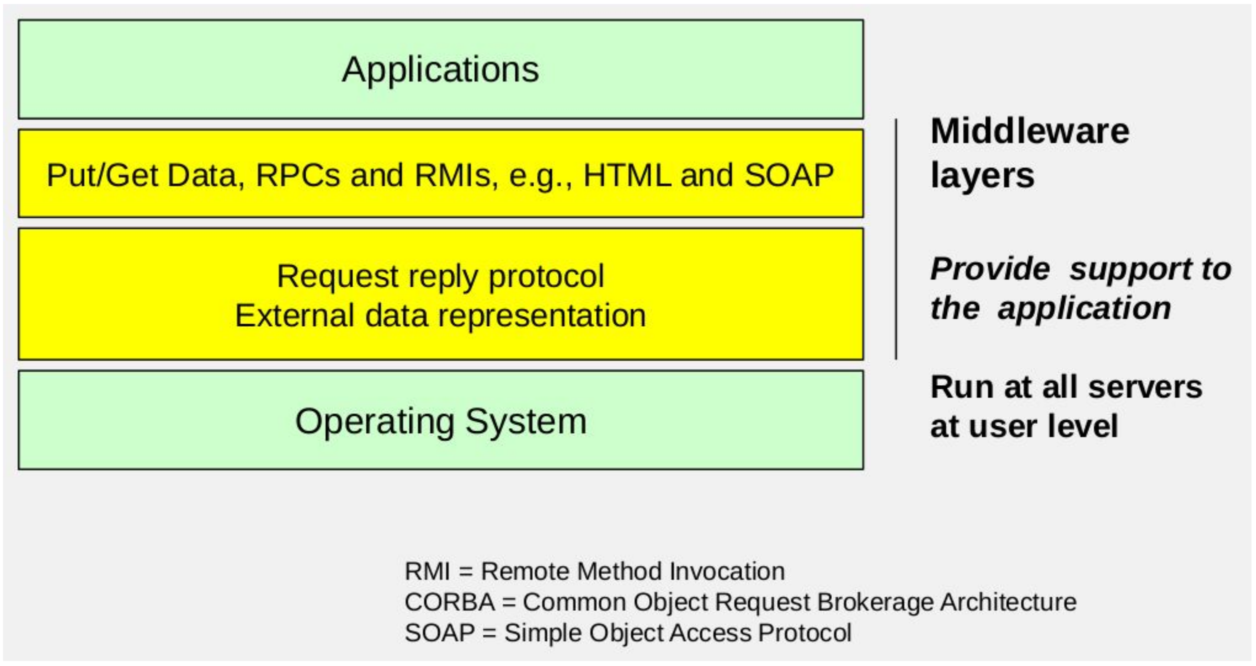
Web Services

- Request enters a router
- Load balancing server determines which web server should serve the request
- Send the request to the appropriate web server



Middleware layer definition

- Software that provides services to applications beyond those generally available at the OS
- Middleware implements functionalities that are common across many different applications
- Building distributed systems while maintaining our code is not very different from a single-node program


Applications
Put/Get Data, RPCs and RMIs, e.g., HTML and SOAP
Request reply protocol
External data representation
Operating System
Middleware layers
Provide support to the application
Run at all servers at user level
RMI = Remote Method Invocation
CORBA = Common Object Request Brokerage Architecture
SOAP = Simple Object Access Protocol

Simple Object Access Protocol

- Transmitted by HTTP or SMTP (or many others)
- Coded in XML (Can be decoded on any machine)
- Return value: Any XML document
- Underlies Web Services Description Language (WSDL)

Common abstraction

- In single-node programs, we often rely on concepts such as
    - Procedure calls
    - Objects
    - Shared memory
- A Middleware Layer can provide the same abstractions

Local objects

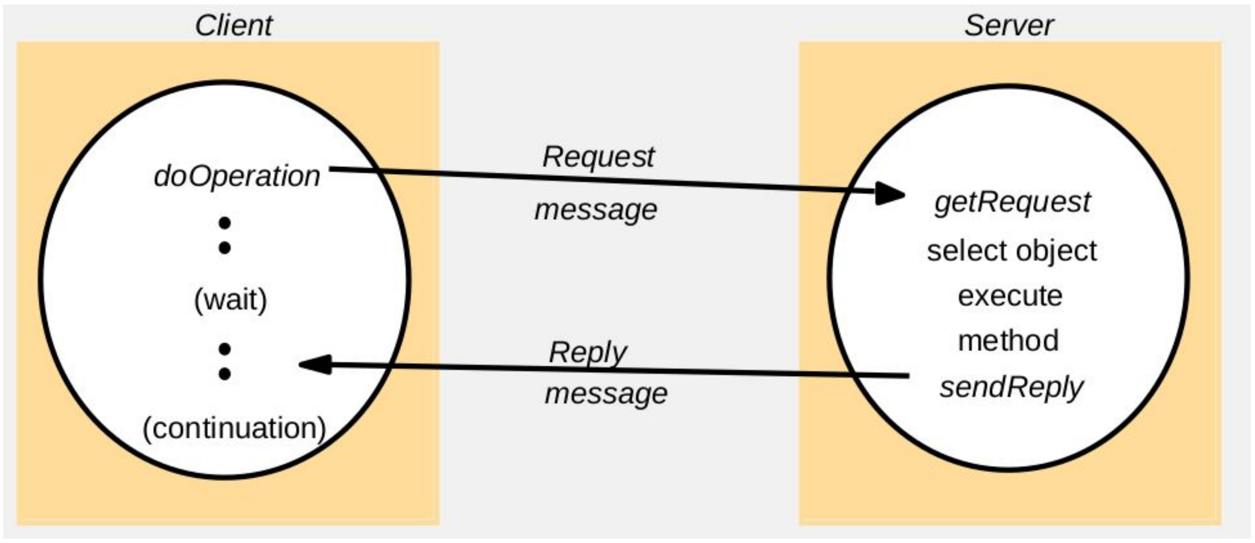
- Within one process' address space
- Object
    - Consists of a set of data and a set of methods
    - For example, C++ object, Chord object at a client (Chord data structures + functions at a node)
- Object reference
    - An identifier by which objects can be accessed, i.e., a pointer
- Interface

- Provides a definition of the signatures of a set of methods (i.e., the types of their arguments, return values, and exceptions) without specifying their implementation
- For example, {put(objectname), get(objectname)} interface for Chord object. Same interface also applies to other P2P objects such as Gnutella, Kazaa, etc

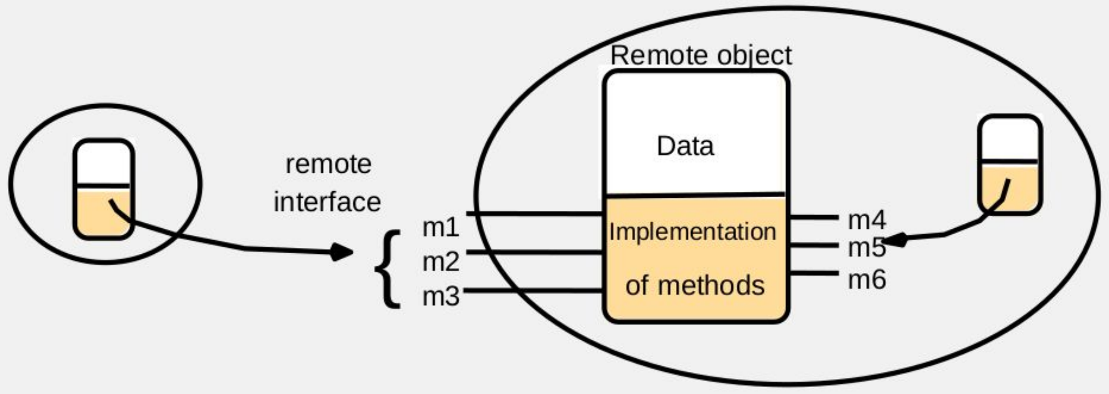
Remote objects

- May cross multiple process' address spaces
- Remote method invocation
    - Method invocations between objects in different processes (processes may be on the same or different host)
    - Remote Procedure Call (RPC): procedure call between functions on different processes in non-object-based system
- Remote objects
    - Objects that can receive remote invocations
- Remote object reference
    - An identifier that can be used globally throughout a distributed system to refer to a particular unique remote object
- Remote interface
    - Every remote object has a remote interface that specifies which of its methods can be invoked remotely, e.g., CORBA interface definition language (IDL)

Request-reply communication

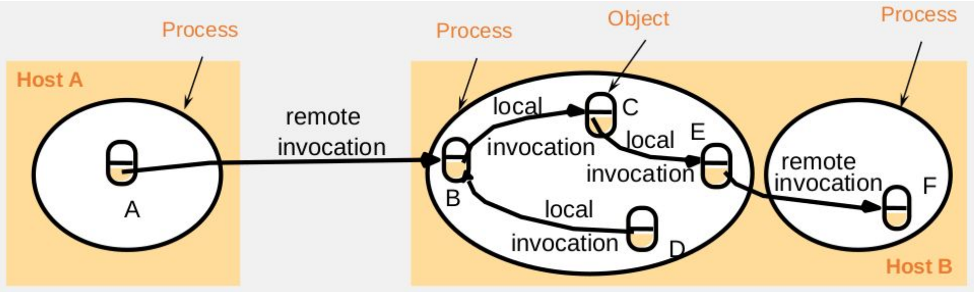


Remote object and its remote interface



Example remote object reference = (IP, port, object number, signature, time)

Remote and local method invocation

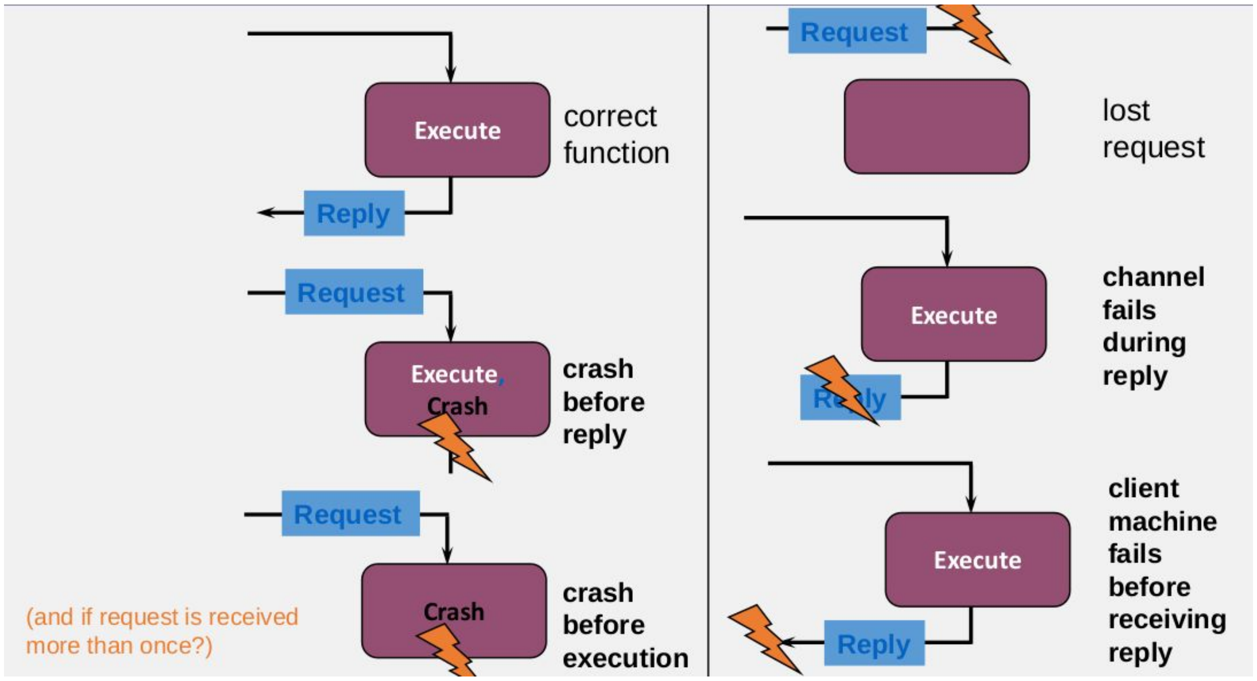


Local invocation = between objects on same process; has *exactly once* semantics

Remote invocation = between objects on different processes. Ideally, you'd also want *exactly once* semantics for remote invocations, but it's difficult in distributed systems (why?)

- Remote method invocation (RMI)
- Remote procedure call (RPC)

Failure mode of RMI RPC

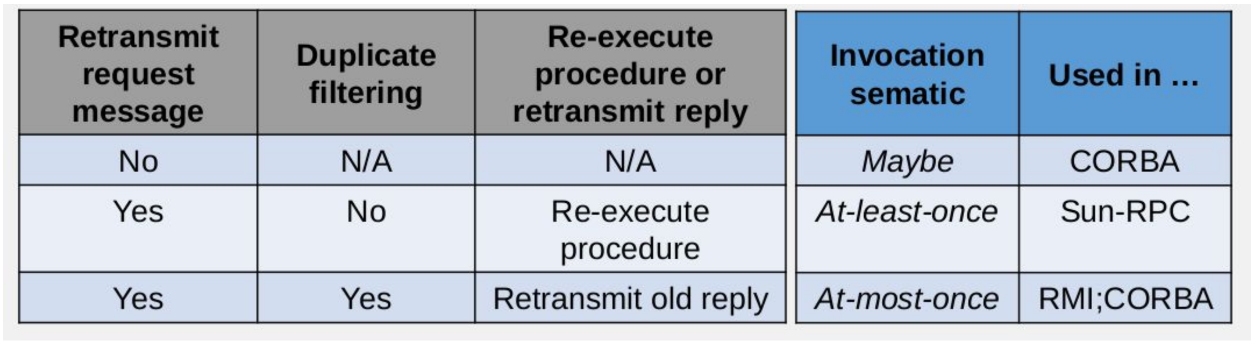


Invocation Semantics

- Middleware aims at providing transparency
  - Sometimes obtaining transparency is expensive and unnecessary
  - Because of faults, obtaining “exactly once” semantics is expensive
- Other invocation semantics are possible
  - Maybe: Caller cannot determine whether the remote method has been executed
  - At-least-once: Caller either receives a result (in which case the user knows the method was executed at least once) or an exception
  - At-most-once: Caller either receives a result (in which case the user knows the method was executed at most once) or an exception
- The right invocation semantic depends on the application
  - For example, at-least-once semantic + idempotent operations can provide a semantic that would be equivalent to exactly-once applications

(Idempotent = same result if applied repeatedly, w/o side effects)

Invocation Semantics

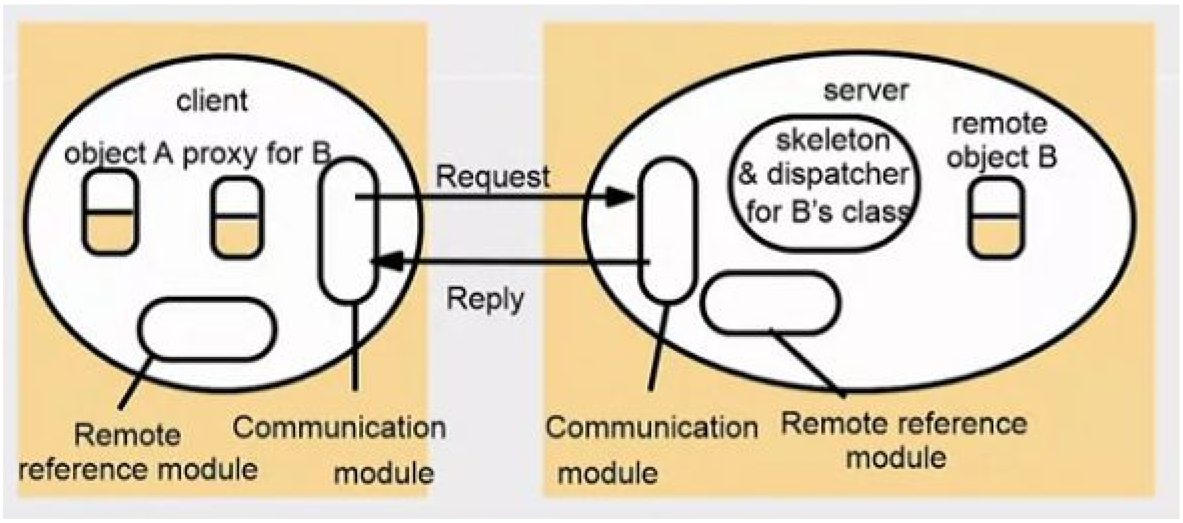


Retransmit request message	Duplicate filtering	Re-execute procedure or retransmit reply	Invocation sematic	Used in …
No	N/A	N/A	*Maybe*	CORBA
Yes	No	Re-execute procedure	*At-least-once*	Sun-RPC
Yes	Yes	Retransmit old reply	*At-most-once*	RMI;CORBA

- **Retransmit request message**: Whether to retransmit the request message until either a reply is received or the server is assumed to be failed
- **Duplicate filtering**: When retransmissions are used, whether to filter out duplicate requests at the server
- **Re-execute procedure or retransmit reply**: Whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations

Middleware aims to make the call appear similar to a local car

- Proxy on the “client” (process P1) side
- Skeleton and dispatcher on the “server” (process P2) side



Proxy

- Is responsible for making RMI transparent to clients by behaving like a local

object to the invoker

  - The proxy implements (Java term, not literally) the methods in the interface of the remote object that it represents. But…
- Instead of executing an invocation, the proxy forwards it to a remote object
  - On invocation, a method of the proxy *marshals* the following into a request message: (i) a reference to the target object, (ii) its own method id and (iii) the

    argument values. Request message is sent to the target, Do then proxy awaits the reply message, unmarshals it, and returns the results to the invoker
  - Invoked object *unmarshals* arguments from request message, and when done, marshall return values into reply message

Marshalling / unmarshalling

- External data representation: an agreed, platform-independent, standard for the representation of data structures and primitive values
  - CORBA Common Data Representation (CDR)
  - Allows an ARM client (possibly big endian) to interact with a x86 Unix server (little endian).
- Marshalling: the act of taking a collection of data items (platform dependent) and assembling them into the external data representation
- Unmarshalling: the process of disassembling data that is in external data representation form into a locally interpretable form

CORBA: Common Object Request Broker Architecture
It is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented. CORBA is an example of the distributed object paradigm.

PXE

In computing, the Preboot eXecution Environment (PXE, most often pronounced as pixie) specification describes a standardized client-server environment that boots a software assembly, retrieved from a network, on PXE-enabled clients. On the client side it requires only a PXE-capable network interface controller (NIC), and uses a small set of industry-standard network protocols such as DHCP and TFTP.

IPMI

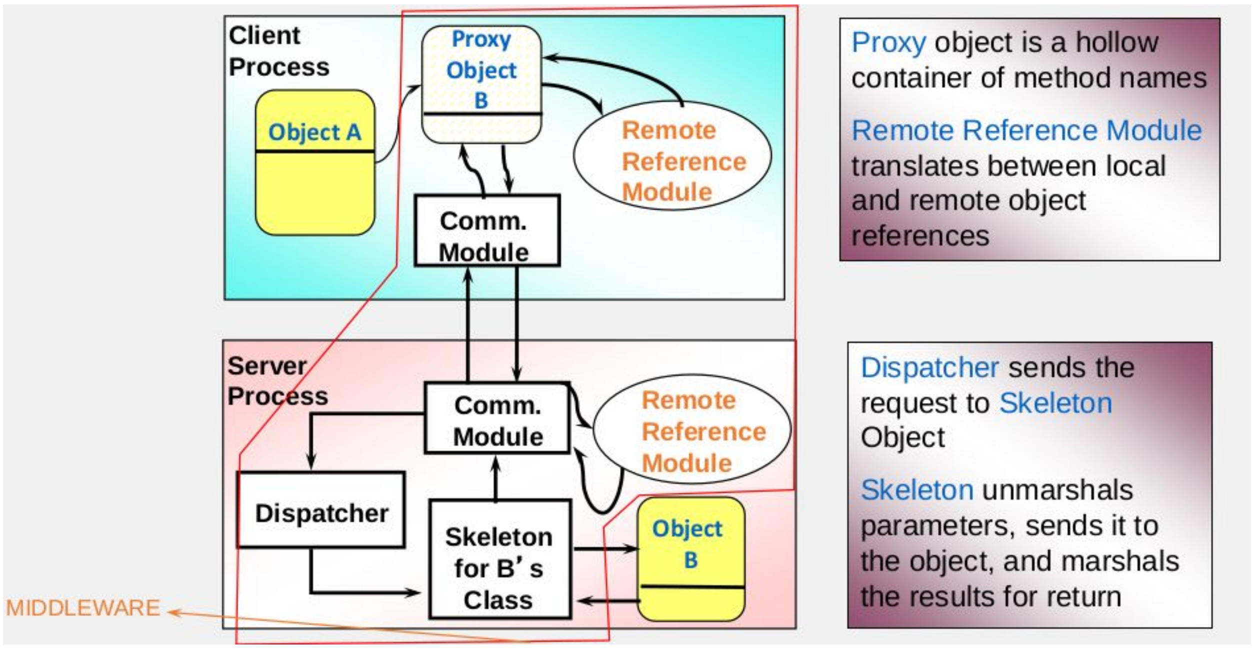
The **Intelligent Platform Management Interface** (IPMI) is a set of computer interface specifications for an autonomous computer subsystem that provides management and monitoring capabilities independently of the host system's CPU, firmware (BIOS or UEFI) and operating system. IPMI defines a set of interfaces used by system administrators for out-of-band management of computer systems and monitoring of their operation. For example, IPMI provides a way to manage a computer that may be powered off or otherwise unresponsive by using a network connection to the hardware rather than to an operating system or login shell. Another use case may be installing a custom operating system remotely. Without IPMI, installing a custom operating system may require an administrator to be physically present near the computer, insert a DVD or a USB flash drive containing the OS installer and complete the installation process using a monitor and a keyboard. Using IPMI, an administrator can mount an ISO image, simulate an installer DVD, and perform the installation remotely

Remote Reference Module

- Is responsible for translating between local and remote object references and for creating remote object references
- Has a remote object table
    - An entry for each remote object held by any process (e.g., B at P2)
    - An entry for each local proxy (e.g., proxy-B at P1)
- When the remote reference module sees a new remote object, it creates a remote object reference and adds it to the table
- When a remote object reference arrives in a request or reply message, the remote reference module is asked for the corresponding local object reference, which may refer to either a proxy or to a remote object
- In case the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table

Dispatcher and Skeleton

- Each process has one dispatcher and a skeleton for each local object (actually, for the class)
- The dispatcher receives all request messages from the communication module
    - For the request message, it uses the method id to select the appropriate method in the appropriate skeleton, passing on the request message
- Skeleton "implements" the methods in the remote interface
    - A skeleton method unmarshals the arguments in the request message and invokes the corresponding method in the remote object (the actual object)
    - It waits for the invocation to complete and marshals the result, together with any exceptions, into a reply message


Client Process
Proxy Object B
Object A
Remote Reference Module
Comm. Module
Server Process
Comm. Module
Remote Reference Module
Dispatcher
Skeleton for B's Class
Object B
MIDDLEWARE
Proxy object is a hollow container of method names
Remote Reference Module translates between local and remote object references
Dispatcher sends the request to Skeleton Object
Skeleton unmarshals parameters, sends it to the object, and marshals the results for return

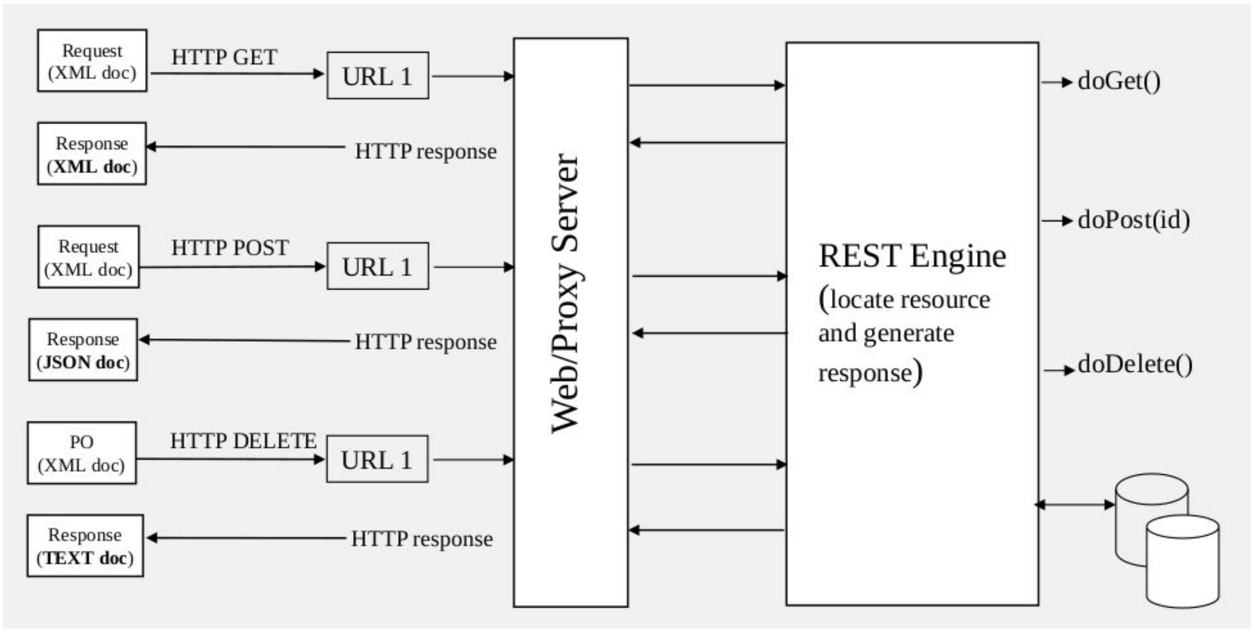
Hypertext Transfer Protocol (HTTP)

- A communications protocol
- Allows retrieving inter-linked text documents (hypertext)
    - World Wide Web
- HTTP verbs
    - HEAD
    - GET: How clients ask for the information they seek
        - Issuing a GET request transfers the data from the server to the client in some representation
    - POST: creates a resource
    - PUT updates a resource
    - DELETE Removes the resource identified by the URI
    - TRACE
    - OPTIONS
    - CONNECT

How data is represented or returned to the client for presentation

- Two main formats:
    - JavaScript Object Notation (JSON)
    - XML
- It is common to have multiple representations of the same data

Architecture style



SOAP- Simple Object Access Protocol

- Transmitted by HTTP or SMTP (or many others)

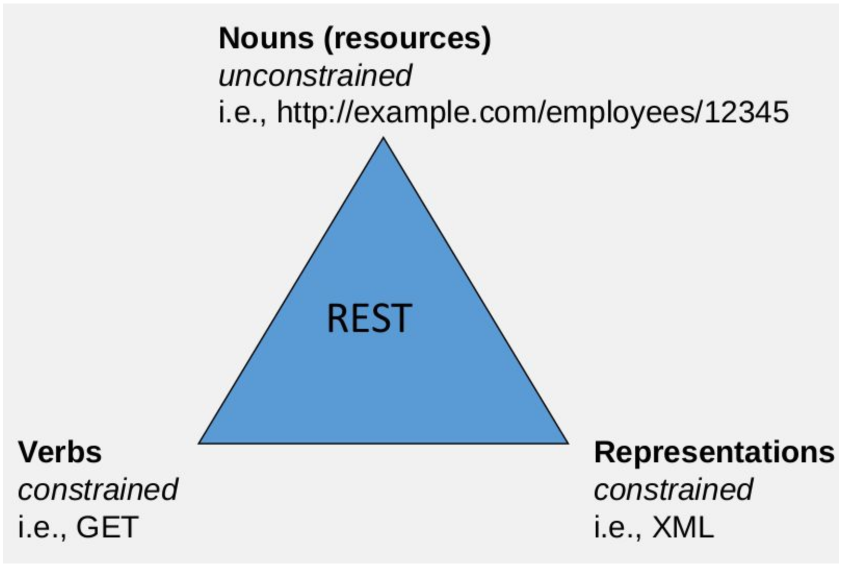
- Coded in XML (can be decoded on any machine)
- Return value: any XML document
- Underlies Web Services Description Language (WSDL)

Representational State Transfer (REST)

- A style of software architecture for distributed hypermedia systems such as the World Wide Web
- Introduced in the doctoral dissertation of Roy Fielding
    - One of the principal authors of the HTTP specification
- A collection of network architecture principles that outline how resources are defined and addressed
- The motivation for REST was to capture those characteristics of the Web that made the Web successful
    - URI-addressable resources
    - HTTP
    - Make a request – receive response – display response
- Exploits the use of the HTTP beyond HTTP POST , HTTP for GET
    - HTTP PUT, HTTP DELETE

REST – Not a Standard

- But it uses several standard
    - HTTP
    - URL
    - XML/HTML/GIF/JPEG/etc. (resource representations)
    - Text/xml, text/html, image/gif, image/jpeg, etc. (resource types, MIME types)



Resources

- The key abstraction of information in REST is a resource

- A resource is a conceptual mapping to a set of entities
    - Any information that can be named can be a resource: a document or image, a temporal service (e.g., "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g., a person), and so on
- Represented with a global identifier (URI in HTTP)
    - http://www.boeing.com/aircraft/747
- REST uses URI to identify resources
- As you traverse the path from more generic to more specific, you are navigating the data

Protocol buffer and Thrift

This is about how we get the data from the client to the application in a very reliable way.

Protocol Buffers (Protobuf) is a method of serializing structured data. It is useful in developing programs to communicate with each other over a wire or for storing data. The method involves an interface description language that describes the structure of some data and a program that generates source code from that description for generating or parsing a stream of bytes that represents the structured data.

- Invented by Google
- Build on RPCs
- Language-neutral, platform-neutral
- Extensible mechanism
- Serializing structured data
- For distributed services

Example schema

```
Message Person{
required int32 id = 1;
required string name = 2;
optional string email = 3;
}
```

Thrift Network Stack

Interface Definition Language. Creates files for clients and servers from needs to serialize structured data (Facebook)

Server (Single-threaded, event-driven)
Processor (Compiler-generated)
Protocol (JSON, compact)

Transport (raw TCP, HTTP, etc.)

Transport method

- Method: Open, close, read, write, flush
- Server transport also has open, listen, accept, and close allowing
    - Read / write to / from a file on disk
    - Http

Example file transports

- TFileTransport – This transport writes to a file
- TFramedTransport – Transport for non-blocking server
- TMemoryTransport – Uses memory for I/O
- TSocket – Uses blocking socket I/O for transport
- TZlibTransport – Performs compression using zlib

Example server codes

- TNonblockingServer – A multi-threaded server using non-blocking I/O
- TSimpleServer – A single-threaded server using standard blocking I/O. Useful for testing.
- TThreadPoolServer – A multi-threaded server using standard blocking I/O.

Mobile Backend as a Service (MBaaS)

- General idea: mobile apps need common services that can be shared among apps instead of being custom developed for each
- Enable web and mobile app developers to link their applications to backend cloud storage and backend APIs
    - Cloud storage
    - User management
    - Push notifications
    - Integration with social networking services
- Provide all of these in a one-shop mode

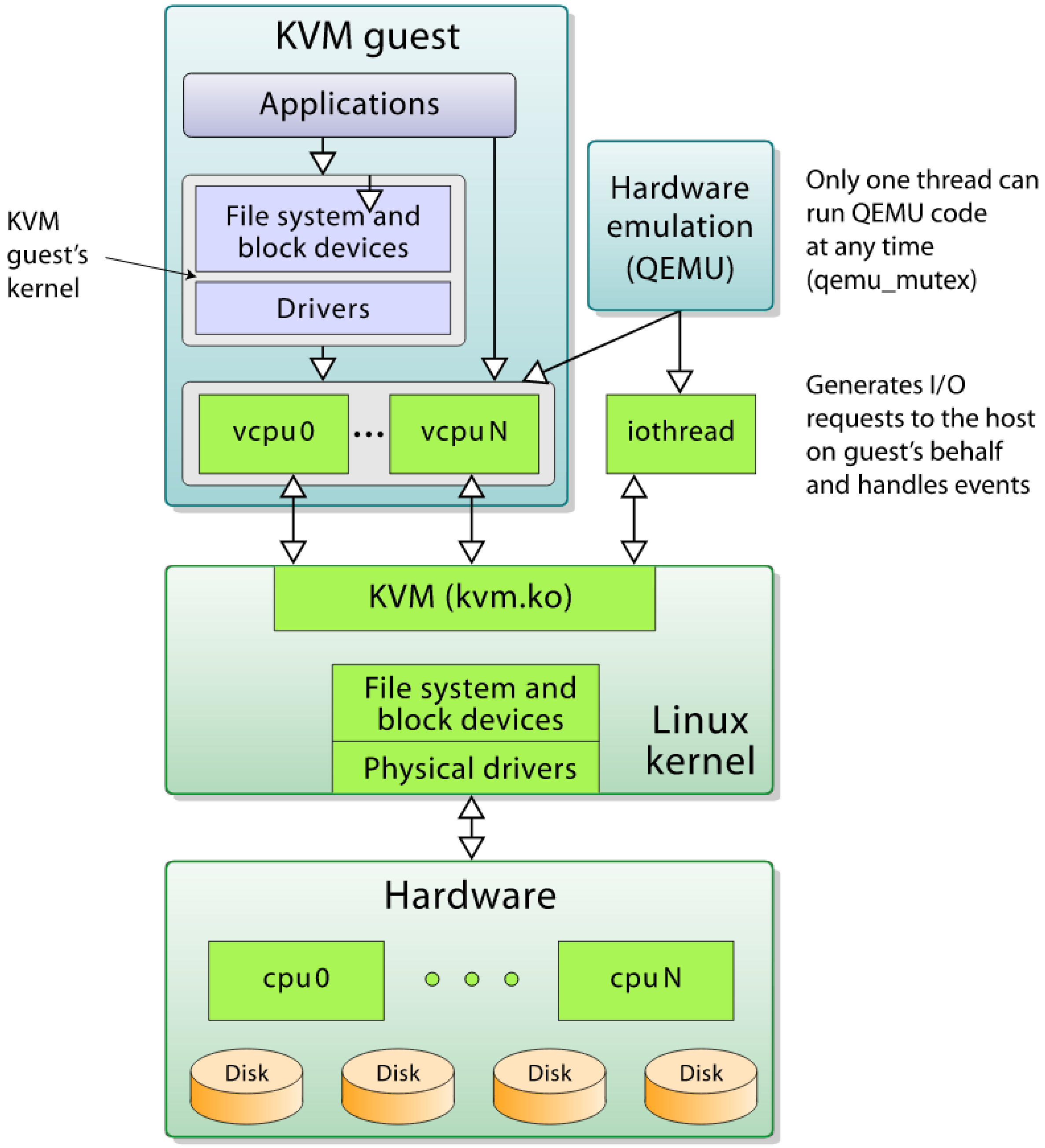
MBaaS example

- Appcelerator, AnyPresence, Appery, Built.io, FeedHenry, Kinvey, TruMobi, Apple CloudKit (iCloud), etc.
- Many commonalities
    - E.g. many use MongoDB to serve JSON objects
    - REST API common
    - MicroServices
    - DevOps
    - Frontend design framework
- Different levels of enterprise integration

- Either on-premise or in private clouds
- Some support compliance with HIPAA, PCI, FIPS, and EU data security standards

Virtualization Examples - KVM

"**KVM** for (Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware."


KVM guest
Applications
File system and block devices
Drivers
KVM guest's kernel
vcpu 0
vcpu N
Hardware emulation (QEMU)
Only one thread can run QEMU code at any time (qemu_mutex)
iothread
Generates I/O requests to the host on guest's behalf and handles events
KVM (kvm.ko)
File system and block devices
Physical drivers
Linux kernel
Hardware
cpu 0
cpu N
Disk
Disk
Disk
Disk

Advantages of KVM

- It is an Open Source solution, and, as such, free to customize.

- Using KVM is efficient from a financial perspective as well, due to the lower costs associated with it.
- It is highly scalable.
- KVM employs advanced security features, utilizing SELinux. It provides MAC (Mandatory Access Control) security between Virtual Machines. KVM has received awards for meeting common government and military security standards and for allowing open virtualization for homeland security projects.

VirtualBox is an x86 and AMD64/Intel64 virtualization product from Oracle, which runs on Windows, Linux, Macintosh, and Solaris hosts and supports Guest OSes from Windows, Linux families, and others, like Solaris, FreeBSD, DOS, etc.

Advantages of VirtualBox

- It is an Open Source solution.
- It is free to use.
- It runs on Linux, Windows, OS X, and Solaris.
- It provides two virtualization choices: software-based virtualization and hardware-assisted virtualization.
- It is an easy-to-use multi-platform Hypervisor.
- It provides the ability to run virtualized applications side by side with normal desktop applications.
- It provides teleportation - live migration.

Using virtual machines in a development environment has numerous benefits:

- Reproducible environment
- Management of multiple projects in their restricted environment
- Sharing the environment with other teammates
- Keeping the development and deployment environments in sync
- Running the same VM on different OSes, with a hypervisor like VirtualBox.

Vagrant by HashiCorp helps us automate the setup of one or more VMs by providing an end-to-end lifecycle using the vagrant command line

Boxes: We need to provide an image in the Vagrantfile, which we can use to instantiate machines.
Synced Folders: With the Synced Folder feature, we can sync a directory on the host system with a VM, which helps the user manage shared files/directories easily.
Providers: While Vagrant ships out of the box with support for VirtualBox, Hyper-V, and Docker, Vagrant has the ability to manage other types of machines as well. This is done by using other providers with Vagrant.
Provisioning: Provisioners allow us to automatically install software, make configuration changes, etc. after the machine is booted

Some of the benefits of using Vagrant are:

- It automates the setup of one or more VMs, which results in saved time and lower operational costs.

- It is a cross-platform tool.
- It provides support for Docker, thus helping us manage Docker containers.
- It is easy to install.
- It is very useful in multi-developer teams.

Infrastructure as a Service (IaaS) is a form of cloud computing which provides on-demand physical and virtual computing resources, storage, network, firewall, load balancers, etc. To provide virtual computing resources, IaaS uses some form of hypervisor, like Xen, KVM, VMware ESX/ESXi, Hyper-V, etc.

Let's say that you want to have a set of ten Linux systems with 4GB RAM each, and two Windows systems with 8GB each to deploy your software. You can go to any of the IaaS providers and request these systems. Generally, a IaaS provider creates the respective VMs in the background, puts them in the same internal network, and shares the credentials with you, thus allowing you to access them. Other than VMs, some IaaS providers offer bare metal machines for provisioning.

Amazon EC2: Command line interface,
It provides some preconfigured images, called Amazon Machine Images (AMIs). These images can be used to quickly start instances. We can also create our own custom AMIs to boot our instances.
Example of instances:
- t2.nano: 512 MiB of memory, 1 vCPU, 3 CPU Credits/hour, EBS-only, 32-bit or 64-bit platform
- c4.large: 3.75 GiB of memory, 2 vCPUs, 64-bit platform
- d2.8xlarge: 244 GiB of memory, 36 vCPUs, 24 x 2000 GB of HDD-based instance storage, 64-bit platform, 10 Gigabit Ethernet.

Amazon supports configuring security and network access to our instances.
With Amazon Elastic Block Store (EBS) we can attach/detach persistent storage to our instances.
EC2 supports the provisioning of dedicated hosts, which means we can get an entire physical machine for our use.
Amazon EC2 has many other features, allowing you to:
- Create an Elastic IP for remapping the Static IP address automatically
- Provision a Virtual Private Cloud for isolation. Amazon Virtual Private Cloud provides secure and robust networking for Amazon EC2 instances
- Use CloudWatch for monitoring resources and applications
- Use Auto Scaling to dynamically resize your resources, etc.

Benefits of EC2:
- It is an easy-to-use IaaS solution.
- It is flexible and scalable.
- It provides a secure and robust functionality for your compute resources.

- It enables automation.
- It is cost-effective: you only pay for the time and resources you use.
- It is designed to work in conjunction with other AWS components.
- It promises 99.99% uptime.
- It provides specialized instances for workloads, such as floating point operations, high graphics capability, high input/output (I/O), High Performance Computing (HPC), etc.

MS Azure: command line utility
Benefits
- It is an easy-to-use IaaS solution.
- It is flexible and scalable.
- It provides a secure and robust functionality for your compute resources.
- It enables automation.
- It is cost-effective: you only pay for the time and resources you use.
- It is designed to work in conjunction with other Azure services.

DigitalOcean helps you create a simple cloud quickly, in as little as 55 seconds. All of the VMs are created on top of the KVM hypervisor and have SSD (Solid-State Drive) as the primary disk.
- It allows you to configure a cloud in as little as 55 seconds.
- It is flexible and scalable.
- It provides a high level of security by using KVM virtualized droplets.
- It enables automation.
- It is cost-effective: you only pay for the time and resources you use.
- It is focused on providing a simple, user-friendly experience.
- It uses high-performance Solid State Disks.
- It offers a one-click installation of a multitude of application stacks like LAMP, LEMP, MEAN, and Docker.

Google's Cloud offering, which has many products in different domains, like compute, storage, networking, big data, and others. **Google Compute Engine** provides the compute service. We can manage the instances through GUI, APIs or command line. Access to the individual VM's console is also available.
GCE supports different machine types, which we can choose from depending on our need. They are categorized in the following types:
- Standard machine types
- High-CPU machine types
- High-memory machine types
- Shared-core machine types
- We can also configure custom machine types.

GCE has other features as well, like Persistent Disk, Local SSD, Global Load Balancing, Compliance and Security, Automatic Discount, etc.
It is flexible and allows you to scale your applications easily.
- Fast boot time.

- It is very secure, encrypting all data stored.
- It enables automation.
- It is cost-effective: you only pay for the time and resources you use.
- It supports custom machine types.
- It supports Virtual Private Cloud, Load Balancers, etc.

Cloud Foundry

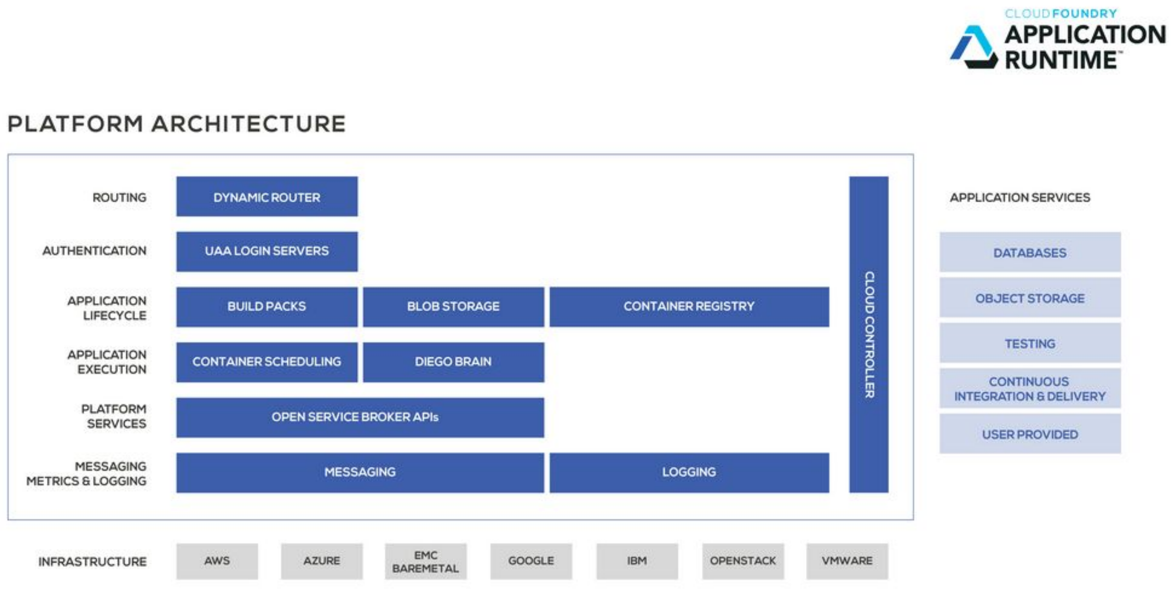
Cloud Foundry CF) is an open source Platform as a Service (PaaS) that provides a choice of clouds, developer frameworks, and application services. It can be deployed on-premise or on IaaS, like AWS, vSphere, or OpenStack. There are many commercial CF cloud providers as well, like IBM Cloud Foundry, SAP Cloud Platform, Pivotal Cloud Foundry, etc. It characteristics include:

- Application portability
- Application auto-scaling
- Centralized platform management
- Centralized logging
- Dynamic routing
- Application health management
- Role-based application deployment
- Horizontal and vertical scaling
- Security
- Support for different IaaS technologies.

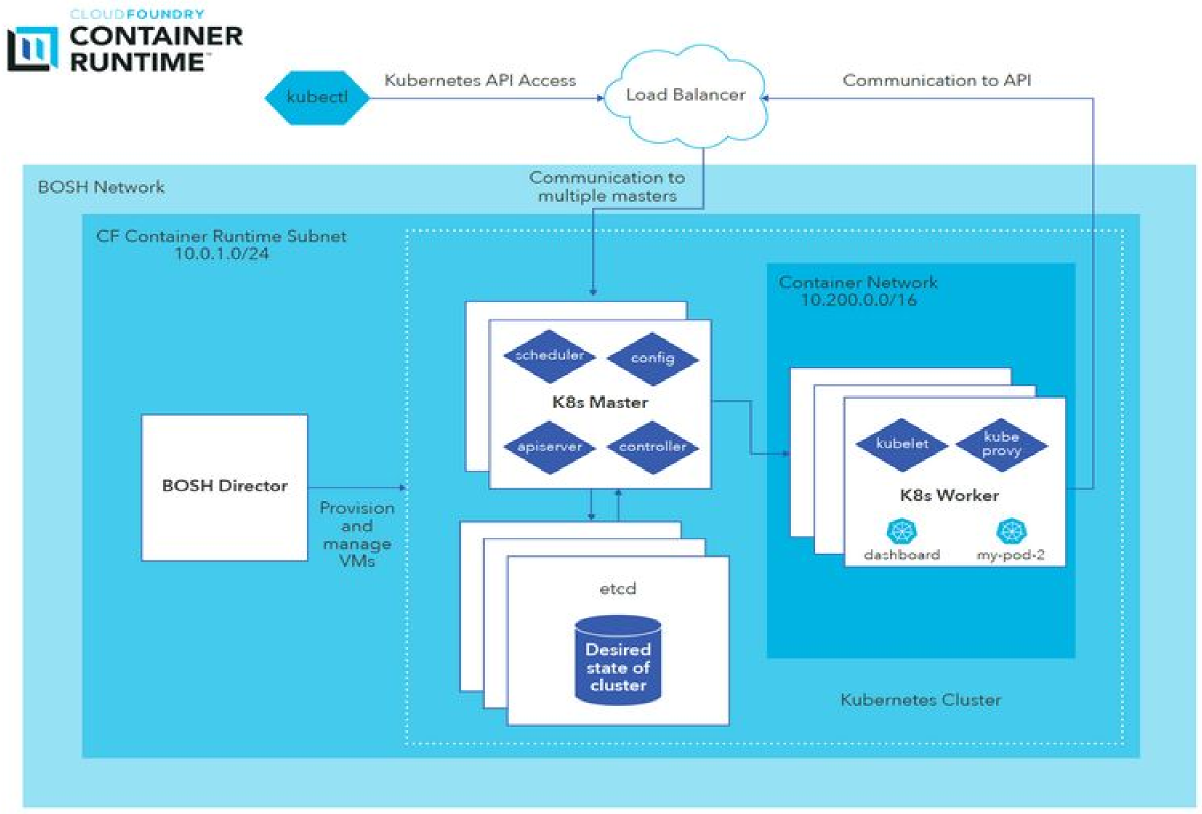
In order to manage the underlying infrastructure for both of them, the Cloud Foundry uses BOSH. BOSH is a cloud-agnostic open source tool for release engineering, deployment, and lifecycle management of complex distributed systems.

CF Application Runtime, previously known as Elastic Runtime, is used by developers to run applications written in any language or framework on the cloud of their choice.

CF Application Runtime uses buildpacks, which provide the framework and runtime support for the applications. They are programming language-specific and have information about how to download dependencies and configure specific applications. Below are some of the languages for which we have well-defined buildpacks: Java, Python, GO, Ruby, .Net, Node.js, PHP. Developers can build new buildpacks or customize the existing ones.



CF Container Runtime gives Cloud Foundry the flexibility to deploy developer-built, pre-packaged applications using containers. The CF Container Runtime platform does it by deploying containers on a Kubernetes cluster, which is managed by CF BOSH. Kubernetes is a container orchestrator which allows us to deploy containers. With CF BOSH, you can achieve high availability, scaling, VM healing and upgrades for the Kubernetes cluster.



Some of the benefits of using Cloud Foundry are:

- It is an open source platform, but there are also many commercial Cloud Foundry providers.
- It offers centralized platform management.
- It enables horizontal and vertical scaling.
- It provides infrastructure security.
- It provides multi-tenant compute efficiency.
- It offers support for multiple IaaS providers.
- It supports the full lifecycle: development, testing, and deployment, thus enabling a continuous delivery strategy. It also provides integration with CI/CD tools.
- It is a simple and flexible solution, supported by an extensive community of developers.
- It reduces the chance of human errors.
- It is cost-effective, reducing the overhead for Ops teams.

Cloud data storage

CEPH

Motivation

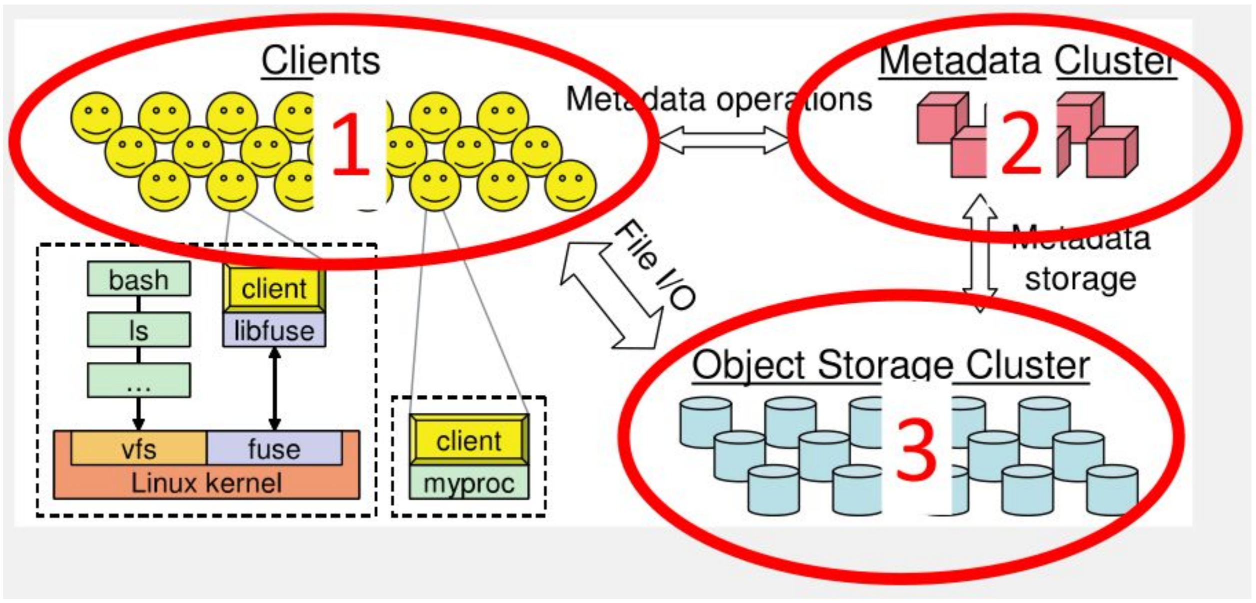
- Ceph is an emerging technology in the production-clustered environment

- Designed for:
    - Performance – Striped data over data servers
    - Reliability – No single point of failure
    - Scalability – Adaptable metadata cluster
- More general than HDFS
    - Smaller files
    - GlusterFS has more or less similar ideas
    - Uses ring-based hashing; Ceph uses CRUSH

Overview

- MDS – Meta Data Server
- ODS – Object Data Server
- MON – Monitor (now fully implemented)
- Decoupled data and metadata
    - I/O directly with object servers
- Dynamic distributed metadata management
    - Multiple metadata servers handling different directories (subtrees)
- Reliable autonomic distributed storage
    - ODS's manage themselves by replicating and monitoring

Components: Ordered: Clients, Metadata, Object Storage



Decoupled Data and Metadata

- Increases performance by limiting
- Interaction between clients and servers
- Decoupling is common in distributed filesystems: HDFS, Lustre, Panasas…
- In contrast to other file systems, Ceph uses a function to calculate the block locations

Dynamic Distributed Metadata Management

- Metadata is split among cluster of servers

- Distribution of metadata changes with the number of requests to even load among metadata servers
- Metadata servers also can quickly recover from failures by taking over neighbors' data
- Improves performance by leveling metadata load

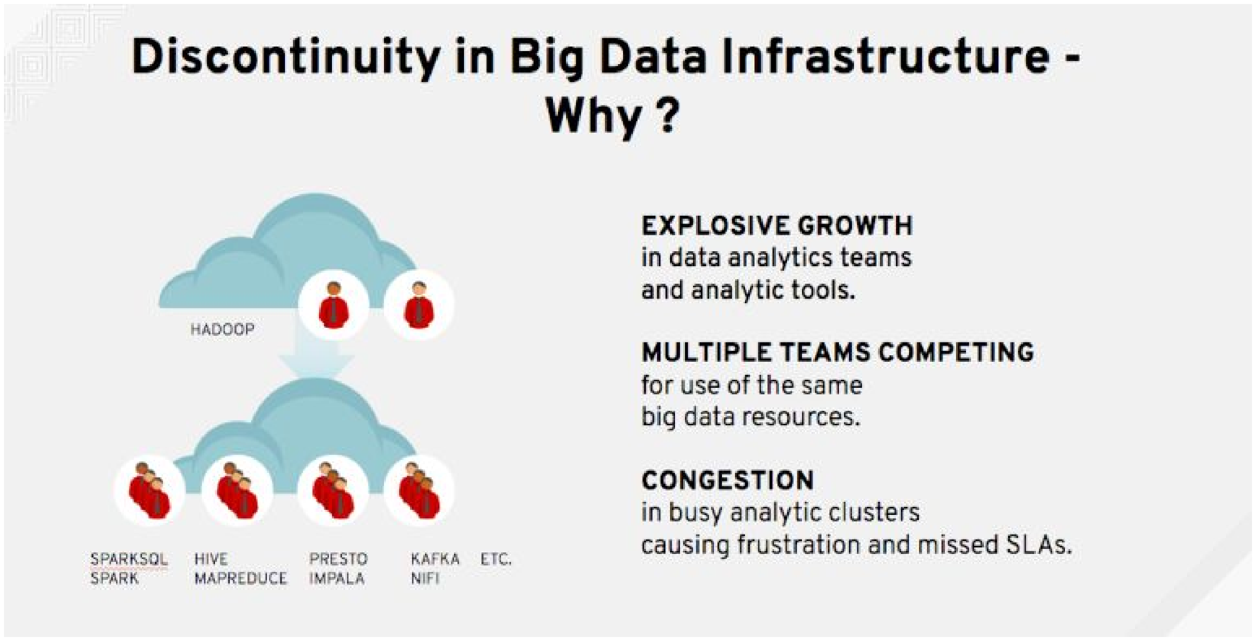
Reliable Autonomic Distributed Storage

- Data storage servers act on events by themselves
- Initiates replication and
- Improves performance by offloading decision making to the many data servers
- Improves reliability by removing central control of the cluster (single point of failure)

OpenShift is an open source PaaS solution provided by Red Hat. It is built on top of the container technology, which uses Kubernetes underneath. OpenShift can be deployed on top of a full-fledged Linux OS or on a Micro OS which is specifically designed to run containers and Kubernetes.

Why Spark on Ceph?

Core problem


HADOOP
SPARKSQL SPARK
HIVE MAPREDUCE
PRESTO IMPALA
KAFKA NIFI
ETC.

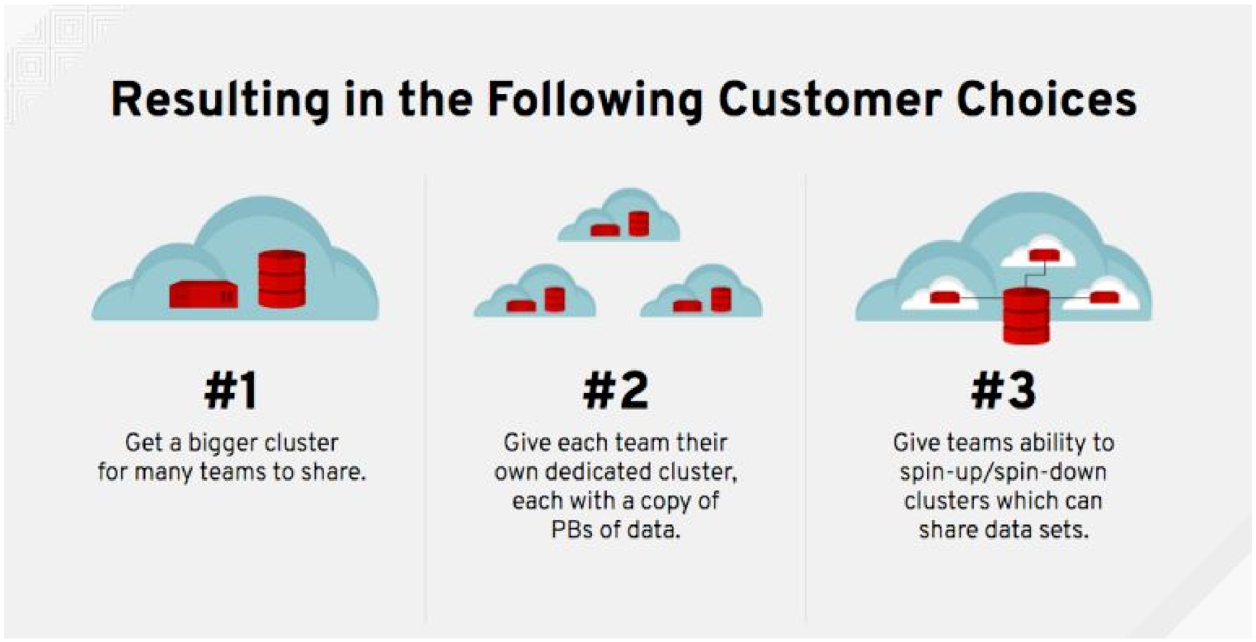
Discontinuity in Big Data Infrastructure - Why ?

EXPLOSIVE GROWTH
in data analytics teams and analytic tools.

MULTIPLE TEAMS COMPETING
for use of the same big data resources.

CONGESTION
in busy analytic clusters causing frustration and missed SLAs.

Solution options



Resulting in the Following Customer Choices

#1
Get a bigger cluster for many teams to share.

#2
Give each team their own dedicated cluster, each with a copy of PBs of data.

#3
Give teams ability to spin-up/spin-down clusters which can share data sets.

OpenShift is an open source PaaS solution provided by Red Hat. It is built on top of the container technology, which uses Kubernetes underneath. OpenShift can be deployed on top of a full-fledged Linux OS or on a Micro OS which is specifically designed to run containers and Kubernetes.

As OpenShift uses Kubernetes, we get all the features offered by Kubernetes, like adding or removing nodes at runtime, persistent storage, auto-scaling, etc.

OpenShift has a framework called Source to Image (S2I), which enables us to create container images from the source code repository to deploy applications easily.

OpenShift integrates well with Continuous Deployment tools to deploy applications as part of the CI/CD pipeline.
With CLI, GUI and IDE integration applications can be managed easily.

Hive

Background

- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- “ETL” via hand-coded Python
- HQL, a variant of SQL
- Translates queries into map/reduce jobs, Hadoop YARN, Tez, or Spark
- Note
    - No UPDATE or DELETE support, for example
    - Focuses primarily on the query part of SQL

Example
- Hive looks similar to an SQL database
- Relational join on two tables:
    - Table of word counts from Shakespeare collection
    - Table of word counts from Homer

Hive Components
- Shell: allows interactive queries
- Driver: session handles, fetch, execute
- Compiler: parse, plan, optimize
- Execution engine: DAG of stages (MR, HDFS, metadata)
- Metastore: schema, location in HDFS, etc.

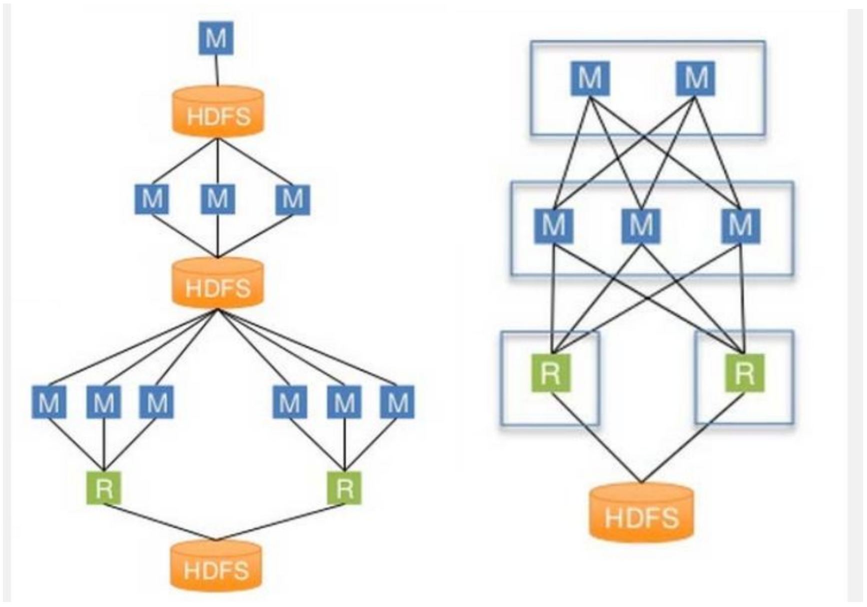
Metastore
- Hive uses a traditional database to store its metadata
    - A namespace containing a set of tables
    - Holds table definitions (column types, physical layout)
    - Holds partitioning information
- Can be stored in MySQL, Oracle, and many other relational databases

Physical Layout
- Warehouse directory in HDFS
    - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
    - Partitions form subdirectories of tables
- Actual data stored in flat files
    - Control char-delimited text, or SequenceFiles
    - Can be customized to use arbitrary format

Tez

Design Goals of Tez
- Empowering end users by:
    - Expressive dataflow definition APIs
    - Flexible Input-Processor-Output runtime model
    - Data type agnostic
    - Simplifying deployment
- Execution Performance
    - Performance gains over Map Reduce
    - Optimal resource management
    - Plan reconfiguration at runtime
    - Dynamic physical data flow decisions

Integrate Hive and Pig
- extra map/reduce jobs when implemented with Hadoop compared with using Tez on Yarn



SWIFT - an object store

A **Binary Large OBject (BLOB)** is a collection of binary data stored as a single entity in a database management system.

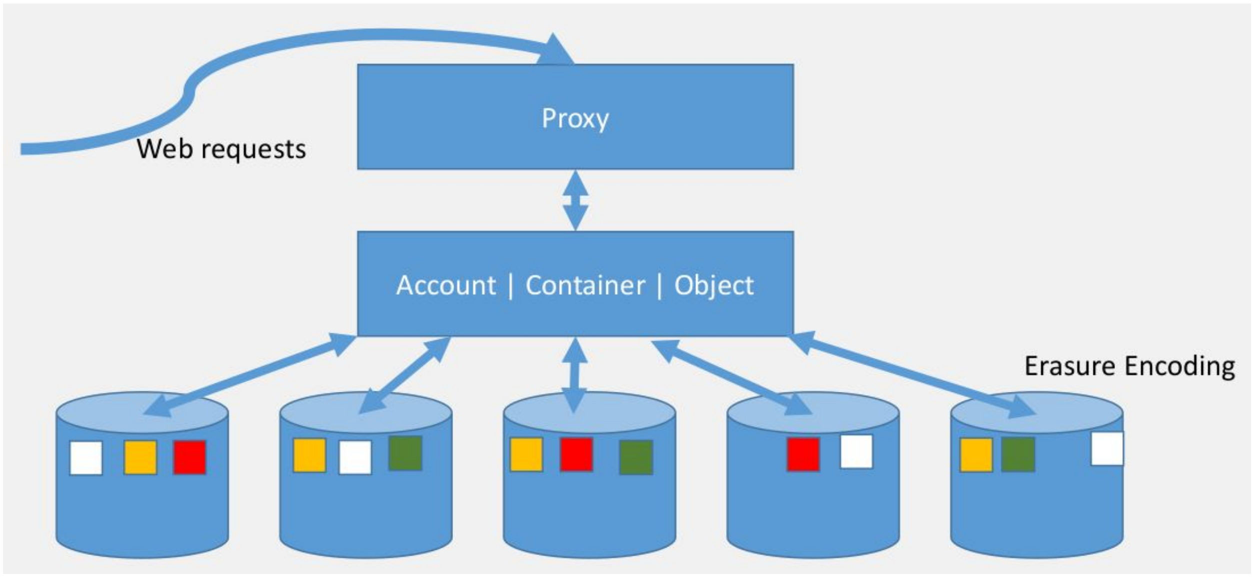
Use cases:

- Store unstructured object data like text or binary data
- Images
- Movies
- Audio, Signal data
- Large queue of messages
- Example is LinkedIn data in a user page (Uses Space Ambry)
- Usually accessible over the web

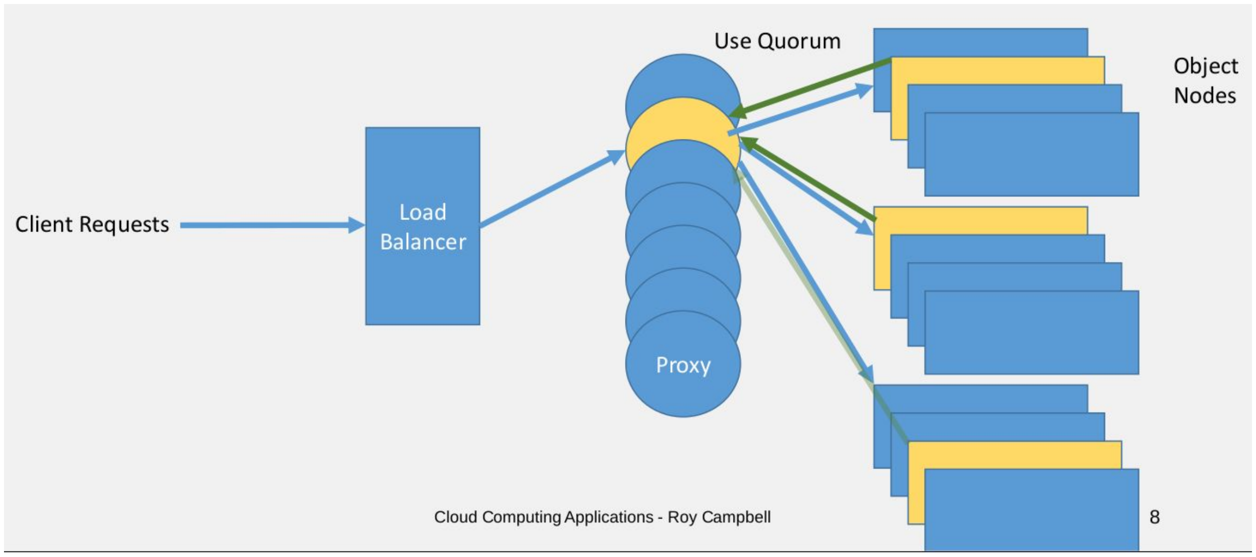
Goal

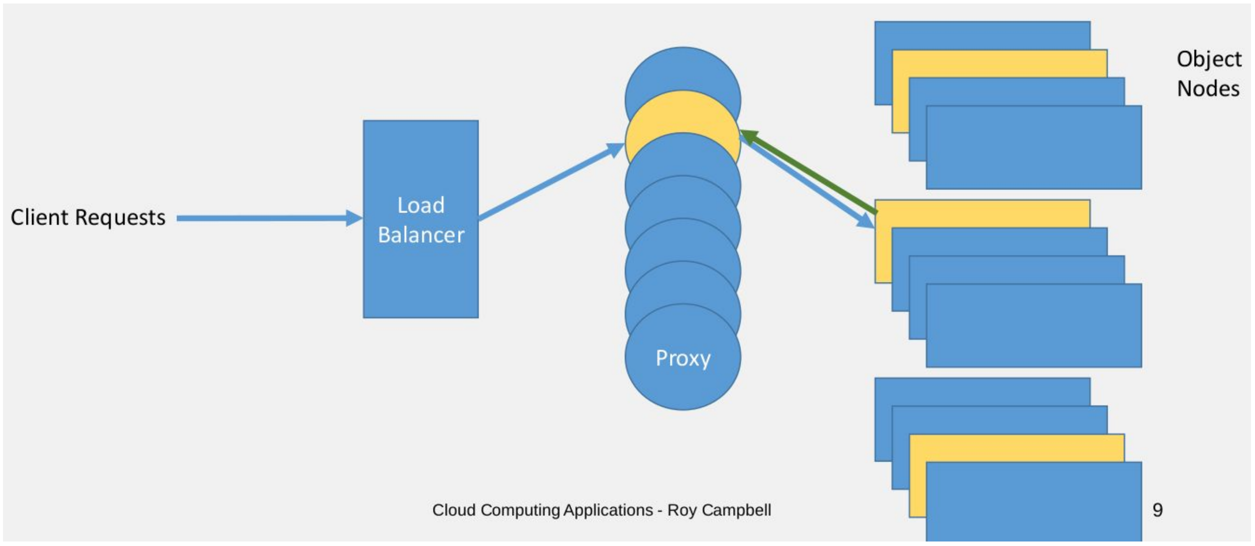
- Data growth ~ 50% a year
- 50%-70% data is unstructured or archival
- RESTful API (HTTP)
- High availability (no single point of failure)
- Agile data centers
- Open Source
- Multi-region, geographic distribution of data
- Storage policies
- Erasure Coding

SWIFT components



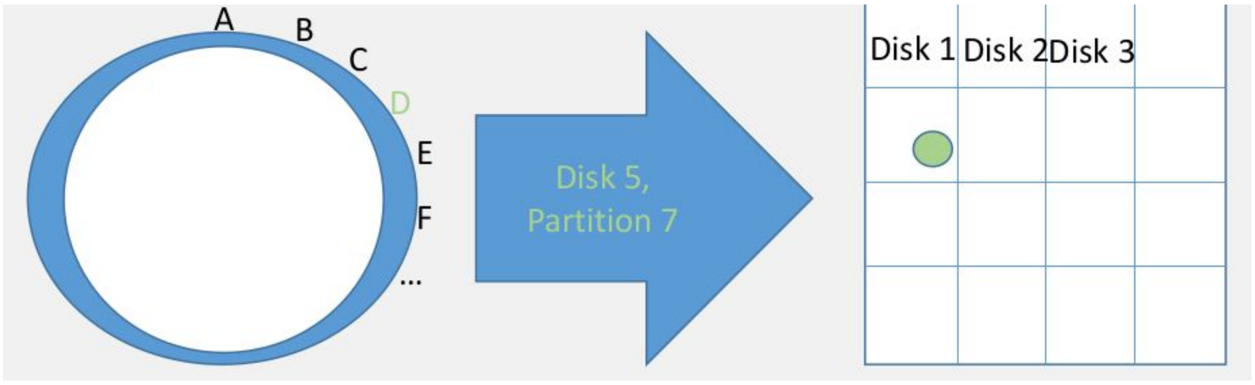
Write and read request





Details
- MD5 Checksums with each object
- Auditing and active replication
- Any sized disks

Swift partitions
- 1 Node, 8 Disks, 16 Partitions per disk, 8*16 = 128 partitions
- 2 Nodes, 8 Disks each, 8 Partitions per disk, 8*16 = 128 partitions
- Use Hash into Ring to map objects into storage partitions



Amazon S3 Blob Storage

- Scalability, high availability, low latency – 99.99% availability
- Files up to 5 terabytes
- Objects stored in buckets owned by users
- User assigned keys refer to objects
- Amazon Machine Images (exported as a bundle of objects)
- SmugMug, Hadoop file store, Netflix, reddit, Dropbox, Tumbler

Simple Storage Service (S3)

- A bucket is a container for objects and describes location, logging, accounting, and access control.
- A bucket has a name that must be globally unique.
    - http://bucket.s3.amazonaws.com
    - http://bucket.s3-aws-region.amazonaws.com.
- A bucket can hold any number of objects, which are files of up to 5TB.
    - http://bucket.s3.amazonaws.com/object
    - http://johnsmith.s3.amazonaws.com/photos/puppy.jpg

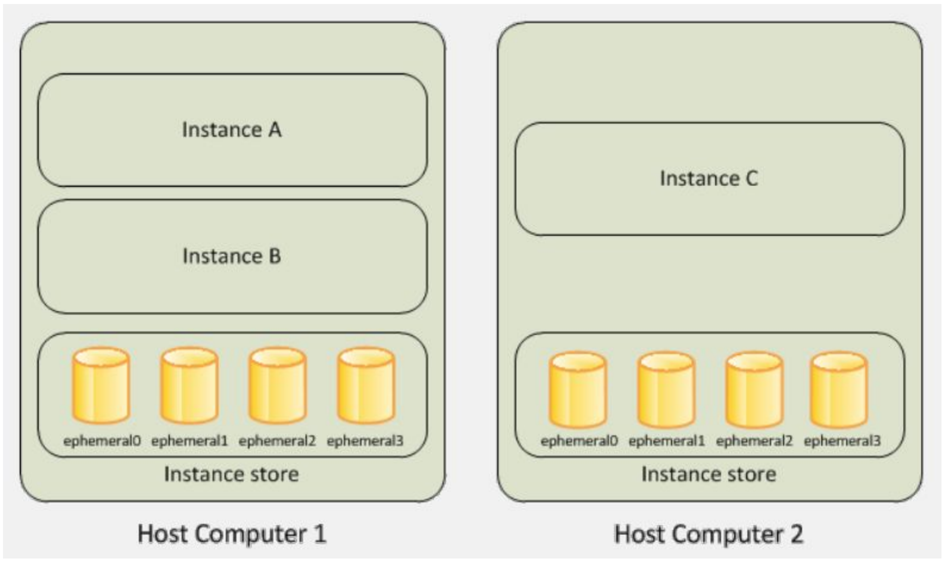
S3 Weak Consistency Model

- “Updates to a single key are atomic....”
- “Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. If a PUT request is successful, your data is safely stored. However:
    - A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
    - A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
    - A process replaces an existing object and immediately attempts to read it. Until change is fully propagated, Amazon S3 might return the prior data.
    - A process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.”

AWS block stores

AWS instance store

- Instance stores are another form of cloud-hosted temporary block-level storage
    - These are provided as part of an 'instance', such as an Amazon EC2
- Their contents will be lost if the cloud instance is stopped.
    - But offer higher performance and bandwidth to the instance.
    - Located on disks that are physically attached to the host computer
- They are best used for temporary storage such as caching or temporary files, with persistent storage held on a different type of server.



Instance Store Lifetime

- Data persists only during the lifetime of its associated instance
    - It persists a reboot
- Data lost if
    - The underlying disk drive fails
    - The instance stops
    - The instance terminates
- You can get reliability by
    - A distributed file system (e.g. HDFS)
    - Backup to S3 or EBS

Instance Store Size

- A typical instance store is small
- SSD: can be any where from around 80 GB to 320 GB SSD, up to 3,840 GB on x1.32xlarge
- HDD: when available (on older generation instances), up to 1,680 GB

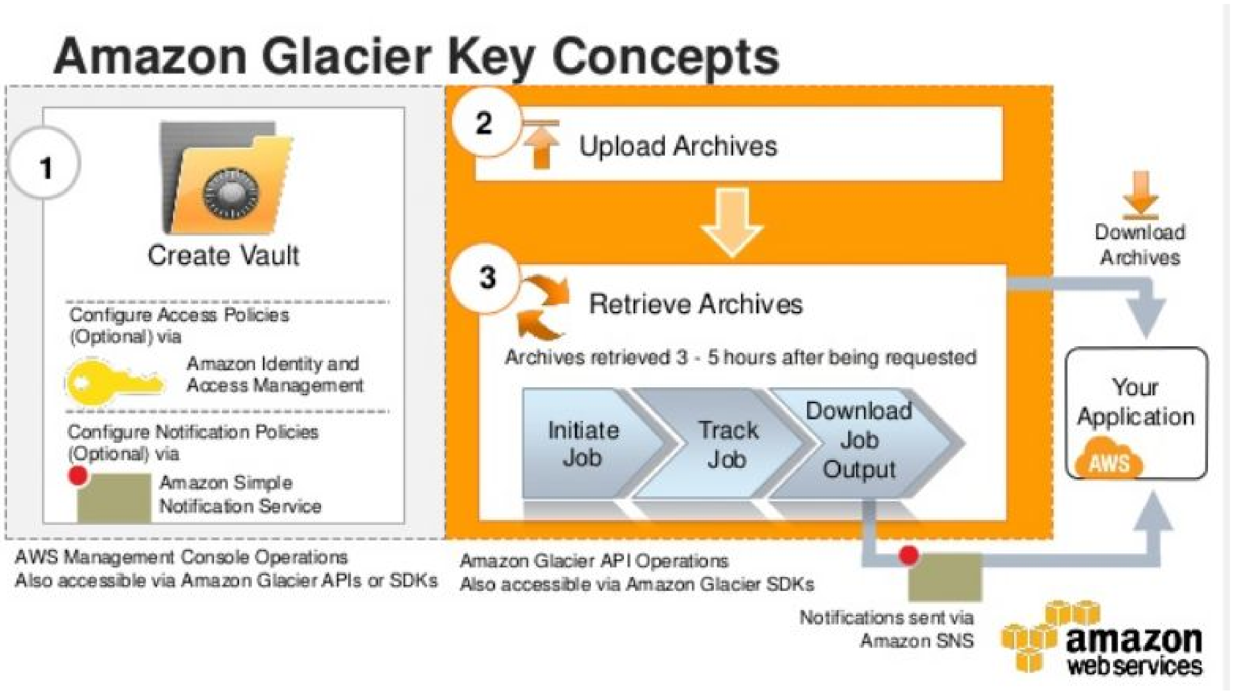
Amazon AWS EBS

- EBS Volumes are highly available and reliable
- Can be attached to running instances in the same availability zones
- Persist independently of the life of an instance
- Use when data must be quickly accessible and requires long-term persistence
- Support encryption
- Up to 16TB in size
- Different types
    - General Purpose SSD (gp2)
    - 100 IOPS/GiB, burst up to 10,000, 160 MB/s throughput
    - Provisioned IOPS SSD (io1)
        - Provision a specific level of performance
        - up to 20,000 IOPS and 320 MB/s of throughput

    - Throughput Optimized HDD (st1)
        - Low cost magnetic storage
        - Throughput of up to 500 MB/s
        - Large, sequential workloads such as Amazon EMR, ETL, data warehouses, and log processing
    - Cold HDD (sc1)
        - Inexpensive magnetic
        - Throughput of up to 250 MB/s

AWS glacier

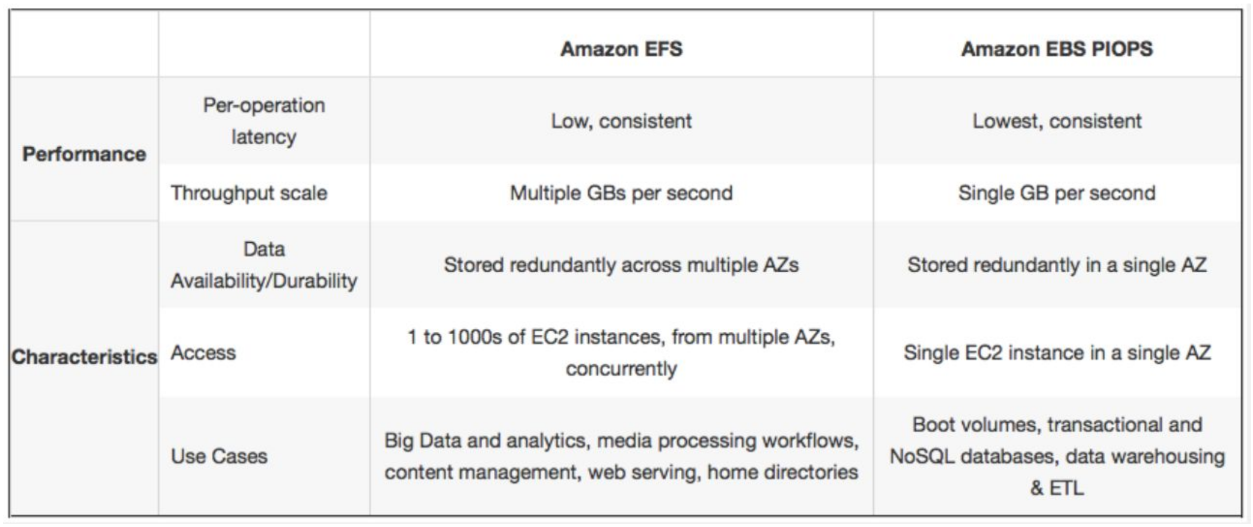
- Allows you to archive your data
- Very low cost, $0.007 per GB per month
- Very durable
    - Average annual durability of 99.999999999%
- Each single archive up to 40 TB
- Archives stored in vaults
- The main access point to glacier is S3
- Typically takes between 3 to 5 hours to prepare a download request
    - After that you have 24 hours to download from the staging location


Amazon Glacier Key Concepts
1
Create Vault
Configure Access Policies (Optional) via
Amazon Identity and Access Management
Configure Notification Policies (Optional) via
Amazon Simple Notification Service
AWS Management Console Operations
Also accessible via Amazon Glacier APIs or SDKs
2
Upload Archives
3
Retrieve Archives
Archives retrieved 3 - 5 hours after being requested
Initiate Job
Track Job
Download Job Output
Amazon Glacier API Operations
Also accessible via Amazon Glacier SDKs
Download Archives
Your Application
AWS
Notifications sent via Amazon SNS
amazon web services

AWS EFS

- Elastic File System
- Motivation: enterprise customers need a large distributed file system
    - S3 is large and distributed, but it is an object store without performance guarantees and eventual consistency model
    - Block storage (EBS, instance store) are small

        - Enterprise can build a distributed file system on top of these, but it requires operational expertise
    - Glacier: Good for only archival storage
- EFS provides a fully NFSv4 compliant network file system
- SSD backed
- Highly available and highly durable
    - files, directories, and links are stored redundantly across multiple Availability Zones within an AWS region
- Grow or shrink as needed
    - No need to pre-provision capacity

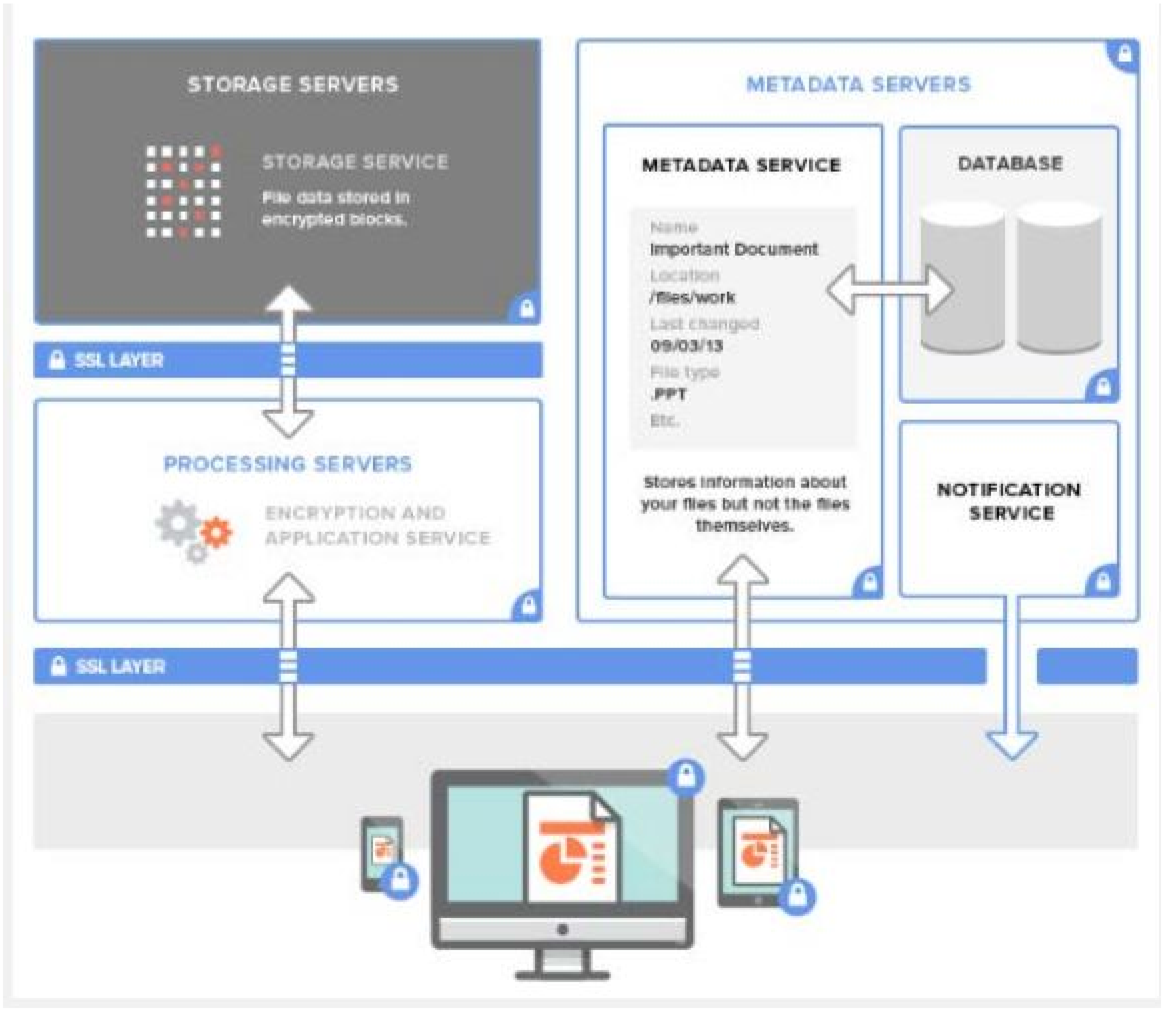


		Amazon EFS	**Amazon EBS PIOPS**
Performance	Per-operation latency	Low, consistent	Lowest, consistent
	Throughput scale	Multiple GBs per second	Single GB per second
Characteristics	Data Availability/Durability	Stored redundantly across multiple AZs	Stored redundantly in a single AZ
	Access	1 to 1000s of EC2 instances, from multiple AZs, concurrently	Single EC2 instance in a single AZ
	Use Cases	Big Data and analytics, media processing workflows, content management, web serving, home directories	Boot volumes, transactional and NoSQL databases, data warehousing & ETL

Dropbox API

Cloud Storage

- One interesting case study is Dropbox
- Dropbox offers cloud file storage
    - Easily synced across multiple devices
    - Accessible through web interface, mobile apps, and directly integrated with the file system on PCs
- Dropbox itself uses clouds!
    - Metadata stored in Dropbox servers
    - Actual files stored in Amazon S3
    - Amazon EC2 instances run the logic



Two levels of API access to Dropbox

- Drop-ins
    - Cross-platform UI components that can be integrated in minutes
    - Chooser allows instant access to files in Dropbox
    - Saver makes saving files to Dropbox easy
- Core API
    - Support for advanced functionality like search, provisions, and restoring file
    - Better fit for deeper integration

Drop-In API

- Simple objects
    - Chooser available for JavaScript, Android and iOS
    - Saver on web and mobile web
- Handles all the authentication (OAuth), file browsing
- Chooser object returns the following:
    - Link: URL to access the file
    - File name
    - File Size

        - Icon
        - Thumbnails
    - Saver
        - Pass in URL, filename and options

Core API

- Many languages and environments
    - Python, Ruby, PHP, Java, Android, iOS, OS X, HTTP
- Based on HTTP and OAuth
    - OAuth v1, OAuth v2
- Low-level calls to access and manipulate a user's Dropbox account
    - Create URL schemes
    - Upload files
    - Download files
    - List files and folders
    - Delta
    - Metadata access
    - Create and manage file sharing

Container

Operating-System-level virtualization allows us to run multiple isolated user-space instances in parallel. These user-space instances have the application code, the required libraries, and the required runtime to run the application without any external dependencies. These user-space instances are referred to as **containers**.

By using a container technology like Docker, we can bundle our application with all its dependencies in a box.

In the container world, this box (containing our application and all its dependencies) is referred to as an image. A running instance of this box is referred to as a container. We can spin multiple containers from the same image.
When a container is created from an image, it runs as a process on the host's kernel. It is the host kernel's job to isolate and provide resources to each container.

Namespaces
A namespace wraps a particular global system resource like network, process IDs in an abstraction, that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. The following global resources are namespaced:

- pid - provides each namespace to have the same PIDs. Each container has its own PID 1.
- net - allows each namespace to have its network stack. Each container has its own IP address.
- mnt - allows each namespace to have its own view of the filesystem hierarchy.

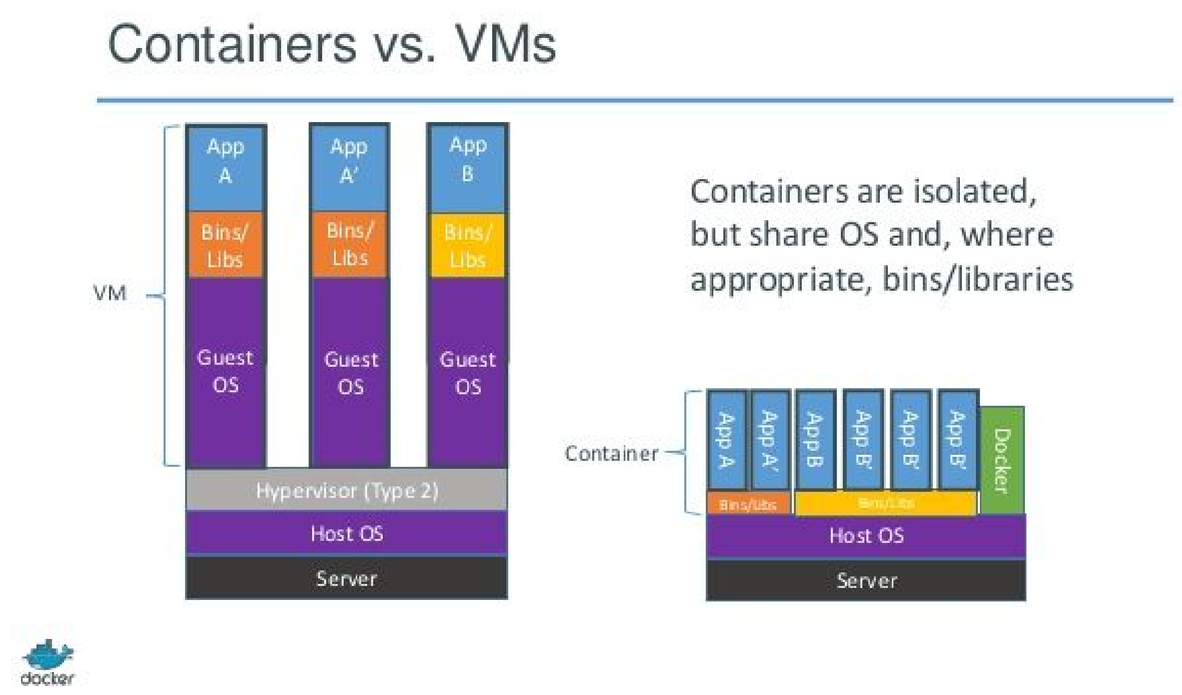
- ipc - allows each namespace to have its own interprocess communication.
- uts - allows each namespace to have its own hostname and domainname.
- user - allows each namespace to have its own user and group ID number spaces. A *root* user inside a container is not the *root* user of the host on which the container is running.

cgroups
Control groups are used to organize processes hierarchically and distribute system resources along the hierarchy in a controlled and configurable manner.
blkio, cpu, cpuacct, cpuset, devices, freezer, memory.
Union filesystem
The Union filesystem allows files and directories of separate filesystems, known as layers, to be transparently overlaid on each other, to create a new virtual filesystem. An image used in Docker is made of multiple layers and, while starting a new container, we merge all those layers to create a read-only filesystem. On top of a read-only filesystem, a container gets a read-write layer, which is an ephemeral layer and it is local to the container.
Docker hides all the complexities in the background and came up with an easy workflow to share and manage both images and containers. Some examples of container runtimes are **runC, containerd, rkt, CRI-O.**

Comparison between VM and containers

A virtual machine runs on top of a hypervisor, which emulates different hardware, like CPU, memory, etc., so that a guest OS can be installed on top of them. Different kinds of guest OSes can run on top of one hypervisor. Between an application running inside a guest OS and in the outside world, there are multiple layers: the guest OS, the hypervisor, and the host OS.

On the other hand, containers run directly as a process on top of the Host OS. There is no indirection as we see in VMs, which help containers to get near-native performance. Also, as the containers have very little footprint, we can pack a higher number of containers than VMs on the same physical machine. As containers run on the host OS, we need to make sure containers are compatible with the host OS.



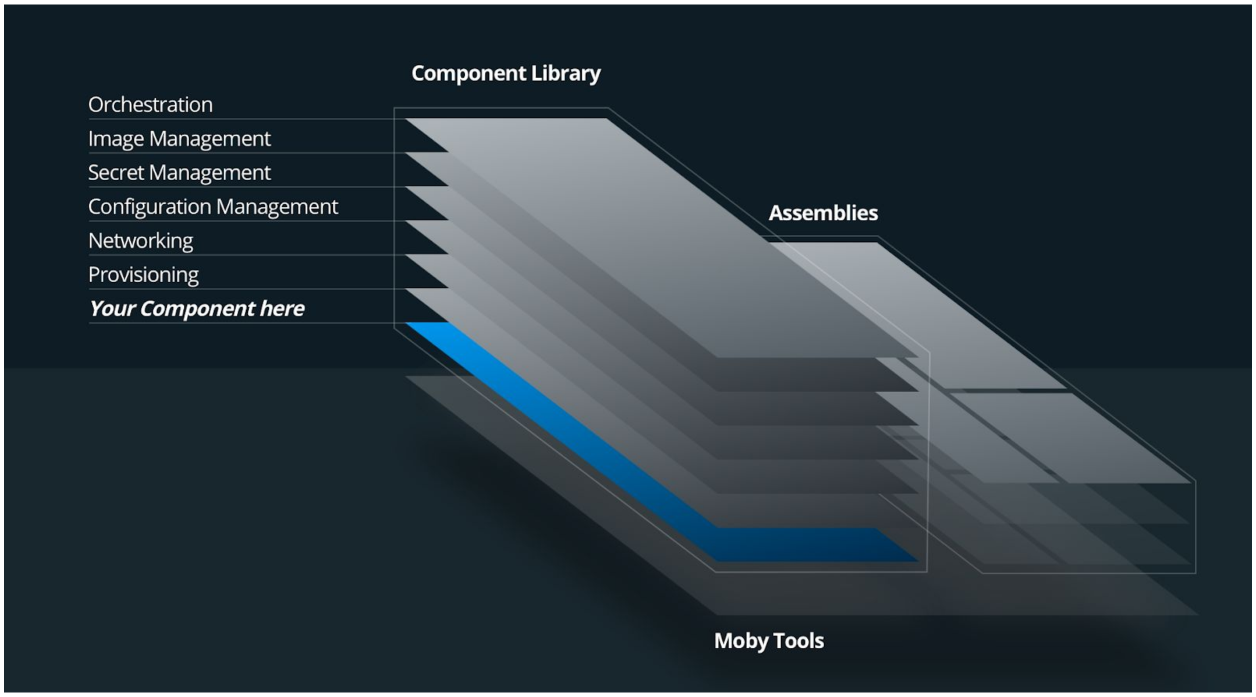
Containers vs. VMs

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

Benefits of using containers are:

- They have very little footprint.
- They can be deployed very fast (within milliseconds).
- They are a flexible solution, as they can run on any computer, infrastructure, or cloud environment.
- They can be scaled up or down with ease.
- There is a very rich ecosystem built around them.
- Problem containers can be easily and quickly isolated when troubleshooting and solving problems.
- Containers use less memory and CPU than VMs running similar workloads.
- Increased productivity with reduced overhead.

Project Moby is an open source project which provides a framework for assembling different container systems to build a container platform like Docker. Individual container systems provide features like image, container, secret management, etc.



Micro OSes for containers

The current technological trend is to run applications in containers. In this context, it makes a lot of sense to eliminate all the packages and services of the **host Operating System (OS)**, which are not essential for running containers. With that in mind, various vendors have come forward with specialized minimal OSes to run just containers. Examples are the following,

- Alpine Linux
- Atomic Host
- Fedora CoreOS (formerly known as Red Hat CoreOS)
- RancherOS
- Ubuntu Core
- VMware Photon.

Alpine Linux

Alpine Linux is an independent, non-commercial, Linux distribution designed for security, simplicity and resource efficiency.

Although small at about 8 MB per container, it is more resource-efficient than typical distributions. Users can control what binary packages to install, thus ensuring a small yet efficient system.

Alpine Linux uses its own package manager called apk, the OpenRC init system, and set-up scripts. Users can add packages as needed such as PVR, iSCSI storage controllers, a mail server container, or an embedded switch.

Alpine Linux was designed with security in mind with embedded proactive security features that prevent the exploitation of entire classes of zero-day and other vulnerabilities.

Upon installation completion, Alpine Linux makes available tools for the initial configuration of the system. Once prepared for reboot, it can be configured to boot in one of the three available runtime modes:

- diskless mode
  The default mode, where the entire system runs from RAM.
- data mode
  Mostly runs from RAM but mounts /var as a writable data partition.
- sys mode
  The typical hard-disk install that mounts /boot, swap, and / .

Benefits

- It is a minimal OS designed to run containerized applications as well.
- It is designed for security, simplicity, and resource efficiency.
- It requires 8 MB as a container.
- It requires 130 MB as a standalone minimal OS installation.
- It provides increased security by compiling user binaries as Position Independent Executables (PIE) with stack smashing protection.
- It can be installed as a container, on bare metal, as well as VMs.
- It offers flavors optimized to support Xen and Raspberry Pi.

Benefits of Atomic Host

- It is an OS specifically designed to run containerized applications.
- It provides close-to-VM like isolation but increased flexibility and efficiency.
- It enables us to perform quick updates and rollbacks.
- It provides increased security through namespaces, cgroups, and SELinux.
- It can be installed on bare metal, as well as VMs.
- It is available in Fedora, CentOS, and Red Hat Enterprise Linux editions.
- Containers can be deployed with Kubernetes and CRI-O.
- It integrates with Cockpit, a cross-cluster container hosts management too

Benefits of Fedora CoreOS

- It is an OS designed to run containerized applications, in both clustered environment or as stand-alone.
- It enables us to perform quick updates and rollbacks.
- It provides increased security through SELinux.

- It can be installed on bare metal, virtual environments, and the cloud, or launched directly on AWS.
- It combines features of both Fedora Atomic Host and CoreOS Container Linux.
- It works well with Kubernetes.
- It uses Ignition as a provisioning tool for early boot disk partitioning, formatting, and other administrative configuration tasks.

Key benefits of using RancherOS are:

- It is a minimalist OS, by eliminating unnecessary libraries and services.
- It decreases complexity and boot time.
- RancherOS is the Micro OS with the least footprint.
- It is highly secure due to a small code base and a decreased attack surface.
- It runs directly on top of the Linux kernel.
- It isolates user-level containers from system containers by running two separate Docker daemon instances.
- It enables us to perform updates and rollbacks in a simple manner.
- We can use the Rancher platform to set up Kubernetes.
- It boots containers within seconds.
- It automates OS configuration with cloud-init.
- It can be customized to add custom system Docker containers using the cloud-init file or Docker Compose.

Key benefits of using Ubuntu Core are:

- It has one of the smallest footprints of all Micro OSes available.
- It supports automated updates and rollbacks.
- It is highly resilient.
- It boots the containers within seconds.
- It is highly secure and extremely reliable.
- It provides application isolation through AppArmor and Seccomp.
- It integrates with CI/CD pipelines.

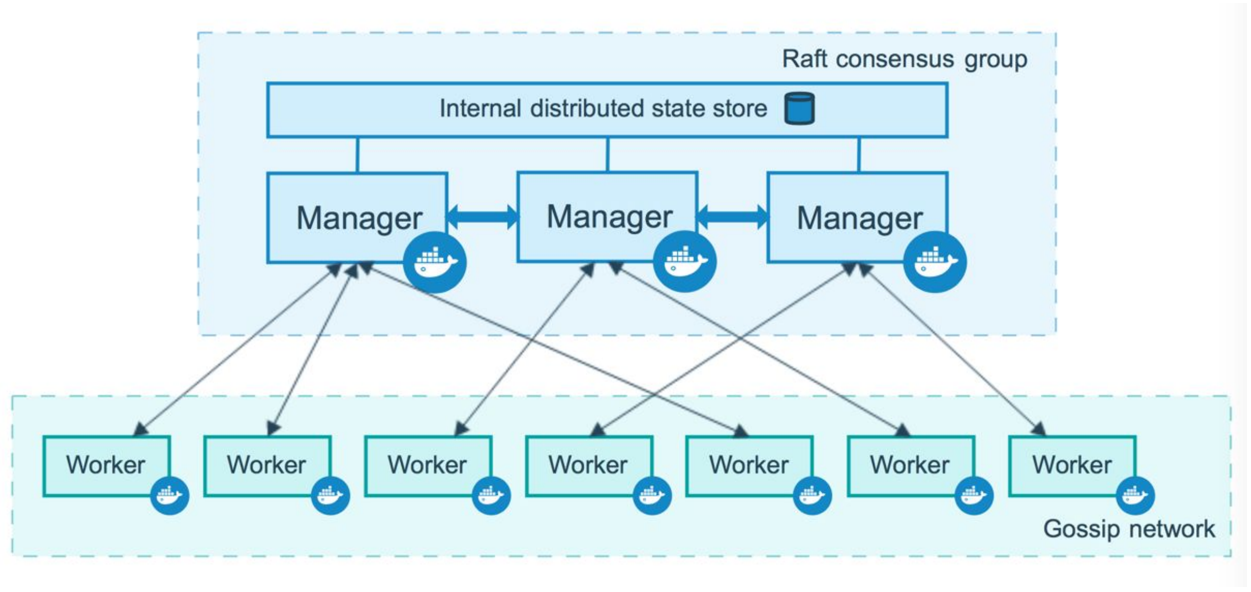
Key benefits of using VMware Photon are:

- It is an open source technology with a small footprint.
- It supports Docker, rkt, and Cloud Foundry Garden container runtimes.
- It includes Kubernetes in the full version, to allow for container cluster management, but it also supports Mesos.
- It boots extremely quickly on VMware platforms.
- It provides efficient lifecycle management with a yum-compatible package manager.
- Its kernel is tuned for higher performance when it is running on VMware platforms.

- It is a security-enhanced Linux as its kernel and other aspects of the operating system are configured according to the security parameter recommendations given by the Kernel Self-Protection Project.

Docker Swarm

Docker Swarm is a native container orchestration solution from Docker, Inc. Docker, in swarm mode, logically groups multiple Docker Engines into a swarm, or cluster, that allows for applications to be deployed and managed at scale.



The above illustration depicts two major components of a swarm:

- Swarm Manager Nodes
  Accept commands on behalf of the cluster and make scheduling decisions. They also maintain the cluster state and store it using the Internal Distributed State Store, which uses the Raft consensus algorithm. One or more nodes can be configured as managers for fault-tolerance. When multiple managers are present they are configured in active/passive modes.
- Swarm Worker Nodes
  Run the Docker Engine and the sole purpose of the worker nodes is to run the container workload dispatched by the manager node(s).

Key characteristics of Docker Swarm:

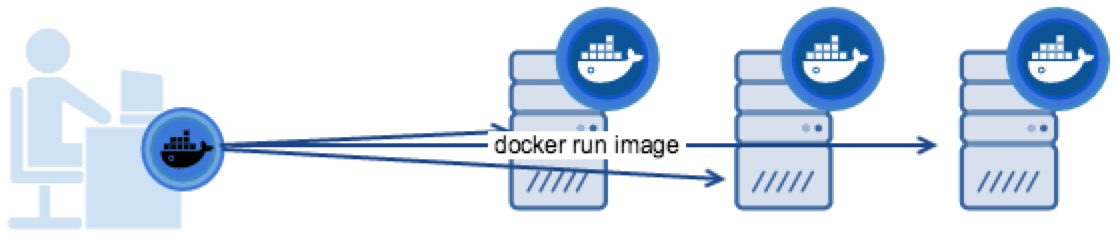
It is compatible with Docker tools and API so that the existing workflow does not change much.

- It provides native support to Docker networking and volumes.
- It can scale up to large numbers of nodes.
- It supports failover and High Availability for the cluster manager for fail-tolerance.
- It uses a declarative approach to define the desired state of the various
- services of the application stack.

- For each service, you can declare the number of tasks you want to run. When you scale up or down, the Swarm manager automatically adapts by adding or removing tasks to maintain the desired state.
- The Docker Swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state.
- The communication between the nodes of Docker Swarm is enforced with Transport Layer Security (TLS), which makes it secure by default.
- It supports rolling updates to control a delay between service deployment to different sets of nodes. If a task rollout is unsuccessful, you can roll back a task to a previous version of the service.

Docker machine

Docker Machine helps us configure and manage one or more Docker Engines running locally or on cloud environments. With Docker Machine we can start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to our host.

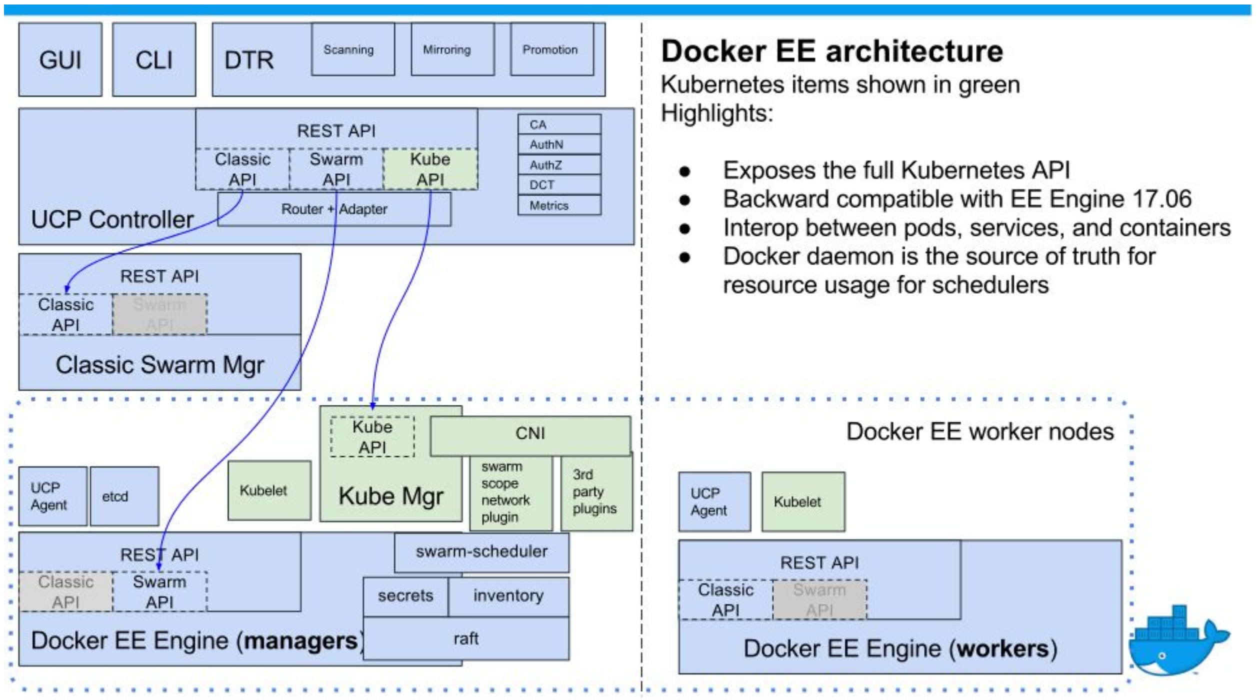


Docker Compose allows us to define and run multi-container applications through a YAML configuration file. In this configuration file, we can define resources such as services, images, Dockerfiles, network, and storage. Below we provide a sample of a Compose file:
Docker Swarm uses *docker stack* to deploy complete distributed application stacks. We can use Docker Compose to generate the stack file, which can be used to deploy the application stack to the swarm.
Benefits of Docker Swarm

- It provides native clustering for Docker.
- It is well integrated with the existing Docker tools and workflow.
- Its setup is easy, straightforward and flexible.
- It manages containerized workload in a cluster of Docker Engines that are treated as a single entity.
- It provides scalability and supports High Availability.
- Efficiency and productivity are increased by reducing deployment and management time, as well as duplication of efforts.

Docker Enterprise Platform is a Container-as-a-Service (CaaS) platform that manages the entire lifecycle of the applications on enterprise Linux or Windows operating systems and Cloud providers. At the time of this writing, the Docker Enterprise Platform has been acquired by Mirantis and is part of their technology stack offered as-a-service. It supports Docker Swarm and Kubernetes as container orchestrators. It has the following three major components:

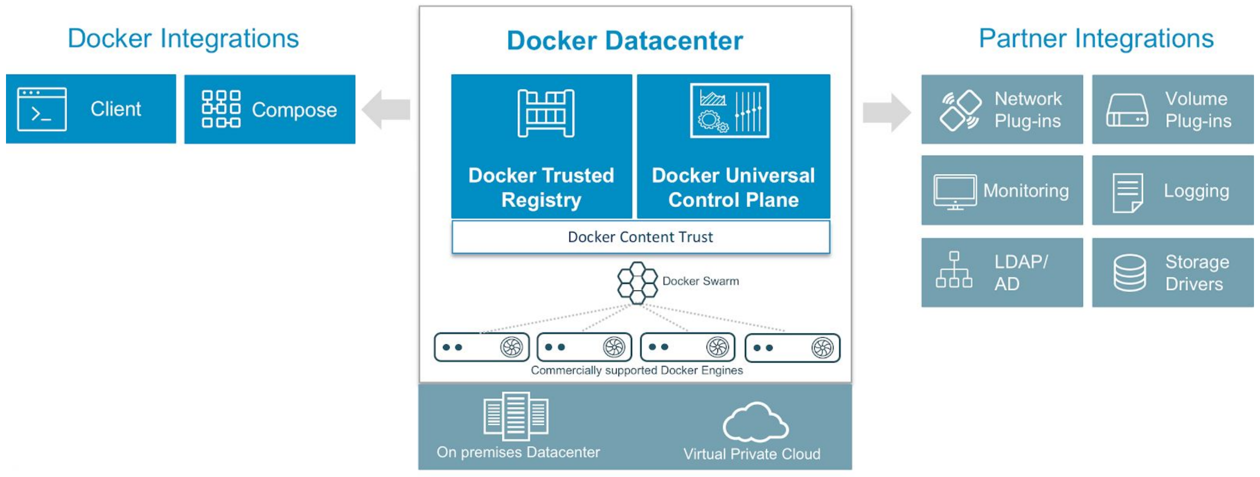
- Docker Engine - Enterprise
  It is a commercially supported Docker Engine for creating images and running Docker containers.
- Docker Trusted Registry (DTR)
  It is a production-grade image registry designed to store images, from Docker, Inc.
- Universal Control Plane (UCP)
  It manages the Kubernetes and Swarm orchestrators, it deploys applications using the CLI and GUI, and supports High Availability. UCP also provides role-based access control (RBAC) to ensure that only authorized users can make changes and deploy applications to your cluster.



Benefits

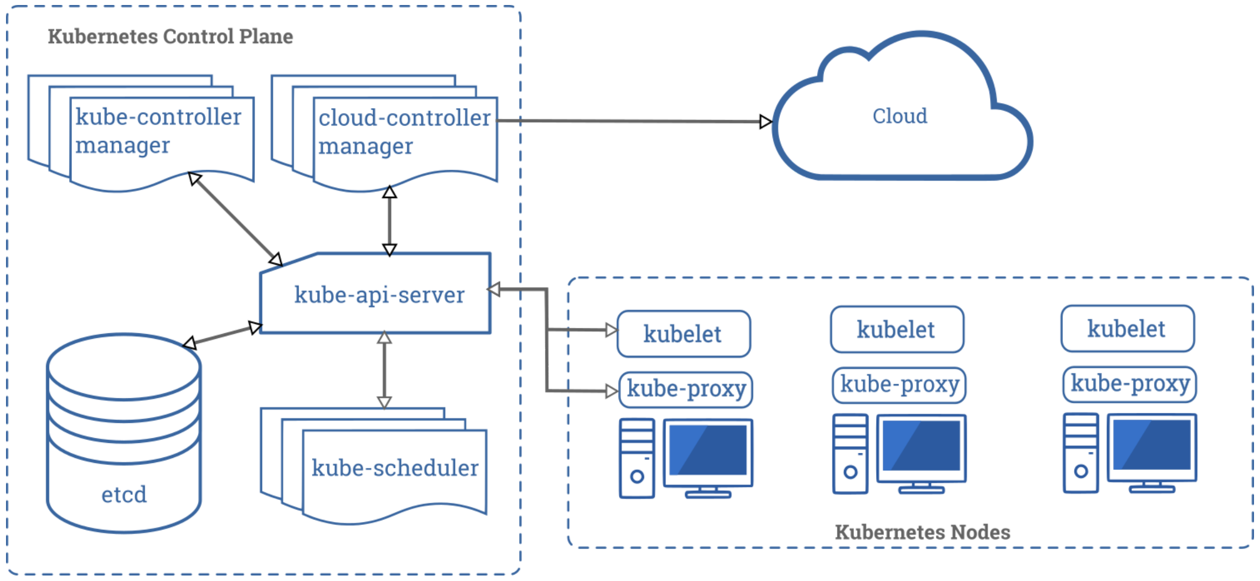
- It is a multi-Linux, multi-OS, multi-Cloud solution.
- It supports Docker Swarm and Kubernetes as container orchestrators.
- It provides centralized cluster management.
- It has a built-in authentication mechanism with role-based access control (RBAC).

The Docker Datacenter, which is built on top of the Universal Control Plane (UCP) and Docker Trusted Registry (DTR). Docker Datacenter is an enterprise container management and deployment service hosted on-premise behind a firewall or in the cloud.
With Docker Datacenter we can build an enterprise-class CaaS platform on-premises, as it is built on top of Swarm and integrates well with Docker tools, and the Docker Registry. In addition, it provides LDAP and AD integration, monitoring, logging, and plugin support for network and storage.



Kubernetes

Kubernetes is an Apache 2.0-licensed open source project for automating deployment, operations, and scaling of containerized applications. It was started by Google in 2014, but many other companies like Docker, Red Hat, and VMware contributed to its success.s Kubernetes supports several container runtimes such as Docker, CRI-O, frakti, and rkt to run containers.



The key components of the Kubernetes architecture are:

- Cluster
  The cluster is a group of systems (bare-metal or virtual) and other infrastructure resources used by Kubernetes to run containerized applications.
- Master Node
  The master is a system that takes pod scheduling decisions and manages the worker

nodes. Its main components are kube-apiserver, etcd, kube-scheduler, and kube-controller-manager and they coordinate tasks around container workload scheduling, deployment, scaling, self-healing, state persistence, and delegation of other tasks to worker node agents. Multiple master nodes may be found in clusters as a solution for High Availability.

- Worker Node
  A system on which containers are scheduled to run in pods. The node runs a daemon called kubelet to communicate with the master node by reporting node status information and receiving instructions from the master about container lifecycle management (this daemon is also found on master nodes). kube-proxy, a network proxy running on all nodes, allows applications running in the cluster to be accessed from the external world.
- Namespace
  The namespace allows us to logically partition the cluster into virtual sub-clusters, for projects, applications, users, and teams' isolation.

The key API resources of the Kubernetes architecture:

- Pod
  The pod is a co-located group of containers with shared volumes, however, it often manages a single container. It is the smallest deployment unit in Kubernetes. A pod can be created independently, but it is recommended the use of controllers such as the ReplicaSet or Deployment, even if only a single pod is being deployed.
- ReplicaSet
  The ReplicaSet is a mid-level controller that manages the lifecycle of pods. It ensures that the desired number of pods is running at all times.
- Deployment
  The Deployment is a top-level controller that allows us to provide declarative updates for pods and ReplicaSets. We can define Deployments to create new resources, or replace existing ones with new ones.
- Service
  The service groups sets of pods together and provides a way to refer to them from a single static IP address and the corresponding DNS name. It can reference a single pod, a group of individual pods, or pods managed by ReplicaSets or Deployments.
- Label
  The label is an arbitrary key-value pair that is attached to a resource like a pod or a ReplicaSet. In the example above, we defined labels as app and tier.
- Selector
  Selectors enable us to group resources based on labels. In the above example, the frontend service will select all pods which have the labels *app==dockchat* and *tier==frontend*.
- Volume
  The volume is an external filesystem or storage which is available to pods and is mounted on a container's filesystem. They are built on top of Docker volumes.

Key features of Kubernetes are:

- It automatically distributes containers on cluster nodes based on containers' resource requirements and other custom constraints.
- It supports horizontal scaling through the CLI or a UI. In addition, it can auto-scale based on resource utilization.
- It supports rolling updates and rollbacks.
- It supports several volume plugins from public cloud providers such as AWS, Azure, and GCP together with network storage plugins like NFS, iSCSI, CephFS, GlusterFS, Cinder, Flocker, and vsphereVolume to orchestrate storage volumes for containers running in pods.
- It automatically self-heals by restarting failed containers, rescheduling containers from failed nodes, and supports custom health checks to ensure containers are continuously ready to serve.
- It manages sensitive and configuration data for an application without rebuilding the image.
- It supports batch execution.
- It supports High Availability of the master node to add control plane resiliency.
- It eliminates infrastructure lock-in by providing core capabilities for containers without imposing restrictions.
- It supports application deployments and updates at scale.
- It allows services to be routed based on the topology of the cluster.

Benefits of Kubernetes

- It is an open source system that packages all the necessary features: orchestration, service discovery, and load balancing.
- It manages multiple containers at scale.
- It automates deployment, scaling, and operations of application containers.
- It is portable, extensible, and self-healing.
- It provides consistency across development, testing, and production environments, on-premise and across clouds.
- It is highly efficient when utilizing resources.

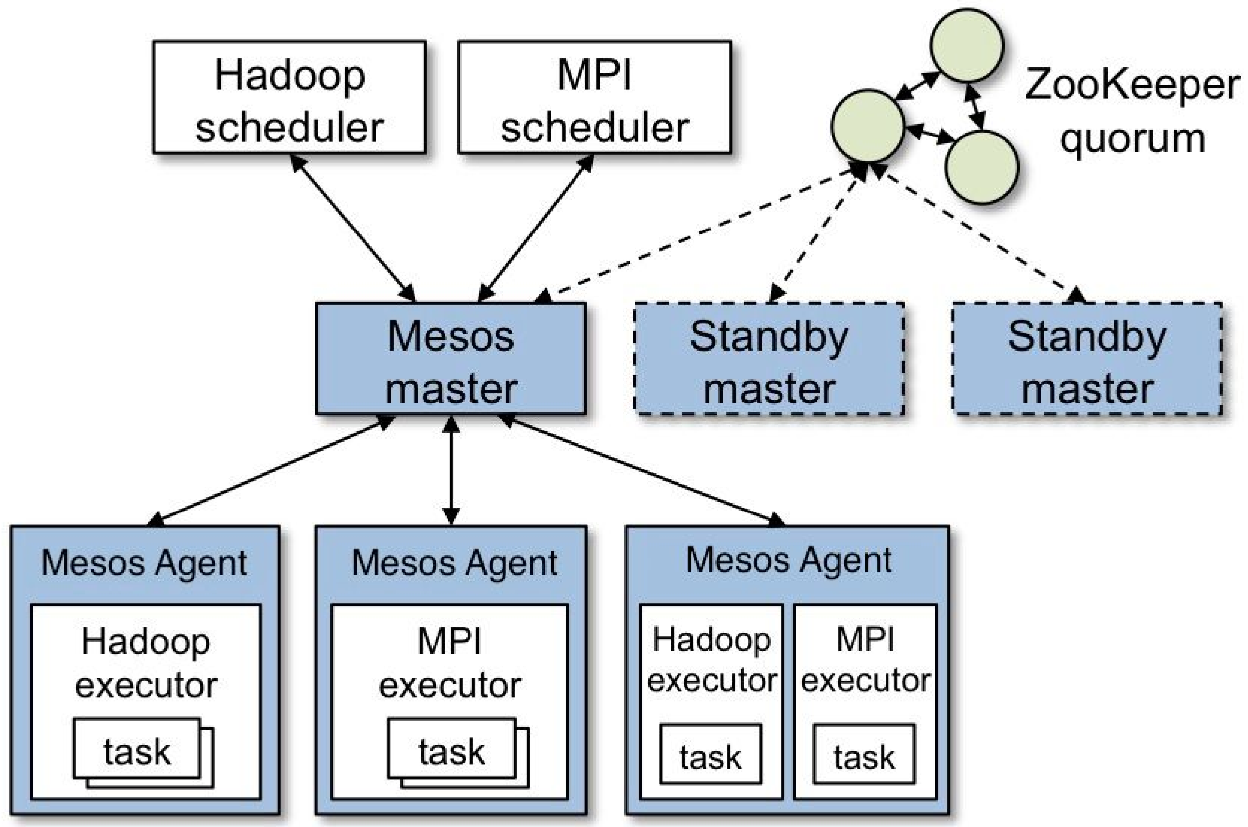
Apache Mesos

When we install and set up a physical machine, we generally use it for very specific purposes, such as running applications or frameworks as Hadoop, Spark, containers, or Jenkins. Some applications might not be using all the system resources (e.g. CPU, memory) while others might be starving for more. Therefore, it would be helpful to have the ability to **combine all the physical resources** across multiple machines and execute tasks on specific machines based on resource requirements.

Mesos is a distributed systems kernel that runs on every machine and introduces a level of abstraction that exposes APIs for resource management and workload scheduling. It helps us treat a cluster of nodes as a single compute unit, which manages CPU, memory, and other cluster resources provisioned across datacenter and cloud environments.

Mesos provides functionality that crosses between Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). It is an open source Apache project.
The key components of the Mesos Architecture:

- Master
  Master nodes are the brain of the cluster and provide a single source of truth for running tasks. There is one active master node at any point in time. The master node mediates between schedulers and slaves. Slaves advertise their resources to the master node, then the master node forwards them to the scheduler. Once the scheduler accepts the offer, it sends the task to run on the slave to the master, and the master forwards these tasks to the slave.
- Slave
  Slaves manage resources at each machine level and execute the tasks submitted via the scheduler.
- Frameworks
  Frameworks are distributed applications that solve a specific use case. They consist of a scheduler and an executor. The scheduler receives a resource offer, which can be accepted or declined. The executor runs the job on the slave, which was scheduled by the scheduler. There are many existing frameworks and we can also create custom ones. Some of the existing frameworks are Hadoop, Spark, Marathon, Chronos, Jenkins, and Aurora.
- Executor
  Executors, that run jobs on slaves, can be shell scripts, Docker containers, or programs written in different languages such as Java.



The key features of Mesos are:

- It can easily scale up to 10,000 nodes.
- It uses ZooKeeper for fault-tolerant replicated master and slaves.
- It provides support for Docker containers.
- It enables native isolation between tasks with Linux containers.
- It allows multi-resource scheduling (memory, CPU, disk, and ports).
- It is cross-platform and cloud provider agnostic.
- It uses Java, Python, and C++ APIs to develop new parallel applications.
- It uses WebUI to view cluster statistics.
- It allows for high resource utilization.
- It helps to handle mixed workloads.
- It provides an easy-to-use container orchestration right out of the box.

Mesosphere offers a commercial solution on top of Apache Mesos. Mesosphere is one of the primary contributors to the Mesos project and to frameworks like Marathon. Their commercial product, Mesosphere Distributed Cloud Operating System (DC/OS), offers a one-click installation together with enterprise features like security, monitoring, user interface, on top of Mesos.

Distributed Cloud Operating System (DC/OS) has recently been open-sourced by Mesosphere. DC/OS has the following features

- It provides an easy-to-use container orchestration right out of the box.
- It can configure multiple resource isolation zones.

- It can support applications with multiple persistent and ephemeral storage options.
- It allows you to install both public and private community packaged applications.
- It allows you to manage your cluster and services using the web and command-line interfaces.
- It allows you to easily scale up and scale down your services.
- It provides automation for updating services and the systems with zero downtime.
- Its Enterprise edition provides centralized management, control plane for service availability and performance monitoring.

Benefits of Mesos

- It is an open source solution, that also has a commercial version available.
- It provides support for Docker containers.
- It allows multi-resource scheduling across several platforms and cloud providers.
- It is highly available and scalable.
- It provides service discovery and load balancing.

Nomad

HashiCorp Nomad is a cluster manager and resource scheduler from HashiCorp, which is distributed, highly available, and scales to thousands of nodes. It is designed to run microservices and batch jobs, and it supports different types of workloads, from containers (Docker) and VMs, to individual applications. In addition, it is capable of scheduling applications and services on different platforms like Linux, Windows, and Mac, both on-premise and clouds.

It is distributed as a single binary, which has all of its dependency and runs in a server and client mode. To submit a job, the user has to define it using a declarative language called HashiCorp Configuration Language (HCL) with its resource requirements. Once submitted, Nomad will find available resources in the cluster and run it to maximize resource utilization.

Features

- It handles both cluster management and resource scheduling.
- It supports multiple workloads, like containers (Docker), VMs, unikernels, and individual applications.
- It has multi-datacenter and multi-region support. We can have a Nomad client/server running in different clouds, while still part of the same logical Nomad cluster.
- It bin-packs applications onto servers to achieve high resource utilization.
- In Nomad, millions of containers can be deployed or upgraded by using the job file.
- It provides a built-in dry run execution facility, which shows the scheduling actions that are going to take place.
- It ensures that applications are running in failure scenarios.
- It supports long-running services, as well as batch jobs and cron jobs.
- It provides a built-in mechanism for rolling upgrades.
- Blue-green and canary deployments are supported through a declarative job file syntax.
- If nodes fail, Nomad automatically redistributes the applications from unhealthy nodes to healthy nodes.

Benefits of Nomad

- It is an open source solution that is distributed as a single binary for *servers* and *agents*.
- It is highly available and scalable.
- It maximizes resource utilization.
- It provides multi-datacenter and multi-region support.
- It supports multi-cloud federation.
- During maintenance, there is zero downtime to datacenter and services.
- It simplifies operations, provides flexible workloads, and fast deployment.
- It can integrate with the entire HashiCorp ecosystem of tools such as Terraform, Consul, and Vault for provisioning, service discovery, and secrets management.
- It can support a cluster size of more than ten thousand nodes.

Benefits of Amazon Elastic Kubernetes Service (EKS)

- Users do not manage the Kubernetes control plane.
- It provides secure communication between the worker nodes and the control plane.
- It supports auto-scaling in response to changes in load.
- It integrates well with various AWS services such as IAM and CloudTrail.
- It is a Certified hosted Kubernetes platform.

Key features and benefits of the Azure Kubernetes Service (AKS) are listed below:

- Users do not manage the Kubernetes Control Plane.
- It supports GUI and CLI-based deployment.
- It integrates well with other Azure services.
- It is a Certified hosted Kubernetes platform.
- It is compliant with SOC and ISO/HIPAA/HITRUST.

Benefits of Google Kubernetes Services

- It has all of Kubernetes' features.
- It runs on a container-optimized OS built and managed by Google.
- It is a fully-managed service, so the users do not have to worry about managing and scaling the cluster.
- We can store images privately, using a private container registry.
- Logging can be enabled easily using Google Cloud Logging.
- It supports Hybrid Networking to reserve an IP address range for the container cluster.
- It enables fast setup of managed clusters.
- It facilitates increased productivity for Dev and Ops teams.
- It is Highly Available in multiple zones and SLA promises 99.5% of availability.
- It has Google-grade managed infrastructure.
- It can be seamlessly integrated with all GCP services.
- It provides a feature called Auto Repair, which initiates a repair process for unhealthy nodes.

Key features of Amazon Elastic Container Service (ECS) are:

- It is compatible with Docker.
- It provides a managed cluster so that users do not have to worry about managing and scaling the cluster.

- The task definition allows the user to define the applications through a .json file. Shared data volumes, as well as resource constraints for memory and CPU, can also be defined in the same file.
- It provides APIs to manage clusters, tasks, etc.
- It allows easy updates of containers to new versions.
- The monitoring feature is available through AWS CloudWatch.
- The logging facility is available through AWS CloudTrail.
- It supports third-party Docker Registry or Docker Hub.
- AWS Fargate allows you to run and manage containers without having to provision or manage servers.
- It allows you to build all types of containers. You can build a long-running service or a batch service in a container and run it on ECS.
- You can apply your Amazon Virtual Private Cloud (VPC), security groups, and AWS Identity and Access Management (IAM) roles to the containers, which helps maintain a secure environment.
- You can run containers across multiple availability zones within regions to maintain High Availability.
- It can be integrated with AWS services like Elastic Load Balancing (ELB), Virtual Private Cloud (VPC), Identity and Access Management (IAM), Amazon ECR, AWS Batch, Amazon CloudWatch, AWS CloudFormation, AWS CodeStar, AWS CloudTrail, and more.

Key benefits of Amazon ECS are:

- It provides a managed cluster.
- It is built on top of Amazon Elastic Compute Cloud (EC2).
- It is highly available and scalable.
- It leverages other AWS services, such as CloudWatch Metrics.
- We can manage it using CLI, Dashboard, or APIs.

Key features of Azure Container Instances (ACI) are:

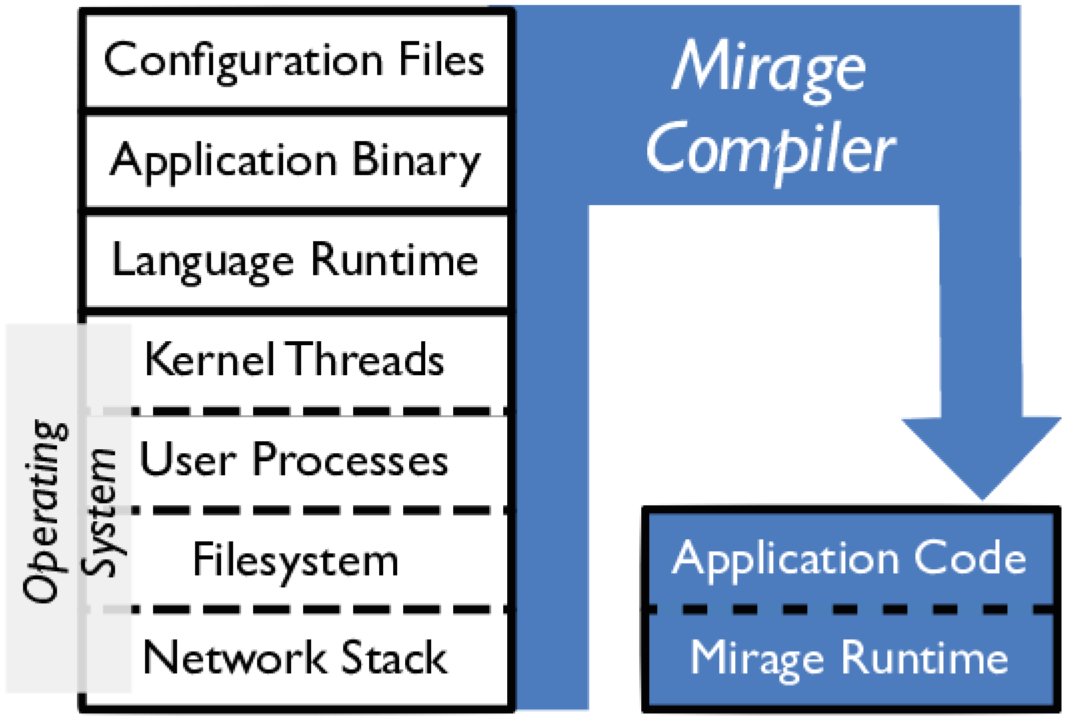
- They expose containers directly to the internet through IP addresses and fully qualified domain names (FQDN).
- Allow user interaction with the environment of a running container by executing commands in the container through a shell.
- Offer VM-like application isolation in the container.
- Allow for resource specification, such as CPU and memory.
- They allow containers to mount directly Azure File shares to persist their state.
- They support the running of both Linux and Windows containers.
- They support the scheduling of single- and multi-container groups, thus allowing patterns like the sidecar pattern.

UniKernel

In most cases, the majority of the libraries would not be consumed by the application. Therefore, it makes sense to ship the application only with the set of user-space libraries which are needed by the application.
Unikernels are specialised, single-address-space machine images constructed by using library operating systems. With **unikernels**, we can also select the part of the kernel needed to run with the specific application. With unikernels, we can create a single address space executable, which has both application and kernel components. The image can be deployed on VMs or bare metal, based on the unikernel's type.
UniKernel creates specialized virtual machine images with just:

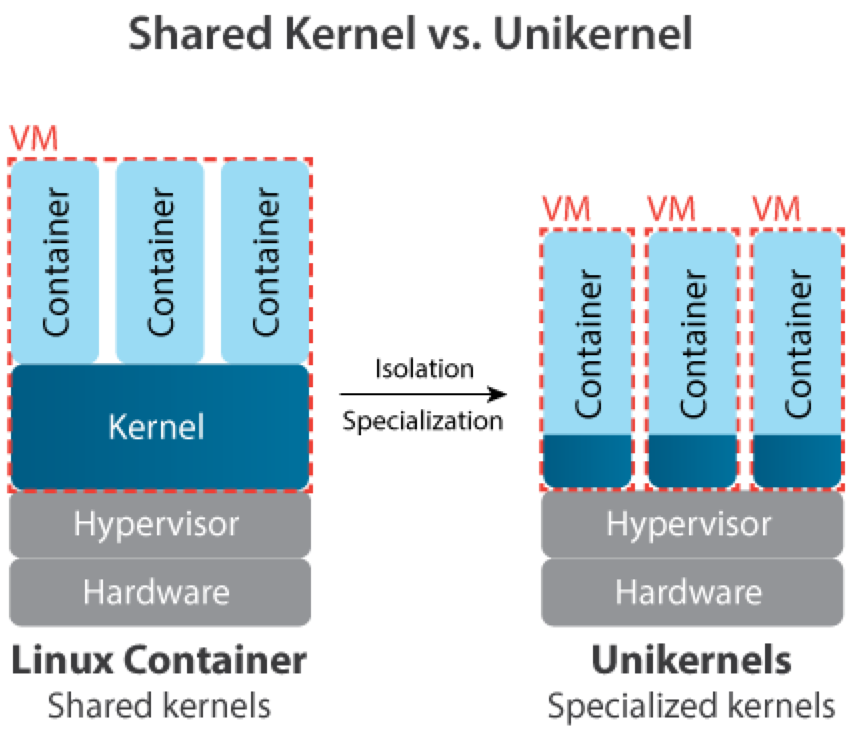
- The application code
- The configuration files of the application
- The user-space libraries needed by the application
- The application runtime (like JVM)
- The system libraries of the unikernel, which allow back and forth communication with the hypervisor.

According to the protection ring of the x86 architecture, we run the kernel on ring0 and the application on ring3, which has the least privileges. Ring0 has the most privileges, like access to hardware, and a typical OS kernel runs on that. With unikernels, a combined binary of the application and the kernel runs on ring0.
Unikernel images would run directly on top of a hypervisor like Xen or on bare metal, based on the unikernel types. The following image shows how the Mirage Compiler creates a unikernel VM image.



The following are key benefits of unikernels:
- A minimalistic VM image to run an application, which allows us to have more applications per host.
- A faster boot time.
- Efficient resource utilization.
- A simplified development and management model.
- A more secure application than the traditional VM, as the attack surface is reduced.
- An easily-reproducible VM environment, which can be managed through a source control system like Git.

Both containers and unikernels can co-exist on the same host and can be managed by the same Docker binary.
Unikernels helped Docker to run the Docker Engine on top of Alpine Linux on Mac and Windows with their default hypervisors, which are xhyve Virtual Machine and Hyper-V VM respectively.


Shared Kernel vs. Unikernel
VM
Container
Container
Container
Kernel
Hypervisor
Hardware
Linux Container
Shared kernels
Isolation
Specialization
VM
VM
VM
Container
Container
Container
Hypervisor
Hardware
Unikernels
Specialized kernels