Introduction
Cloud Computing

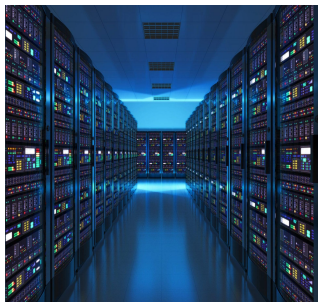
Boris TEABE
boris.teabe@inp-toulouse.fr

Goals

1. General introduction
2. Types of Cloud
3. Types of Cloud services
4. Some Cloud providers

Cloud computing the idea

- **Amazon, 2002**
  - Rent to external users a part part of his facilities during period of low use
  - Creation of Amazon Web Services (AWS) initially for data storage then for compute



Cloud computing the idea

The idea basic idea, **mutualization**

- **My Car**
  - Rent to people not having a car
  - Reduce the Cost of ownership

- **Harvester**
  - Cost a Lot of Money
  - Reduce the cost when shared

Cloud Computing Definition

Set of Resources/applications/services which execute in a distributed environment (hosting center), accessible through Web standard protocols, which globally provide a service with the following characteristics:

- Pays as you Go
- Illusion of infinity of ressources
- Abstraction of the hardware infrastructure
- Mutualization between many users

Cloud Computing Definition

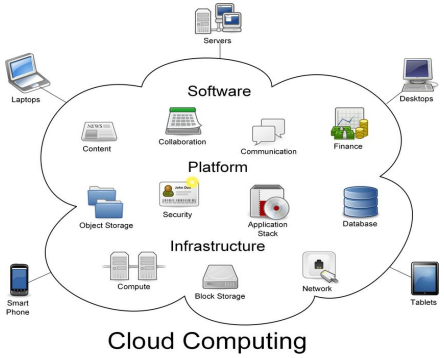
NIST Definition of Cloud Computing

"Cloud computing is a model for enabling 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."

NIST (National Institte of Standards and Technology), *https://www.nist.gov/*

Cloud Computing Definition

General scheme is externalization of resources



Cloud computing

- Enormous **Computer data-centers (DC)** containing Commodity hardwares
- Achieve economies of scale.
  - Reduce costs of electricity, bandwidth, hardware, software and use low-cost locations
  - Lower-cost than provisioning own hardware

Cloud computing

- Cloud computing is **Utility Computing**
  - Cloud services are controlled and monitored by the cloud provider through a **pay-per-use business model**.
- An ideal cloud computing platform is:
  - Efficient in its use of resources
  - Scalable
  - Elastic
  - Self-managing
  - Highly available and accessible
  - Inter-operable and portable

Utility computing is a service provisioning model in which a service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a flat rate.

Roles in the Cloud

- **Cloud provider**
  - Provides hardware infrastructures and a set of services on top
- **Cloud Clients**
  - Use resources of the Cloud
- **Cloud resellers**
  - Build and Sells a Service that rely on the Cloud.
- **Cloud developers**
  - Produce tools for the Cloud

Types of Cloud

- There are **four primary cloud deployment models** :
  - Public Cloud
  - Private Cloud
  - Community Cloud
  - Hybrid Cloud

Public Cloud

- Public clouds are owned by cloud service providers who charge for the use of cloud resources.
- Basic characteristics:
  - Shared resources and multi-tenancy
  - Leased or rented infrastructure
  - Economies of scale

Private Cloud

- The cloud infrastructure belongs to and is operated by only one organization.
- Basic characteristics :
  - Heterogeneous infrastructure
  - Customized policies
  - Dedicated resources
  - In-house infrastructure
  - End-to-end control

EUCALYPTUS

Open Nebula

openstack.

Community Cloud

- The cloud is built to be shared by several organizations for their usage

Grid'5000

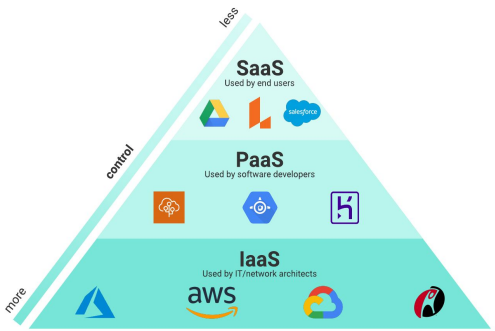
OSTRIM

Hybrid Cloud

- The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability.

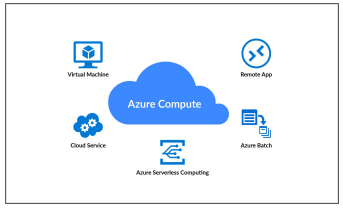
Service Classifications

- **Infrastructure as a Service (IaaS)**
- **Platform as a Service (PaaS)**
- **Software as a Service (SaaS)**



Infrastructure as a Service (IaaS)

- A type of cloud computing service that offers essential compute, storage, and networking resources on demand, on a pay-as-you-go basis
  - **Virtualization**
  - **Bare metal**



Platform as a Service (PaaS)

- A computing platform that abstracts the infrastructure, OS, and middleware to drive developer productivity

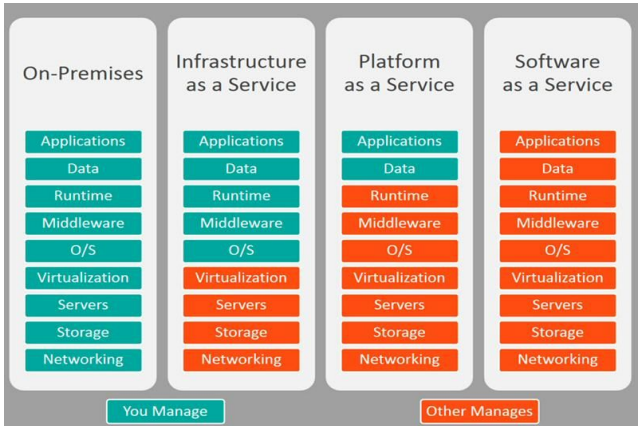
Software as a Service (SaaS)

- Run applications on a provider's on a cloud infrastructure. Applications are accessible from various client devices through a thin client interface such as a web browser.

Serverless, Function as a Service (More later)

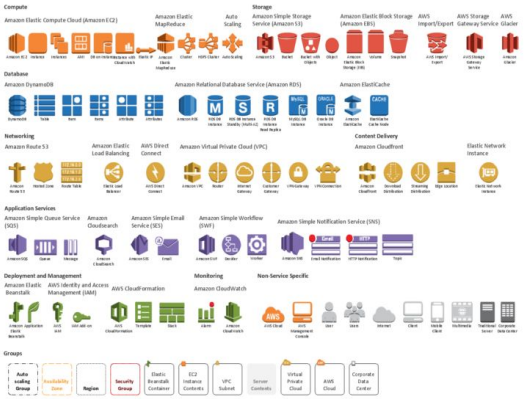
➔ Serverless is an event-driven or code execution where underlying infrastructure is abstracted. Applications consists of stateless functions that are spawned based on the triggering of events.

Cloud Services



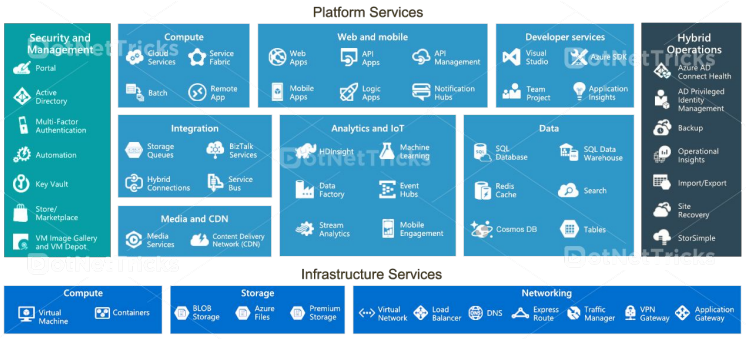
Some Cloud providers

- **Amazon AWS**
  - Public Cloud
  - Most popular cloud provider



Some Cloud providers

- **Microsoft Azure**
  - Public Cloud
  - Provides several services



Some Cloud providers

- French Cloud providers
  - OVHCloud
  - Eolas
  - OutScale
  - ScaleWay

eolas GROUPE BUSINESS & DECISION

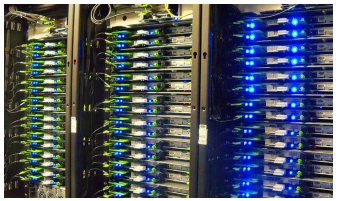
OVH.com

OUTSCALE

Scaleway

Data-centers (DC)

- Scaled-up/out version of machine rooms for enterprise computing
- A **large collection** of commodity components
  - PC-based servers (CPUs, DRAM, disks),
  - Ethernet networking
  - Commodity OS and software stack
  - 10s to 100s of thousands of nodes
- High-bandwidth networking (10Gbps 40Gbps 100Gbps)
- Power delivery, cooling, UPS (50 – 200MW)
- System software for DC management (centralized or distributed)
- Software that implements internet services



Data-centers (DC)

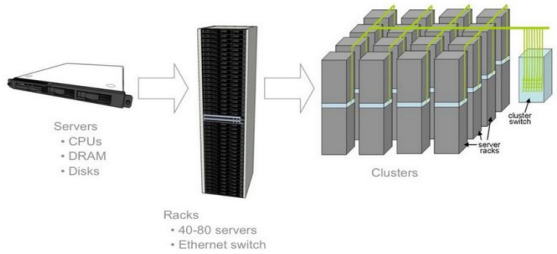
- ~500 Datacenters (w/ >10k servers)
- ~ $30 billion Workloads for AWS 2018
- Microsoft and Google > 2,000,000 servers
- ~3% of US Power consumption

Ouvert en 2009, le data center Azure de Microsoft à Dublin représente pour l'heure une capacité de 39 000 mètres carrés de salles informatiques. L'équivalent de presque huit terrains de football. 80 salariés de Microsoft travaillent sur le site, aux côtés de quelques prestataires triés sur le volet.

Data-centers (DC)

Pour la première fois, Google lève le voile sur l'un de ses secrets les mieux gardés : ses centres de données. Ici, le datacenter de Council Bluffs, dans l'Iowa, qui s'étend sur plus de 10 000 mètres carrés. Il est notamment utilisé pour héberger le moteur recherche de Google ainsi que YouTube.

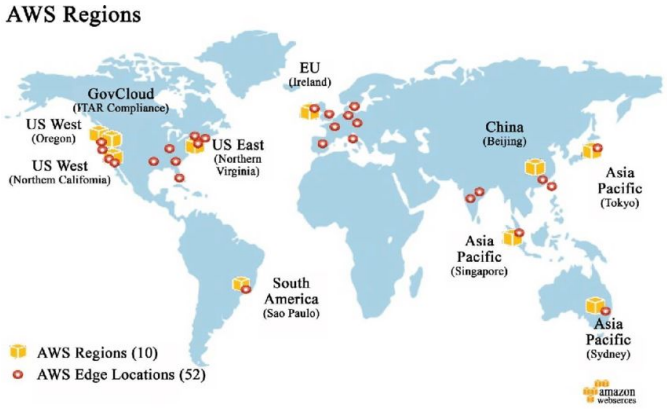
Data-centers (DC)



DataCenters

- **Compute and Memory**
  - A PC-based server (most of them x86)
  - Custom Linux OS (optimized for latency)
  - 1U, 2U, or blade form factors are popular
- **Networking**
  - 10Gbps ethernet links to rack switch
  - A few 10Gbps links to cluster level switches
- **Storage**
  - Distributed file system using disks on servers (GFS)
  - Network attached storage (NAS) devices

Data-centers (DC)



Compute services

Boris TEABE
boris.teabe@inp-toulouse.fr

Goals

1. **Bare Metals**
2. Virtualization
3. Containers
4. Unikernels/MicroVM

Compute Services (IaaS)

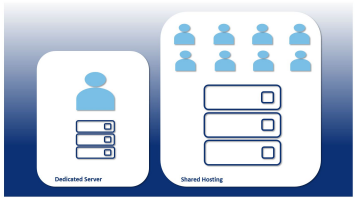
- **An infrastructure is proposed to the User**
  - Number of CPUs
  - Amount of Memory
  - Network Bandwidth
  - Storage capacity

Compute Services (IaaS)

- Several way to provide an infrastructure
  - **BareMetal**
    - Fully dedicated server
  - **Virtualization**
    - Abstraction of a hardware and software isolation
  - **Containers**
    - Lightweight virtualization, less isolated
  - **Unikernel**
    - The future, Strong isolation with good performance

Compute Services (IaaS)

- **Bare Metal (Bare Metal As A Service)**
  - Fully dedicated servers allocated to the clients
  - The client has access fully to a server
  - Can act on the hardware, e.g access hardware features
  - Useful for sensitive workload with lot of computation
  - But, very expensive
  - Provided by AWS, GCP etc.

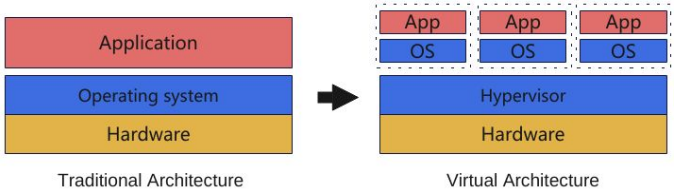


Goals

1. Bare Metals
2. **Virtualization**
3. Containers
4. Unikernels/MicroVM

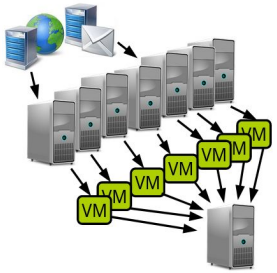
Compute Services (IaaS)

- **Virtualization**
  - Set of techniques, hardware and software, which allow managing simultaneously on a single machine several operating systems (called virtual machines (VMs)). Examples: Xen, VMWare, KVM, HyperV, etc;



Virtualization: motivations

- **Consolidation**
  - Creating X virtual machines from X physical ones and running them on Y physical hosts
    - **With Y < X**
    - Historical motivation for developing virtualization technologies
    - Gives (most of) the benefits of multi-computer systems without the $/management costs:
      - Software dependencies
      - Reliability
      - Security



Virtualization: motivations

- **Use Cases: Software Development**
  - **Flexible OS diversity: different OS on the same machine**
    - e.g VirtualBox with linux for kernel development
  - **Rapid and cost-efficient provisioning**
    - Way faster that Physical machine, we will see during lab works
  - **VMs are self-contained**
    - Practical way to “pack” an application with all its software dependencies
      - Model and version of the OS, libraries, etc.
      - Useful for development, automated testing, and even deployment

Virtualization: motivations

- **Migration and Checkpoint/Restart**
  - VMs are self-contained and can be migrated between hosts
    - **Live migration**: moving VMs from one host to another
    - Freeing resources for maintenance or when a fault is expected
    - Increased performance
    - Distributed resources scheduling: load balancing, consolidation for power savings, etc.
  - Checkpoint/restart for long-running jobs
    - Dump the VM state to disk in order to be able to resume it later
    - Both techniques are straightforward for VM as opposed to processes

Virtualization: motivations

- **Use Cases: Cloud computing**
  - Virtualization central technology in cloud computing
    - Allows the providers to securely share their computing infrastructure between clients (tenants)
    - Offloading local tasks to remote computing resources
      - Rent a VM to put a web server (IaaS)
      - Offload mail server online to Gmail/Outlook (SaaS)
      - To save on management, infrastructure, development maintenance costs

Virtualization: motivations

- **Security**
  - Virtualization provides **very strong isolation** between guests
    - **Sandboxing**
      - Cloud, virus/malware analysis, honeypots, process/task level isolation through virtualization
    - **VM introspection**
      - Analysis of the guest behavior from a privileged level higher than the OS's Guest OS cannot be trusted
        - E.g. Cloudvisor-D

Virtualization: virtual machines

- A complete compute environment with its own isolated processing capabilities, memory and communication channels.

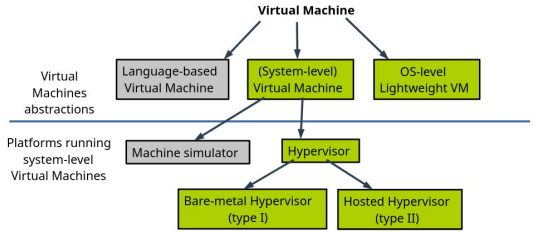
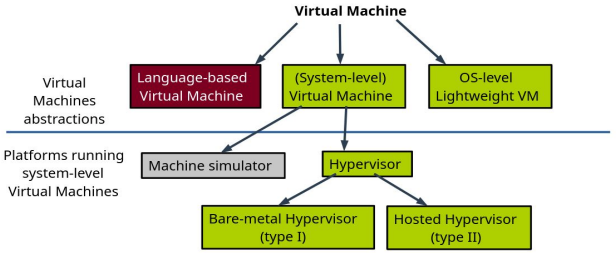


Figure from https://olivierpierre.github.io/virt-101/

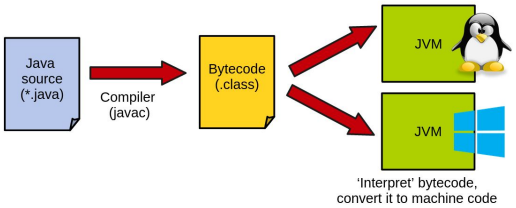
Virtualization: virtual machines



From https://olivierpierre.github.io/virt-101/

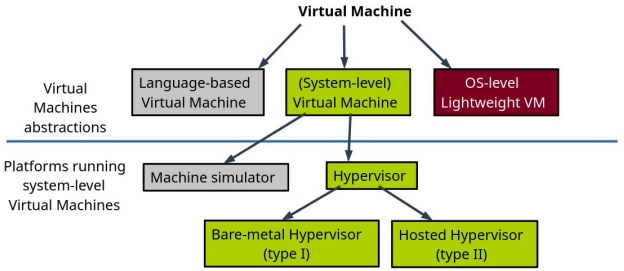
Virtualization: virtual machines

- Java Virtual Machine, JavaScript engines, Microsoft Common Language Runtime
  - Designed to run single applications



From https://olivierpierre.github.io/virt-101/

Virtualization: virtual machines

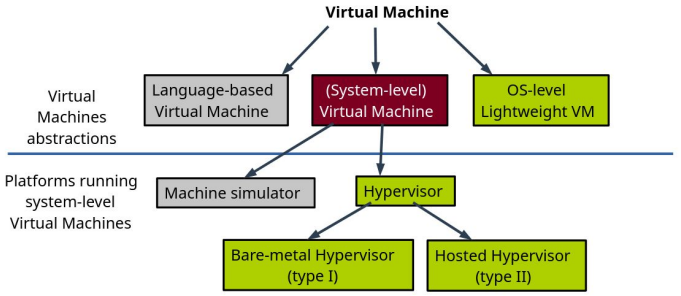


From https://olivierpierre.github.io/virt-101/

Virtualization: virtual machines

- **OS-level Lightweight VMs (discussed in detail later)**
  - Sandboxing: Isolation of native code running directly on the CPU though hardware/software mechanisms. Through OS mechanisms offering more isolation guarantees than regular process-based isolation
  - No attempt is mode to virtualize the hardware: Isolation enforced by the OS/framework level (breaks backward compatibility in some cases)
    - Cannot run unmodified OS
  - E.g. Containers, Google Native Client, LibOSes

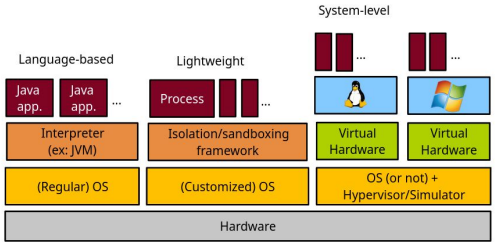
Virtualization: virtual machines



From https://olivierpierre.github.io/virt-101/

Virtualization: virtual machines

- **System-level Virtual Machines**
  - Creates a model of the hardware for a (mostly) unmodified operating system to run on top of it
    - Each VM running on the computer has its own copy of the virtualized hardware

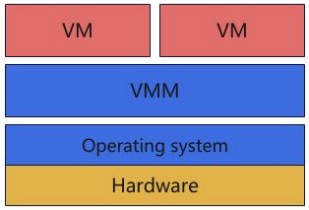


Virtualization

- **System-level Virtual Machines**
  - Full virtualization
  - Para-virtualization
  - Hardware assisted virtualization

Virtualization

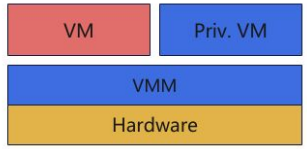
- **Full virtualization**:
  - An OS executes applications at user level. The VMM is one such application. Every instruction from VM is emulated by the VMM. The OS executed on a VM is not modified and can be of any type (Linux, Windows, etc.). Ex: virtualBox



- Concretely:
  - It's hardware simulation (slow)
  - Supports several OS types

Virtualization

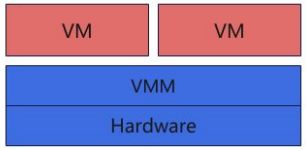
- **Para-virtualization (PV):**
  - The VMM replaces the host OS and behaves as a proxy for accessing the hardware. The host OS is considered as a VM (privileged) and it is used by the VMM to perform particular tasks. Constraint: VMs' OSes have to be modified (to invoke the VMM for privileged operations). Ex: Xen, VMware, etc.



- Concretely:
  - A mix between simulation and native. Several OS types
  - Each VM is allocated hardware resources and the VMM controls access to hardware
  - Hypercalls (a modified OS invokes the VMM)

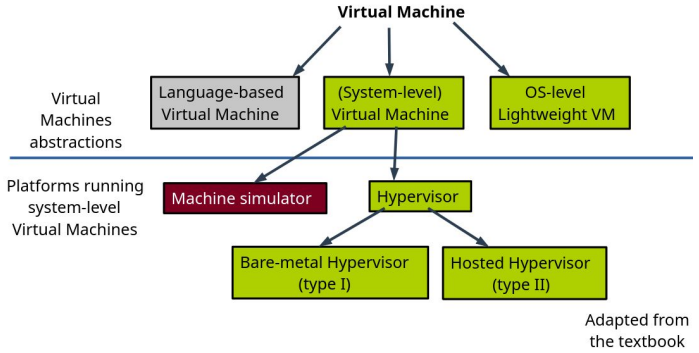
Virtualization

- **Hardware assisted virtualization (HVM):**
  - The hardware is aware of virtualization. VMs' OSes don't have to be modified. Ex: Xen, KVM, VMWare etc.



- Concretely:
  - Native, but without OS modification
  - The hardware is able to manage several VMs (MMU, network device …)
  - Sometimes, para-virtualization is faster or more flexible.

Virtualization

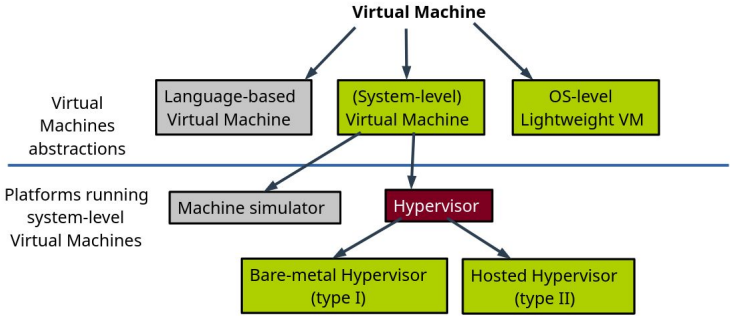


Adapted from the textbook

Virtualization: Machine simulation

- **Full virtualization**
- Cross ISA emulation: for usage as substitute
  - Compatible with legacy applications
    - e.g: Qemu
- Architecture emulation
  - Help simulate a hardware architecture for study purpose
    - e.g: Android systems
- Each guest instruction is interpreted in software

Virtualization



Adapted from the textbook

Virtualization: hypervisor

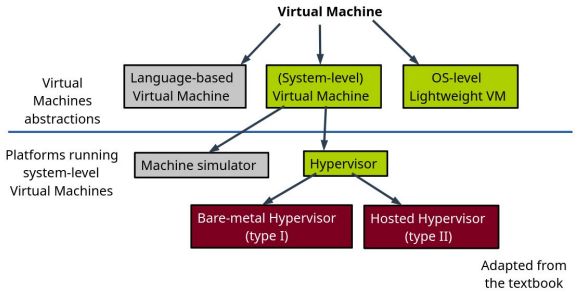
- **Hypervisor or Virtual machine monitor (VMM)**
  - Relies on direct execution for performance reasons (close to native)
  - VM code executes directly on the physical CPU, at a lower privilege level than the hypervisor
  - Privileged instructions trap to the hypervisor
    - Switch to the hypervisor which determines what to do with that instruction: trap-and-emulate model
    - Then back to the VM execution
  - E.g Xen, Linux KVM, VMware ESXi, MS Hyper-V etc.

Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Multiplexes physical resources between VMs-Run VMs while minimizing virtualization overheads
  - Tries to get as close as possible to native performance
  - Ensure isolation between VMs and between VMs and the hypervisor
    - Isolates physical resources: for example memory/address spaces
    - Isolate performance

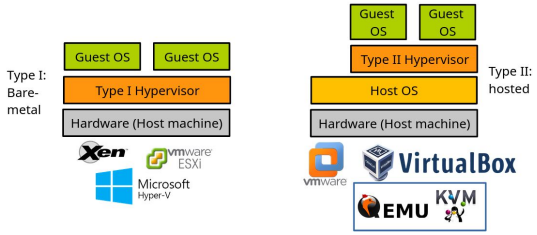
Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**



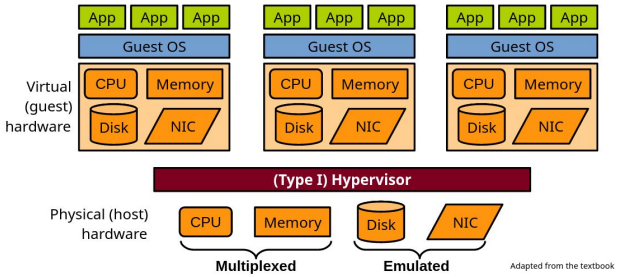
Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Resources allocation and scheduling
    - Type I: done by the hypervisor
    - Type II: more involvement from the Host OS



Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Simplified Hypervisor Sketch

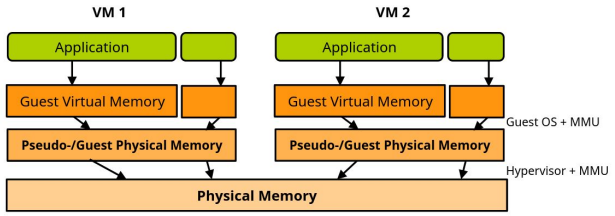


Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Multiplexing CPU and Memory
    - Virtual CPU runs with reduced privileges, cannot execute privileged instructions (that could allow to escape isolation)
    - Traps to the hypervisor on such instructions (virtualization overhead)
    - Run the hypervisor in kernel mode and the VM in user mode.

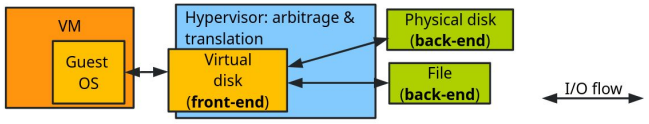
Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Multiplexing CPU and Memory
    - Another level of translation added, taken care of by the hypervisor: (Guest) virtual memory -> (Guest) pseudo-physical memory -> (host) physical memory.



Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Emulating I/Os
    - I/O mostly emulated for compatibility reasons
    - I/O devices have well defined interfaces, for example: send a set of network packets, read 128K from disk from secto X, etc.
    - The hypervisor emulate simple virtual device (disk/NIC) that can be accessed with commonly implemented drivers (ex IDE/SCSI): **front-ent**
    - Hypervisor redirects I/O to actual devices or other abstraction, ex: disk or file: **back-end**.



Compute Services

Boris TEABE
boris.teabe@inp-toulouse.fr

Goals

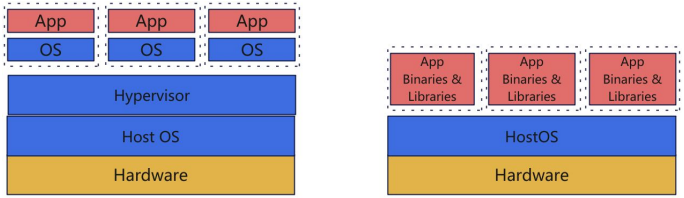
1. Bare Metals
2. Virtualization
3. **Containers**
4. Unikernels/MicroVM

Containers

- Lightweight Virtualization
  - Represent virtualization solutions aiming at providing, vs. traditional virtual machines:
    - Lower memory footprint: 0 to a few MB per virtualized instance
    - Lower disk footprint: in the order of KBs/MBs
    - Faster boot times: in the order of micro/milliseconds
- **Simple examples are containers and unikernels**

Containers

- **Containers: process-level sandboxing technologies**
  - Enforced by the operating system
  - **Called OS-level virtualization**



Containers

- The OS restricts the visibility on system resources for a process or a set of processes
- The OS also controls hardware resource allocation/usage between such isolated processes
  - CPU cores, memory, disk/network bandwidth, etc.
- A container is much lighter than a VM
  - Per-container system memory/disk footprint close to 0
  - Boot time is that of spawning a process, i.e micro seconds
- Still containers are not an ideal form of virtualization
  - **Security issues**

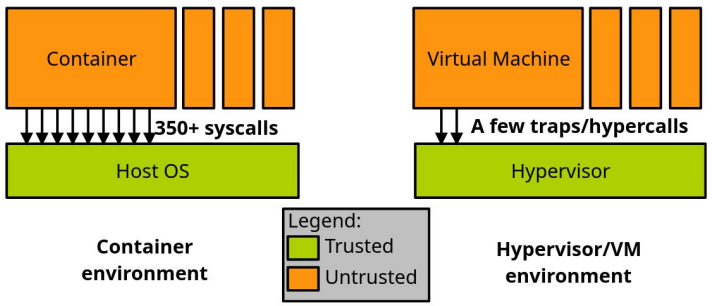
Containers

- **Uses cases**
- **Software development/testing/deployment**
  - Develop, build and test in a controlled, identical environment
  - Deploy in the same environment as the development one (repeatability)
    - Can be deployed on any server/cloud supporting containers independently of the host configuration
- **Lightweight (low cost) & elastic virtualization**
  - Containers consume few resources and can be brought up/destroyed very fast
  - Cloud services such as Gmail and Facebook make extensive use of containers
  - Function as a Service (e.g. AWS Lambda)

Containers

- **Software Development with Containers**
- **Package application programs and dependencies**
  - One of the main benefits: ease of development/testing/deployment
    - "Shipping containers"
- **Developing and running application X requires a complex set of dependencies**
  - Libraries sources and/or binaries (ex: glibc, etc.)
  - Build tools (ex: cmake, autotools, etc.)
  - System tools (ex: perl, grep, etc.)
  - All of these with sometimes very specific versions that may not be compatible with that of application Y that we also want to build & run
- **One solution would be to build and run application X and Y each in their own VM**
  - Too heavy, does not scale to a high number of apps
  - **Containers can help!**

Containers and security

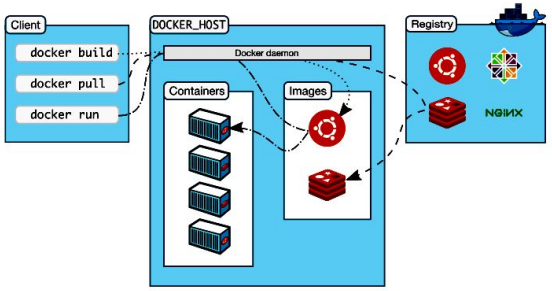


Containers and security

- The isolation enforced by a host OS between containers is not trusted to be as strong as that enforced between VMs by a hypervisor
  - Due to the size and complexity of the interface between a container and the host OS: **the system call interface**
- Attempts at securing containers:
  - Run containers within virtual machines...

Docker

- Virtualization system Allow building very light VMs (containers)
- Set of user-friendly tools for managing containers
- Client-server architecture



Docker

- **The image of a VM**
  - Docker relies on Union File System for the representation of images
  - An image is represented as a set of layers
  - Each layer describes a modification of the file system (like diff)
- **Advantages of this representation**
  - Allows building a file system
  - From a standard image
  - With small additional data (tens of Mb instead of hundreds of Mb)
  - Efficiently
- **The same set of standard images can be reused**
- **The modification of a file system does not generate a full file system (only a layer)**
  - Only diffs are saved
- **Docker allows sharing images**
  - https://hub.docker.com

Docker

- **Some basic commands**
- Installation under Linux
  - *wget -qO- https://get.docker.com/ | sh*
- Starting a container
  - *docker run -it ubuntu bash*
- Lookup the image
  - If the image is not in the local registry, download from the hub
  - Ubuntu: pre-existing image in the hub
- Build the Linux file system
- Start the container
- Configure the IP address of the container
  - Also communication between outside and the container

Docker

- List local images
- docker images

```
hagimont@hagimont-pc:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              cd6d8154f1e1        12 days ago
84.1MB
hagimont@hagimont-pc:~$
```

- Log in the hub
  - docker login/logout
- Lookup an image in the hub
  - docker search hagimont

```
hagimont@hagimont-pc:~$ docker search hagimont
NAME                        DESCRIPTION                     STARS               OFFICIAL
        AUTOMATED
hagimont/docker-whale                                       0

hagimont/hagi               my repo                         0

lwapet/projet_docker        ENSP - hagimont Daniel          0

hagimont@hagimont-pc:~$
```

Docker

- Creation of an image
- From a container instance
  - Start the container (from an initial standard image)
  - Modify the file system (apt-get install ...)
  - Commit the instance with a new image name
    - docker commit c8744fe9eab6 ubuntu:hagi

```
hagimont@hagimont-pc:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
c8744fe9eab6        ubuntu              "bash"              4 seconds ago       Up 2 seconds
hagimont@hagimont-pc:~$ docker commit c8744fe9eab6 ubuntu:hagi
sha256:58bf3876c787780770f7c75740c675bf1c3ab4f34d128f48f8c5163e6a0df422
hagimont@hagimont-pc:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              hagi                58bf3876c787        6 seconds ago       84.1MB
ubuntu              latest              cd6d8154f1e1        12 days ago         84.1MB
hagimont@hagimont-pc:~$
```

Docker

- **Creation of an image**
- **From a Dockerfile**
  - mkdir foo
  - cd foo
  - Create a file Dockerfile
    - # This is a comment
    - FROM ubuntu
    - RUN apt-get update && apt-get install -y apache2
- *docker build -t hagimont/ubapache:v2 .*

```
hagimont@hagimont-pc:~/foo$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hagimont/ubapache   v2                  904d4cc37cdd        About a minute ago   222MB
ubuntu              latest              cd6d8154f1e1        12 days ago         84.1MB
hagimont@hagimont-pc:~/foo$
```

Docker

- **Management of images in the hub**
  - You must be logged in
  - Save the image in the hub
  - *docker push hagimont/ubapache:v2*
- **Download an image from the hub**
  - *docker pull hagimont/ubapache:v2*
- **Tag an image (versioning)**
  - *docker tag id_image training/sinatra:thetag*

Docker

- **Goal of data volumes**
  - make visible in one or more containers a directory or file from the host file system
  - Allows file sharing between several containers
- **Persistent even after container destruction**
- Any modification is immediately effective
- **Command**:
  - *docker run -it -v /tmp/host_file:/tmp/container_f*
- Port redirection
- Example of link: host → container
  - *docker run -d -p 80:5000 hagimont/apache*
- Any connection on port 80 of the host is forwarded to port 5000 of the container

Unikernels

Boris TEABE
boris.teabe@inp-toulouse.fr

Goals

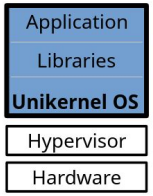
1. Bare Metals
2. Virtualization
3. Containers
4. **Unikernels/MicroVM**

Unikernel

- **Unikernel: application + dependencies + thin OS compiled as a static binary running on top of a hypervisor**
- **Single-***
  - Single purpose: run 1 application
    - Want to run multiple applications? run multiple unikernels
  - Single process
    - Want to run a multi-process application? run multiple unikernels
    - However, SMP (multicores) and multithreading are supported
  - Single binary and single address space for application + kernel
    - No user/kernel protection needed

Unikernel

- **A form of lightweight virtualization**
  - Contain and run only what is absolutely necessary to the application
  - Cost advantage: memory/disk footprint reduction
  - Considered as a secure alternative to containers
    - Unikernels are virtual machines!
- Per-application tailored kernel (LibOS/Exokernel model)
  - Specialization for lightweightness but also performance
- Reduced OS noise, increased performance
  - Low system call latency: app + kernel in ring 0, system calls are function calls
  - Sub-second boot time



Unikernel

- Cloud applications: servers, micro-services, SaaS
- Embedded virtualization, Edge computing, IoT
- Network Function Virtualization, HPC, VM introspection, malware analysis, secure Desktop applications
- etc.
- Contrary to containers which are a mature and widespread technology, unikernels are still at the stage of research prototypes

Unikernel

- **Unikernels are lightweight AND secure, why didn't I heard of them?**
- Because it's hard to port existing applications!
  a. Proprietary software → source code not available
  b. Incompatible language
  c. Unsupported features
  d. Porting is hard, needs knowledge about both application and unikernel
  e. Complex built toolchains

Container Orchestrators

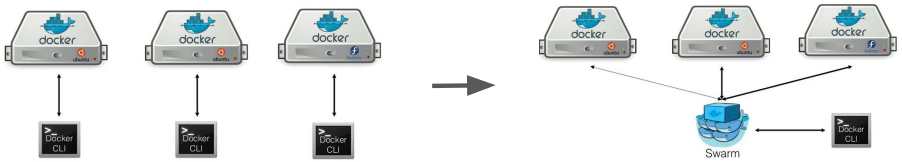
Boris TEABE
boris.teabe@inp-toulouse.fr

Goals

1. **Docker Swarm**
2. Kubernetes
3. Serverless computing

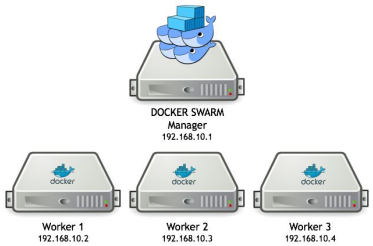
Docker Swarm

- Native solution of Docker for clustering
  - Turn a cluster into a unique virtual host
  - Use the same API as Docker
- Allow to manage and Schedule containers on a cluster



Docker Swarm

- A Docker Swarm is a group of either physical or virtual machines that are running the Docker application and that have been configured to join together in a cluster. machines that have joined the cluster are referred to as nodes or worker.
- Extremely easy to get started



Docker Swarm

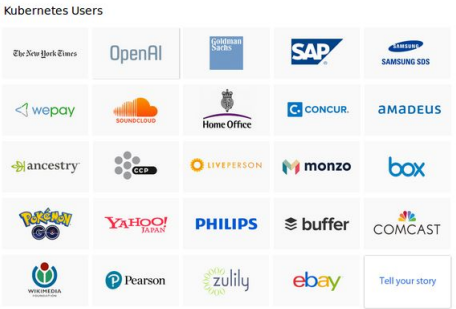
- **Installation**
  - Create a cluster
    - *docker swarm init*
  - Join a cluster
    - *docker swarm join --token ……*
- **Deployment**
  - *docker stack deploy -c ….*

Goals

1. Docker Swarm
2. **Kubernetes**
3. Serverless computing

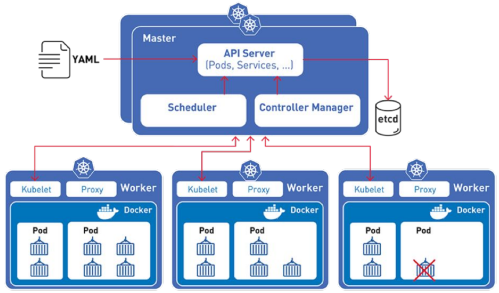
Kubernetes

- A container orchestration system
- Kubernetes abstracts the thousands of nodes in a cluster and provides industry methods to manage applications. Administrator describes and declares the “desired state”, and kubernetes converts the “current state” to “desired state”.



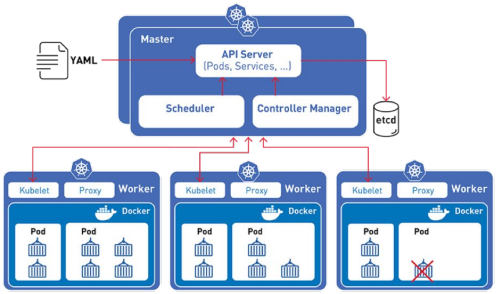
Kubernetes

- **Architecture**
  - Master
  - Nodes
  - Pod
  - Service and Labels
  - Container
  - Node
    - Kubelet
    - KuberntesProxy



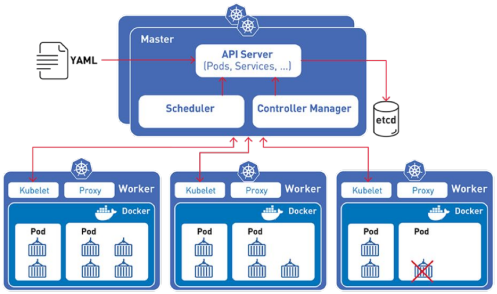
Kubernetes

- **Master**
  - Master maintains the state of the kubernetes server runtime
  - State is maintained in the etcd backend
  - It is the point of entry for all the client calls to configure and manage kubernetes componets like Nodes, Pods, ReplicationControllers, Services



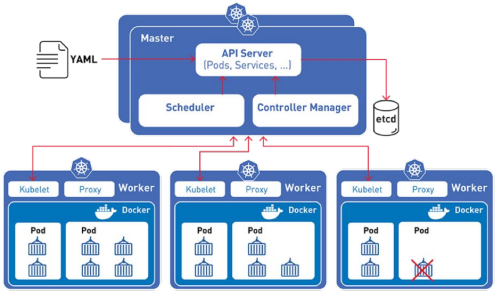
Kubernetes

- **Node/Worker**
  - Represents the resource provided for provisioning pods
  - Node runs a docker etcd and a kubelet daemon



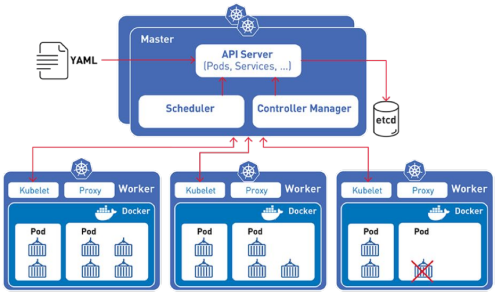
Kubernetes

- **Kubelet**
  - Component which runs on each nodes and manages the pod and container lifecycle



Kubernetes

- **Proxy**
  - Manages the network rules on the node and performs connection forwarding or load balancing for kubernetes cluster services

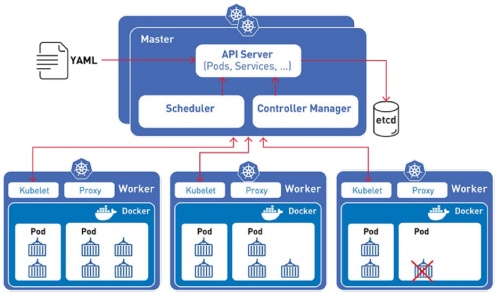


Kubernetes

- **Concepts-core**
  - **Cluster**. A collection of hosts that aggregate their available resources including CPU, ram, disk and their devices into a usable pool
  - **Master**. Represent a collection of components that make up the control plane of Kubernetes. These components are responsible for all cluster decisions including both scheduling and responding to cluster events
  - **Node**. A single host physical or virtual capable of running pods.
  - **Namespace**. A logical cluster or environment. A method of dividing a cluster.

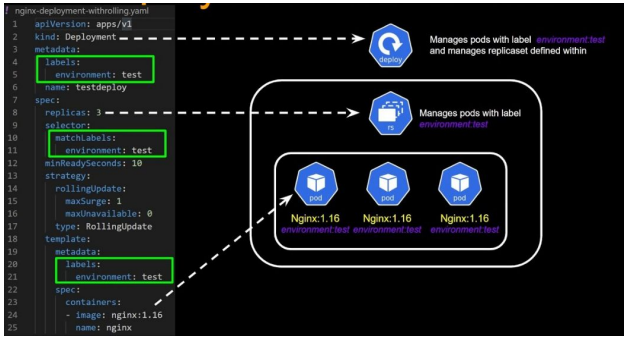
Kubernetes

- **Concepts-workloads**
  - **Pods**
    - Collection of containers that can run on a host, represent an application
    - In Kubernetes a pod represents a bundle of containers with shared volumes



Kubernetes

- **Concepts-workloads**
  - **Deployment.** A declarative method of managing stateless Pods.



Kubernetes

- **Concepts-network**
  - **Service**
    - An abstraction which defines a logical set of pods and a policy by which to access them
    - The set of Pods targeted by a service is (usually) determined by a label Selector
    - A service defines a TCP or UDP port reservation
    - Allows for abstracted configuration and for mobility and load balancing of the providing containers

Kubernetes

- **Concepts-Storage**
  - **Volumes**
    - Container's disks a ephemeral in nature
    - Everytime container restarts ephemeral disks are restarted
    - They are just a mount point or host dir
    - Several kind of volumes exist:
      - empty dir, AWS EBS, GCE Persitent etc.

Goals

1. Docker Swarm
2. Kubernetes
3. **Serverless computing**

Serverless computing

- Real cloud-native applications: only provide code for the busines core features
- All management and execution provided by the cloud platform
  - From execution environment to service availability
- Serverless

Backend-as-a-Service + Function-as-a-Service

Backend-as-a-Service

- Common backend components in application architectures
  - Database servers, message queues, (object) storage…
- Better served by the cloud provider
  - Mutualized, no overhead for the user, available
  - Provides an ecosystem of components
- Elasticity requirement: scale quickly, up and down, with the FaaS workload

Function-as-a-Service

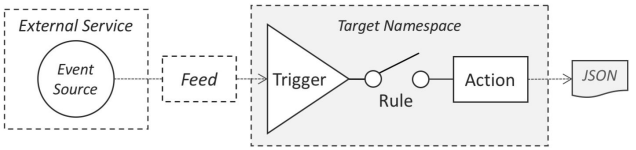
- Run backend code without long-lived servers
  - Execution environments are spawned on-demand
  - All managed by the cloud platform
- The unit of execution is a code block: the function
  - Applications are mostly event-driven
- Central feature of serverless

Benefits of FaaS

- Elasticity: granularity of the request handler
- Deployment: just write code and upload
  - Quick experimentation, update
- Cost: pay only the compute time you need
  - No request=no function running = no resource to pay

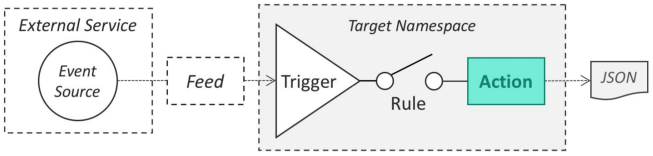
OpenWhisk

- Distributed Serverless platform that executes functions in response to events at any scale
- Is Event-driven
  - Events drive the Serverless execution of functional code called Actions. Events can come from any Event Source or Feed service



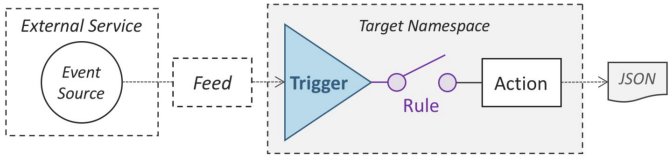
OpenWhisk

- Action:
  - Stateless functions that run on the OpenWhisk platform. Actions encapsulate application logic to be executed in response to events. Actions can be invoked manually by the OpenWhisk REST API, OpenWhisk CLI, simple and user-created APIs or automated via Triggers



OpenWhisk

- **Trigger**: kinds of events sent from Event Sources
- **Rule**: Rules are used to associate one trigger with one action.
- **Event Sources**: These are services that generate events that often indicate changes in data or carry data themselves.



OpenWhisk

- **Architecture of OpenWhisk**