Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

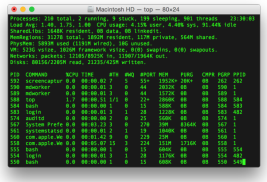
RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, April 14th 2023

Lab Environment

- Homogeneous base environment
  - Ubuntu 20.04 LTS
  - DevOps tools (already installed)
- Used as a playground to build other environments
- Access via OpenSSH from your laptop

+



+

SSH

Microsoft Azure

Standard Ubuntu 20.04 LTS

git

Terraform

ANSIBLE

VirtualBox

docker

openstack

VM-1

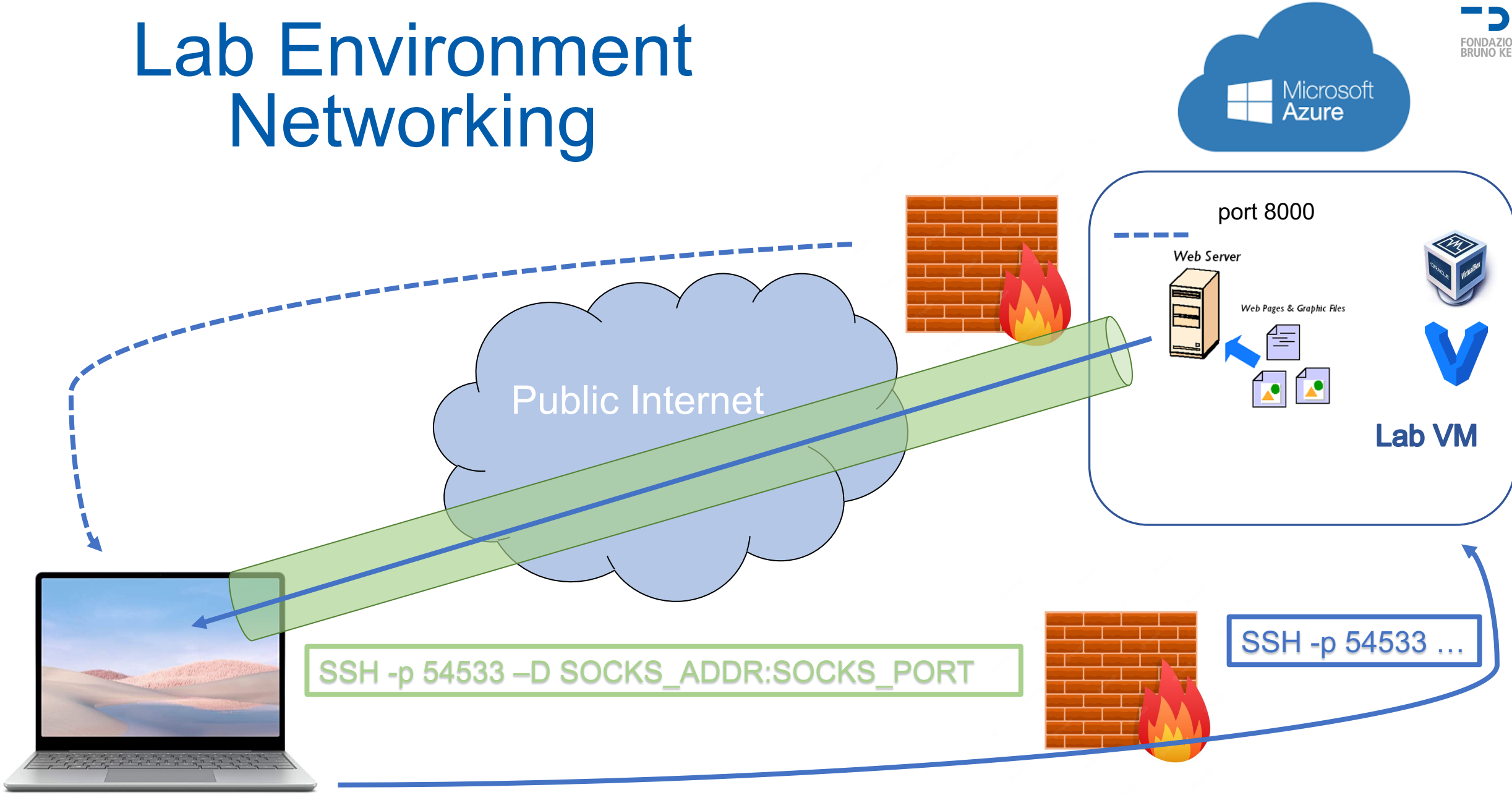
VM-2

VM-3

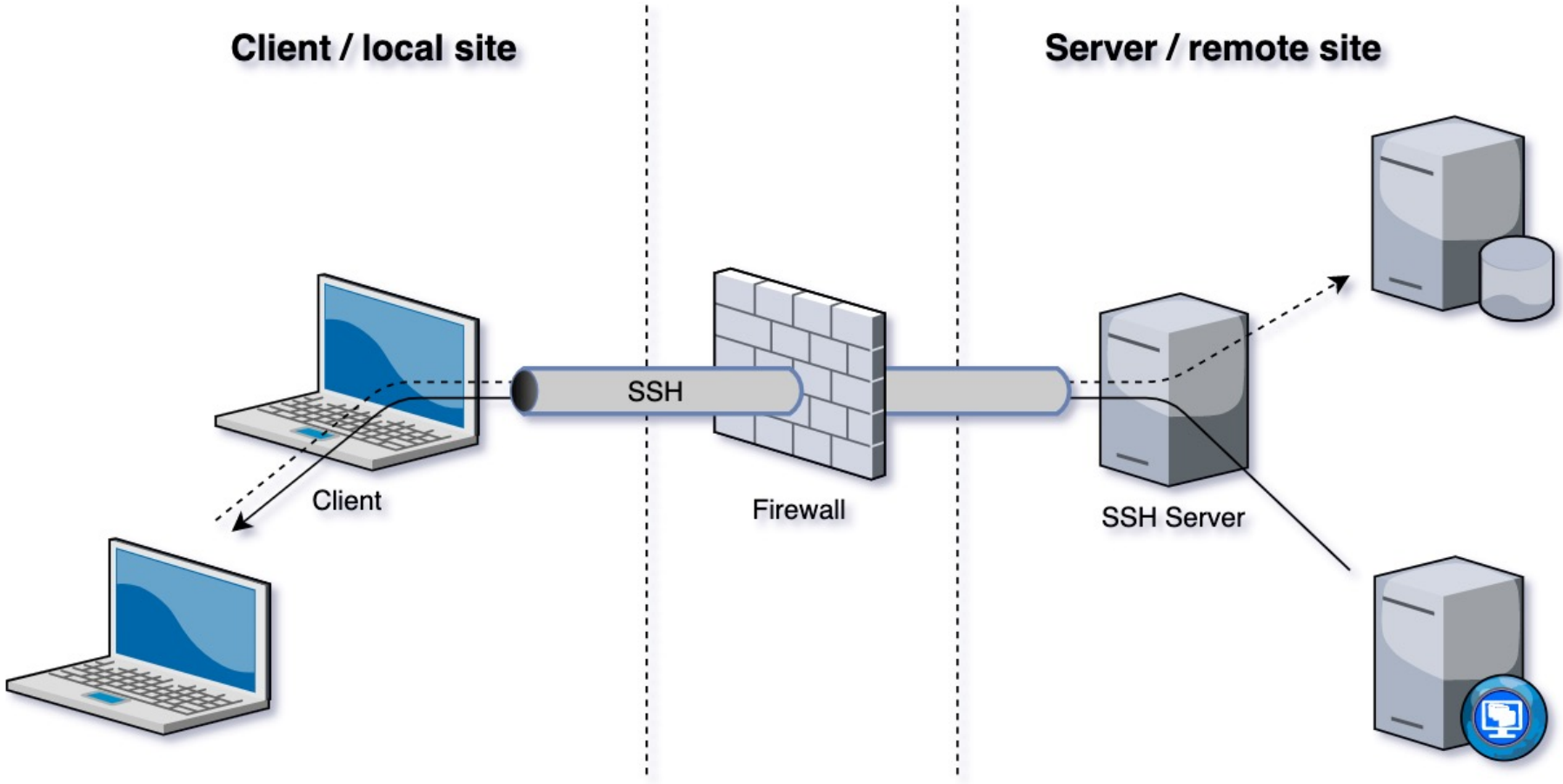
VM-…

SSH

Lab Environment Networking

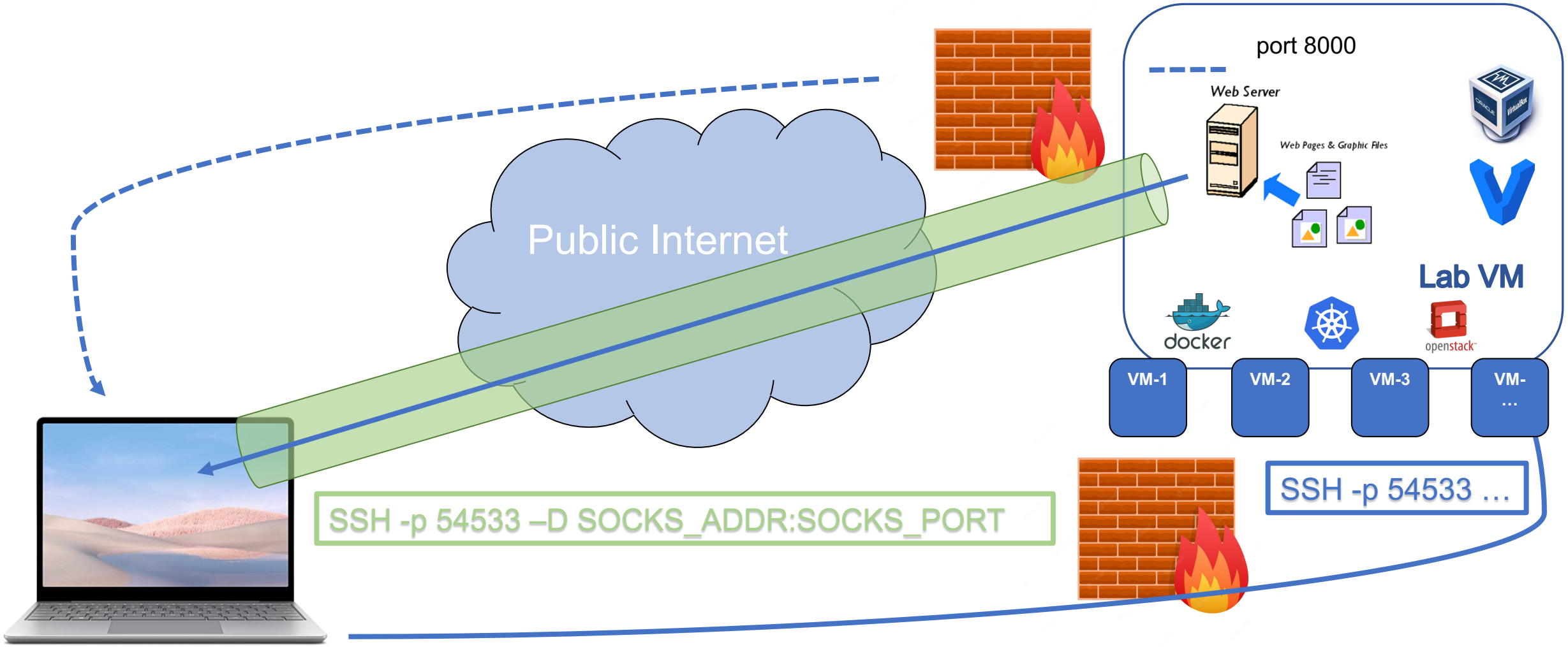


SSH Socks Proxy Tunnel



https://en.wikipedia.org/wiki/SOCKS, How-To

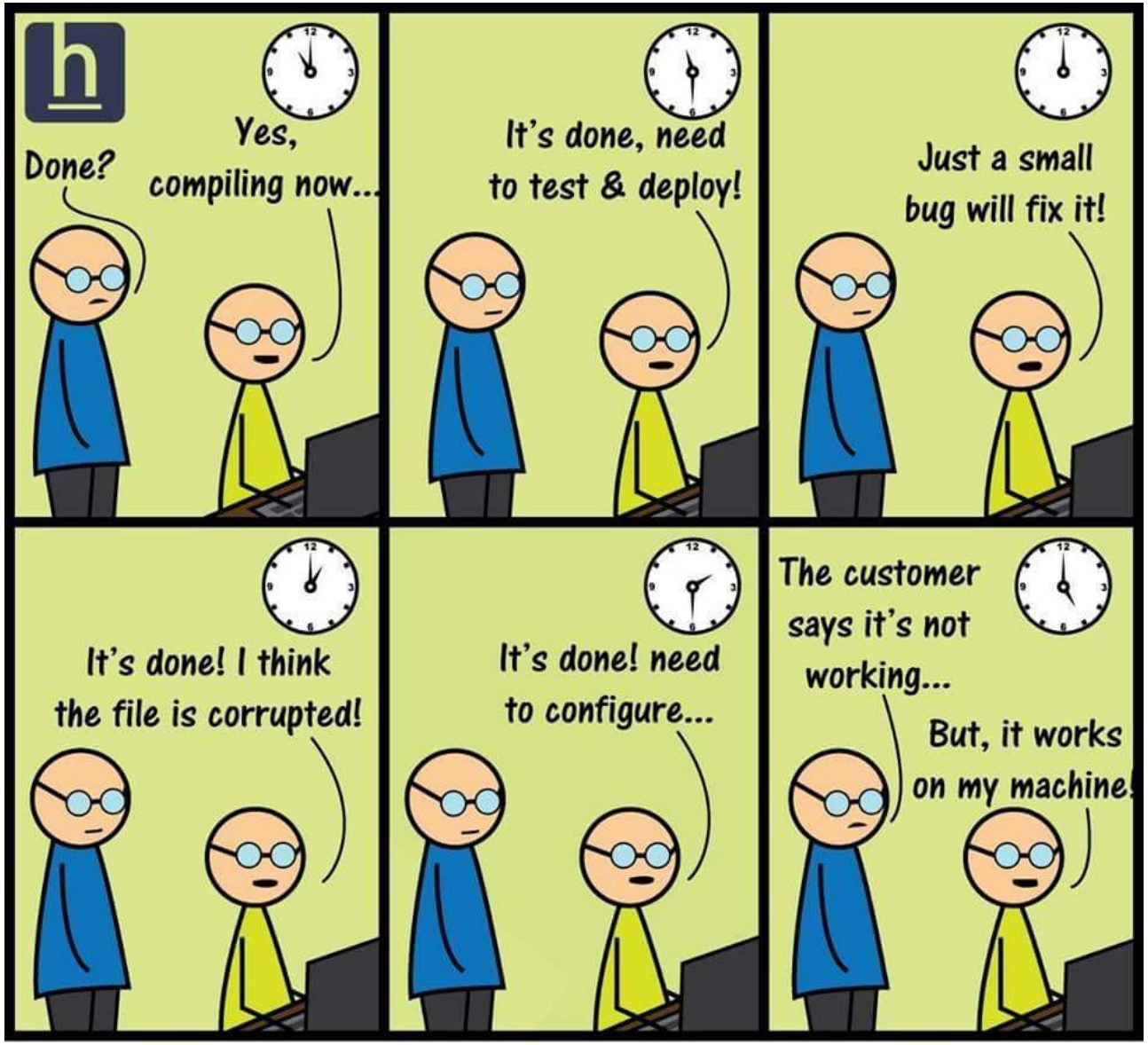
Lab Environment Networking



Application portability & Repeatable Environments

Today's target is to setup an environment which is equal for every student even if we are using different OS and hardware.

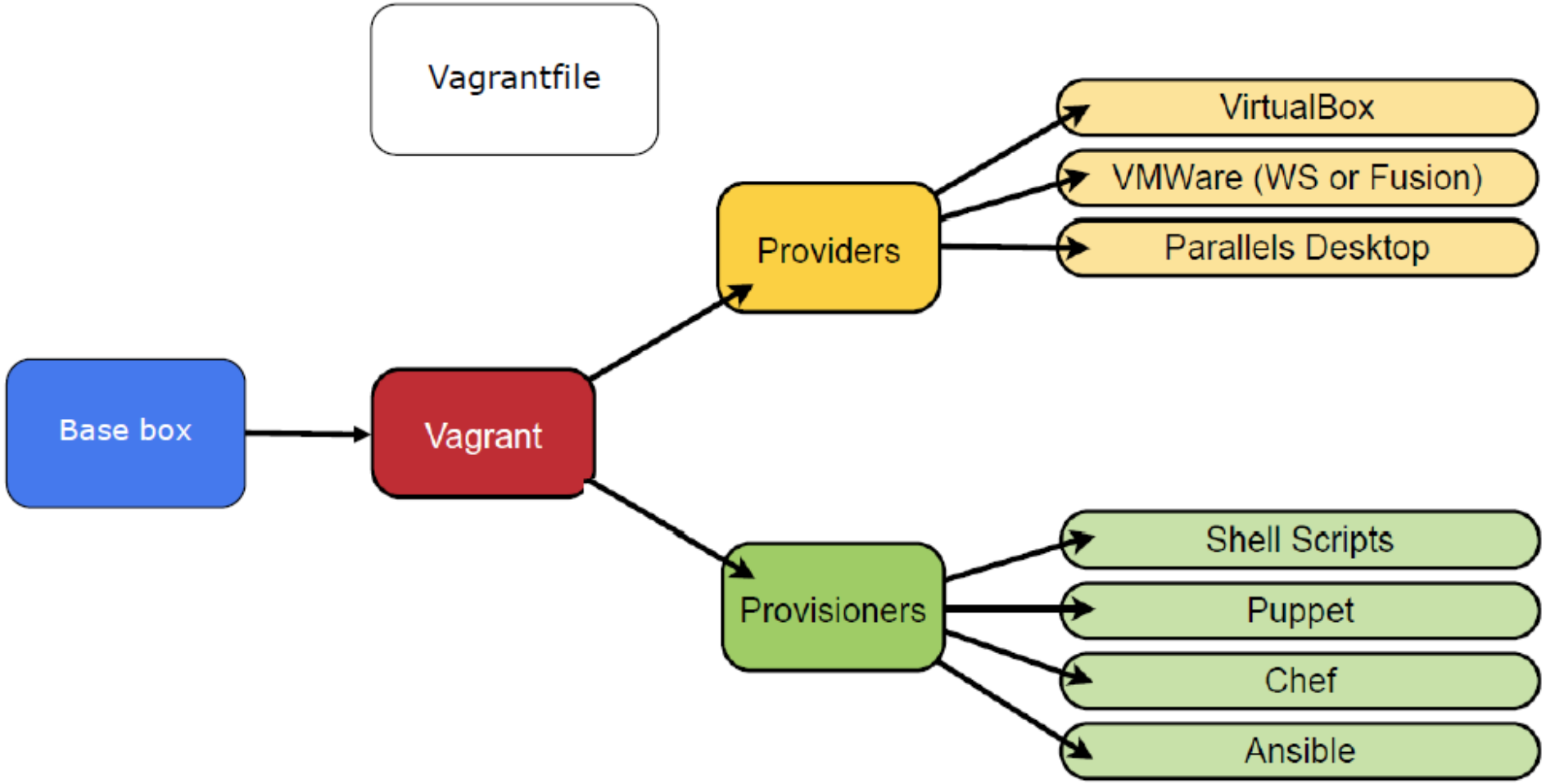
You will learn tools and techniques in order to create such environments.



Be portable and repeatable – Why?

- Working in team
- Different OS, different libraries, different clouds
- Isolate dev environments
  - Security
  - Conflicts
- Speed-up dev environment setup-time
- Easy workflows and automation
  - Do not do it by hand
  - Destroy and recreate
- Increase productivity
- Makes the “*works on my machine*” excuse a relic of the past

Vagrant HowTo 1/2



Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

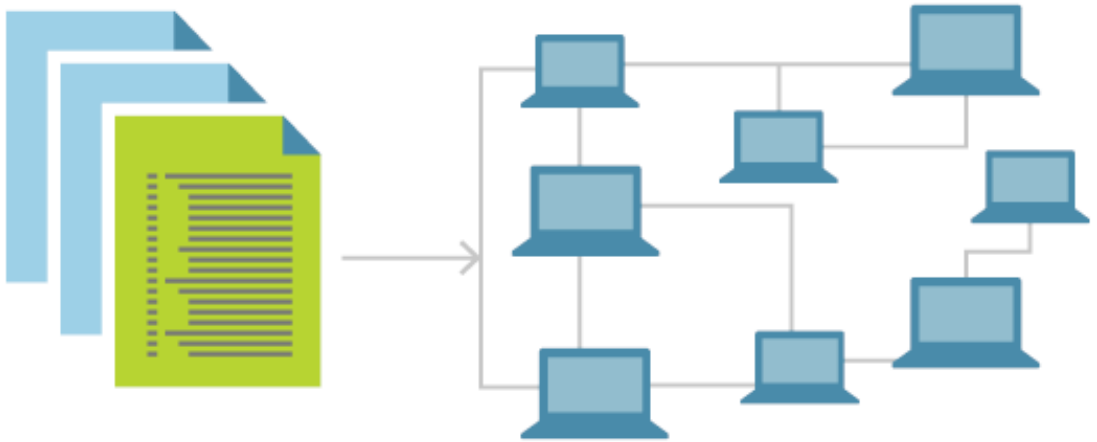
Trento, May 2nd 2023

DevOps Agile Methodology

Development & **Op**eration**s**

The efficient development, deployment and operation of high quality modern software systems

Configuration Management Systems



- In DevOps, permits to maintain OS configuration files
  - **Configuration management (CM)** is a systems engineering process for establishing and maintaining consistency of a product
- Used with IaC
  - **Infrastructure as code** (**IaC**) is the process of managing and/or provisioning computer data centers through machine-readable definition files
- Track changes in VCS, like git

SALTSTACK

CHEF

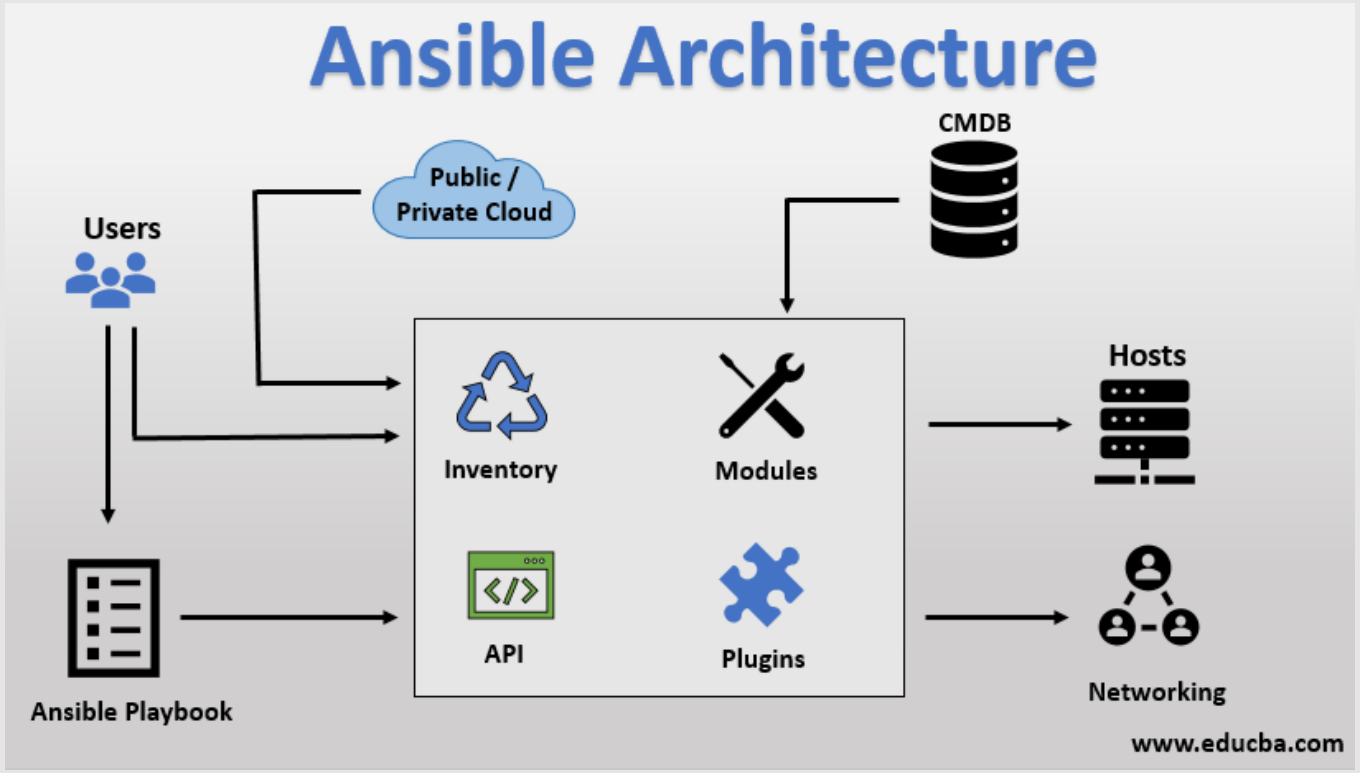
ANSIBLE

CFEngine

puppet

Ansible

(https://www.ansible.com)



- Proprietary / GNU GPL
- Huge docs and community
- Idempotent
- Extensible
- Mostly used for CM
- Can do also IaC
- YAML syntax
- Written in Python
- It is agent-less
- Based on SSH

Quick look at the Ansible Playbook and Inventory

FONDAZIONE BRUNO KESSLER

Ansible playbook [ref]

```yaml
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

Ansible inventory [ref]

```yaml
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
    east:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    west:
      hosts:
        bar.example.com:
        three.example.com:
    prod:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    test:
      hosts:
        bar.example.com:
        three.example.com:
```

```
ansible-playbook playbook.yml \
-f 10
```

```
ansible-playbook -i inventory \
playbook.yml \
-f 10
```

```
PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [default]

TASK [Upgrade the OS (apt-get dist-upgrade)] ***********************************
ok: [default] => {"changed": false, "msg": "Reading package lists...\nBuilding dependency tree..
upgrade...\n0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.\n", "stderr": "", "st
ists...\nBuilding dependency tree...\nReading state information...\nCalculating upgrade...\n0 up
not upgraded.\n", "stdout_lines": ["Reading package lists...", "Building dependency tree...", "R
grade...", "0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded."]}

TASK [Run the equivalent of "apt-get update" as a separate step] ***************
changed: [default] => {"cache_update_time": 1649374426, "cache_updated": true, "changed": true}

TASK [Install required packages] ***********************************************
changed: [default] => {"cache_update_time": 1649374426, "cache_updated": false, "changed": true,
"Reading package lists...\nBuilding dependency tree...\nReading state information...\nThe follow
\n0 upgraded, 1 newly installed, 0 to remove and 26 not upgraded.\nNeed to get 133 kB of archive
l disk space will be used.\nGet:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 figlet a
866 kB/s)\nSelecting previously unselected package figlet.\r\n(Reading database ... \r(Reading d
```

Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, May 5th 2023

OpenStack (https://openstack.org/)

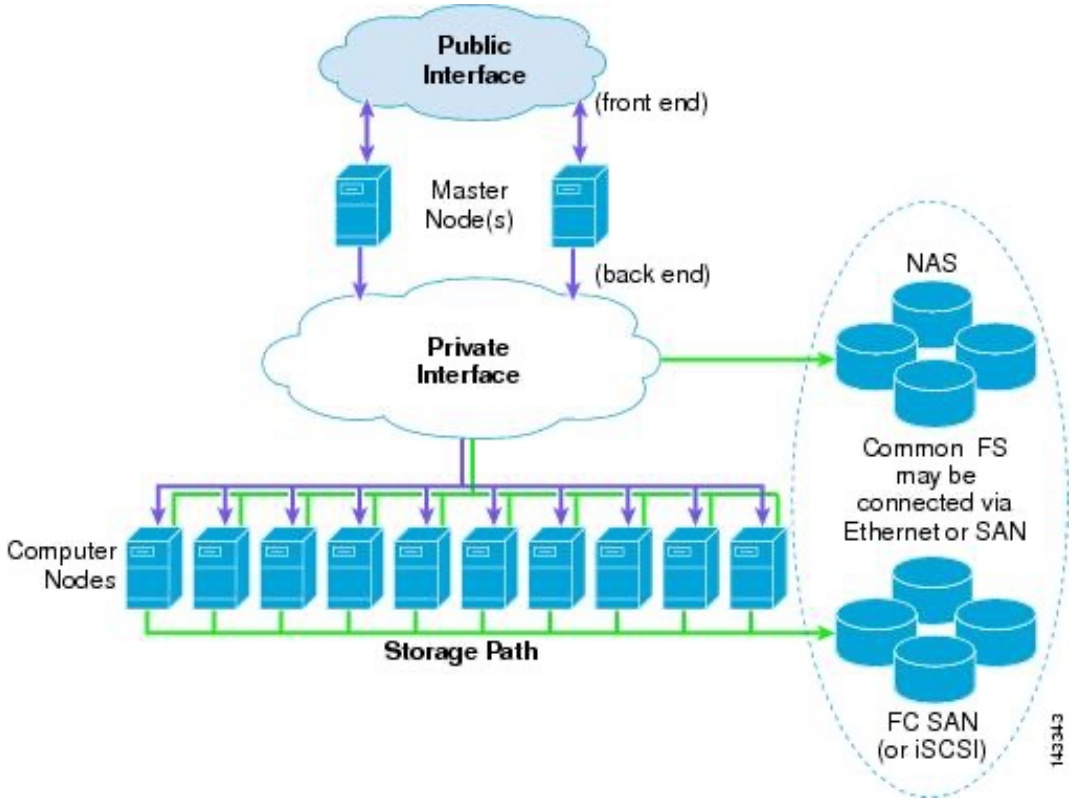
- Open source software for creating private and public clouds.
- OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure.

- OpenStack Community
  - Wiki, Specs, Projects, RC meetings, gerrit, OpenStack Foundation

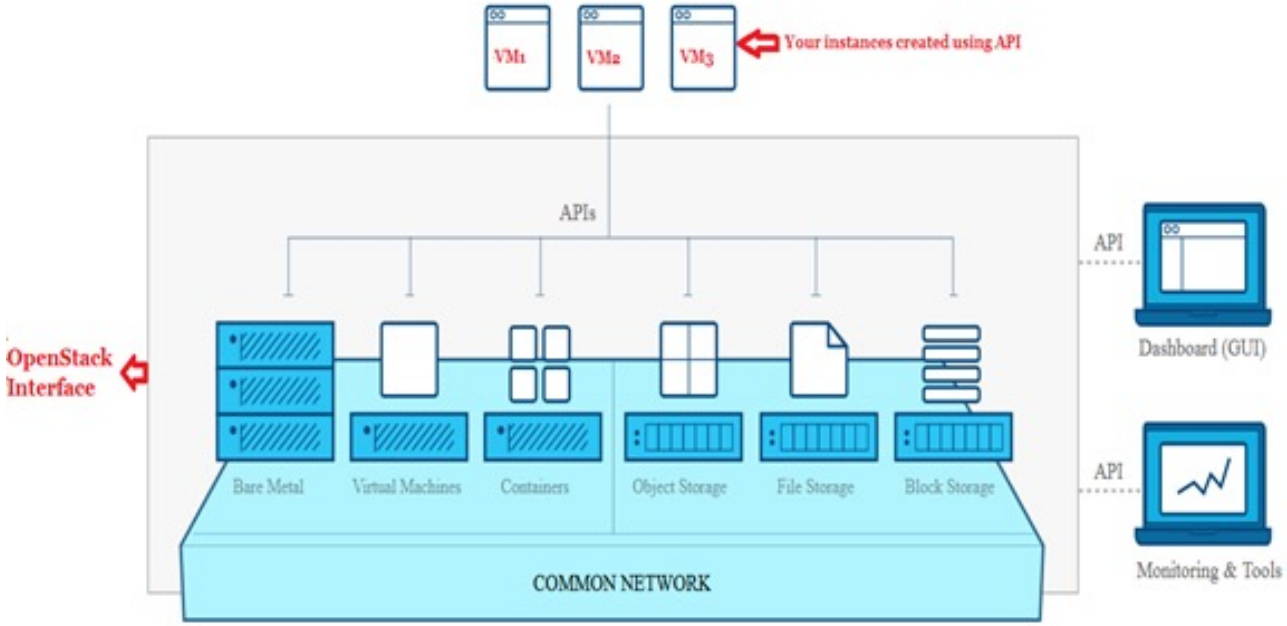
- Four “open”s
  - Open Source, Open Design, Open Development, Open Community
  - More information at the governance page

- https://docs.openstack.org
- http://governance.openstack.org/reference/opens.html

OpenStack Model, Architecture

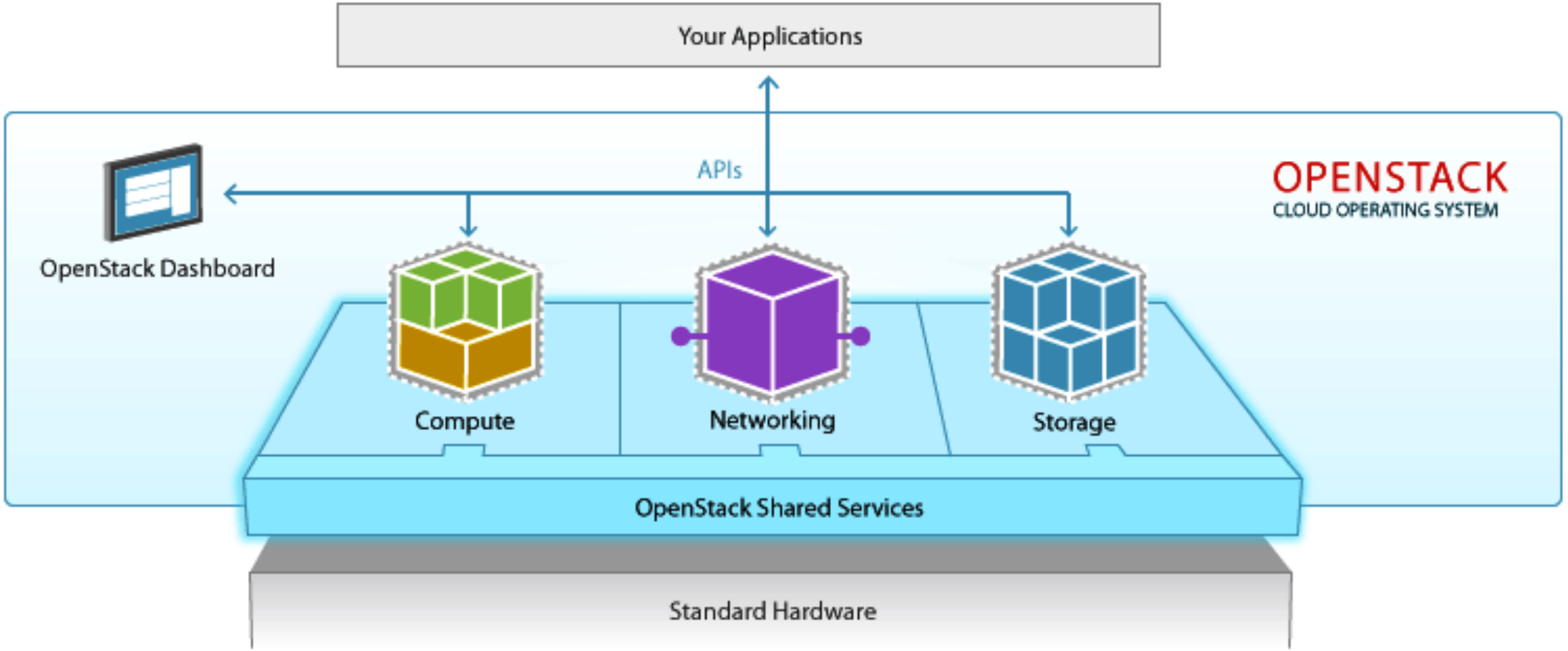


Physical view



Logical view

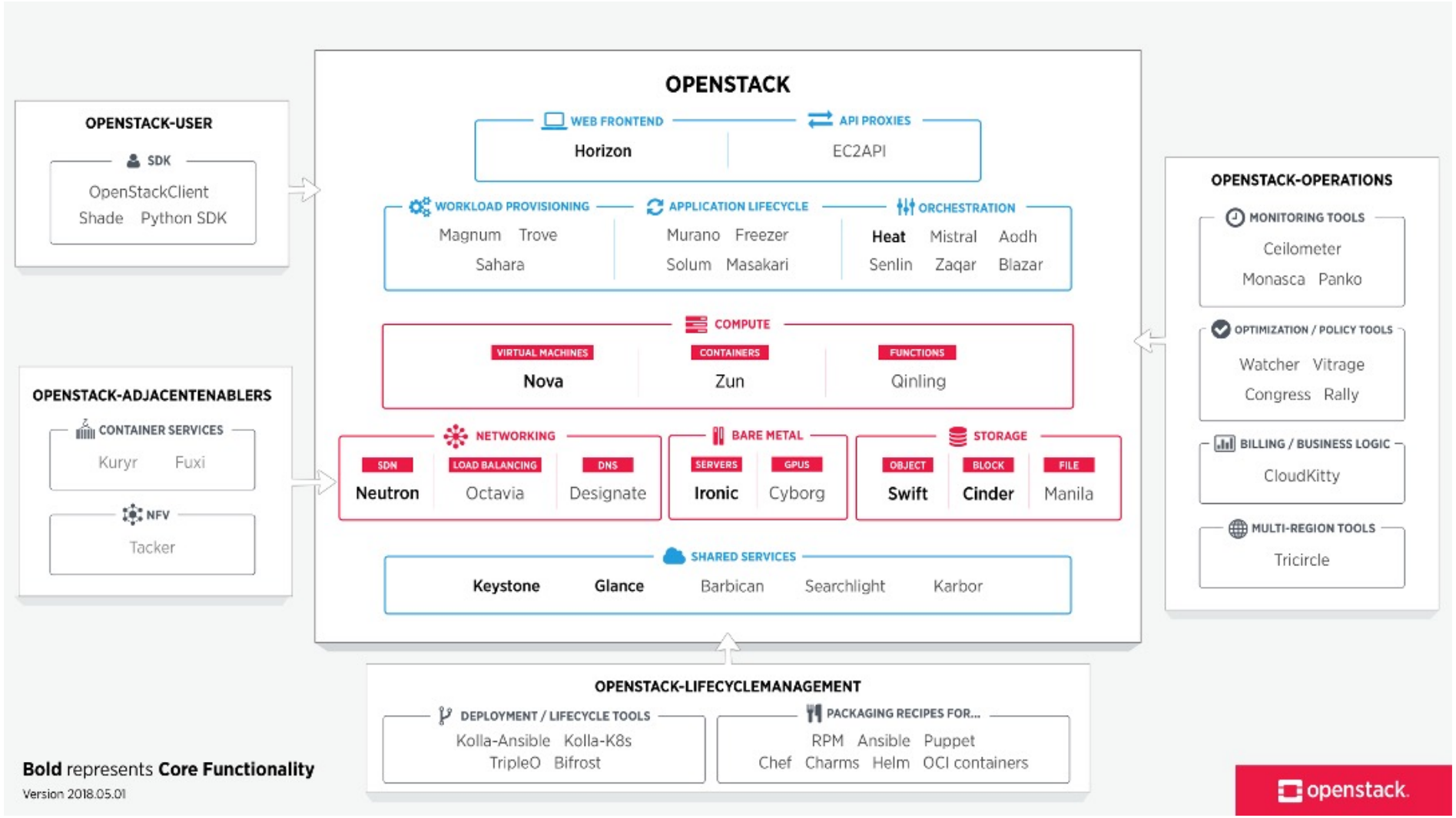
OpenStack Model, Architecture



OpenStack - Main components

- OpenStack Dashboard (Horizon) + CLI
- OpenStack Identity Management (Keystone)
- Compute service (Nova)
  - Flavors
  - SSH Access
  - Cloud Init
  - Security Group
- Networking service (Neutron)
- Image service (Glance)
- Block Storage service (Cinder)
- Object Storage Service (Swift)

OpenStack Components, Services

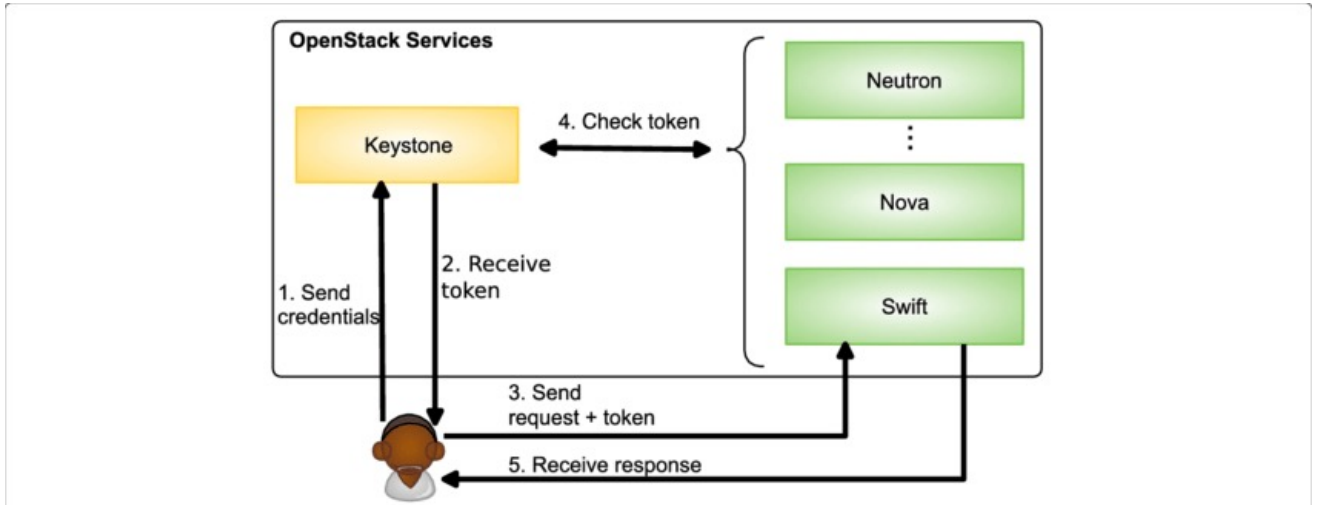


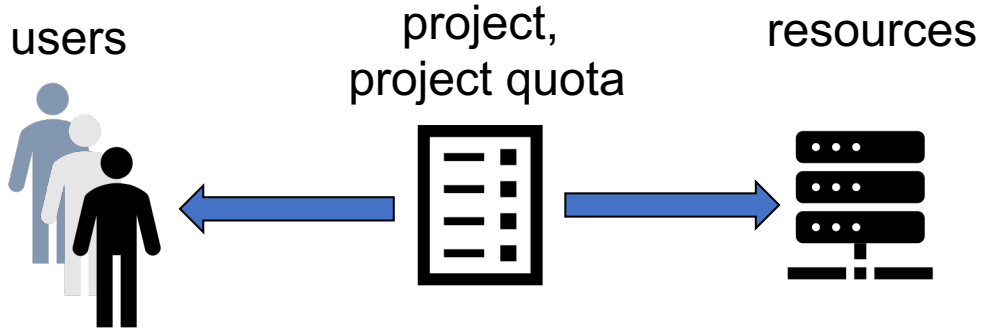
OpenStack Identity Management (Keystone)

Keystone is an OpenStack service that provides:

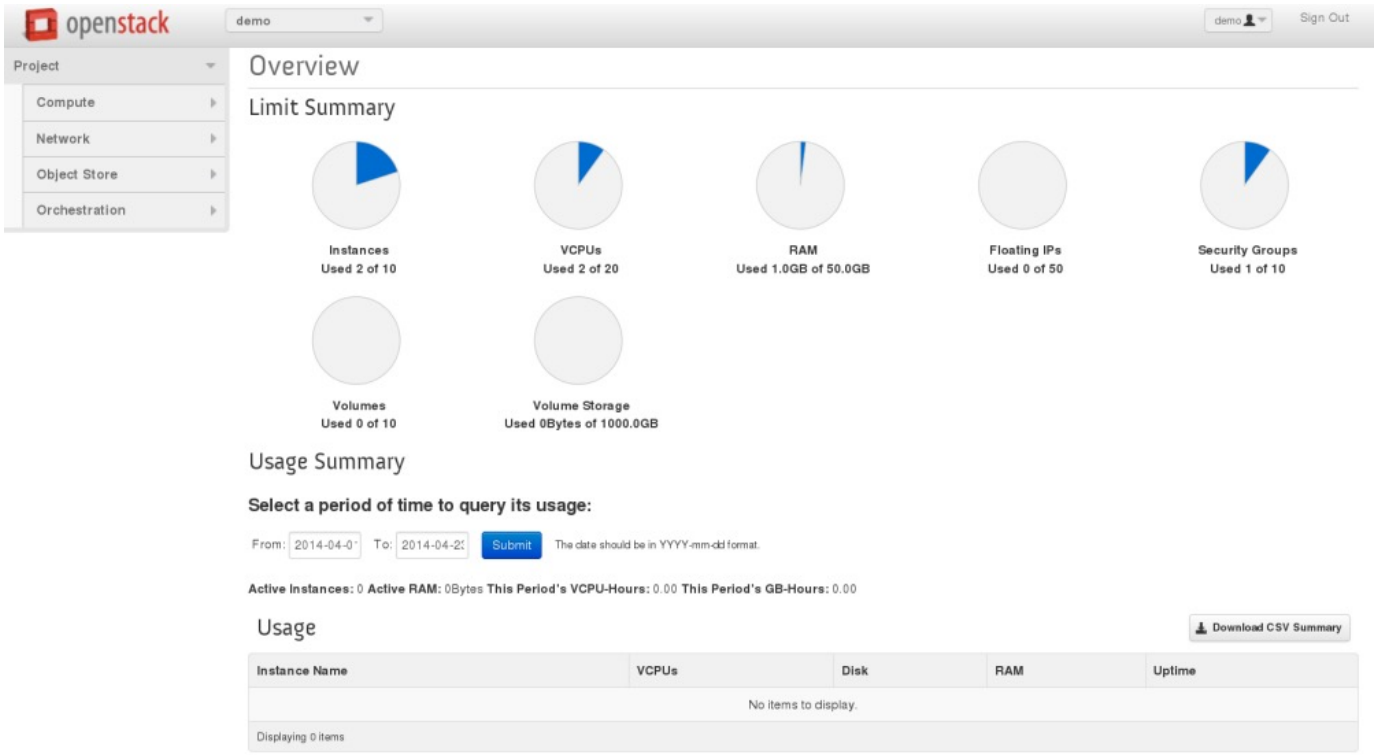
- API client authentication
- Service discovery
- Distributed multi-tenant authorization

by implementing OpenStack's Identity API.





OpenStack Dashboard (Horizon) + CLI



Horizon is the canonical implementation of OpenStack's Dashboard, which provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc.

OpenStack offers also a CLI

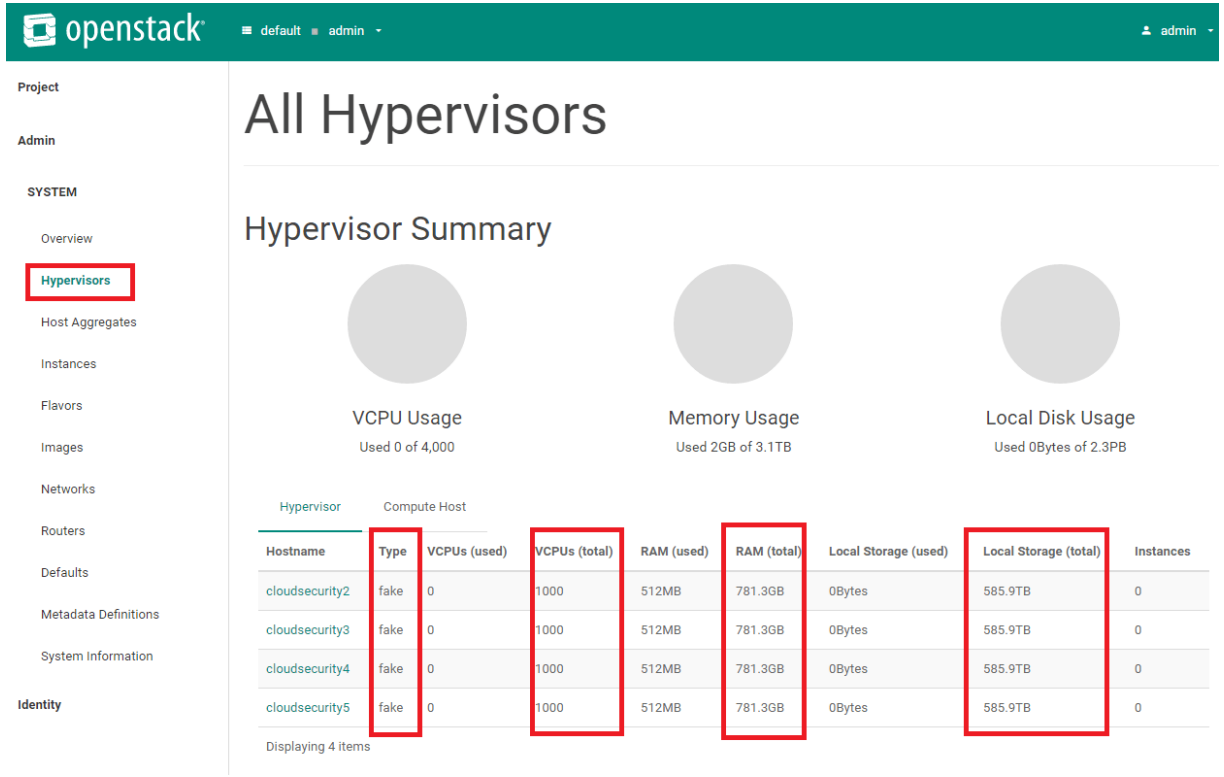
Both relies on top of the OpenStack API interfaces

OpenStack Dashboard (Horizon) + CLI

The OpenStack Dashboard is a web-based interface that allows you to manage OpenStack resources and services.

The Dashboard allows you to interact with the OpenStack Compute cloud controller using the Open- Stack APIs.

Dashboard is available with different functionalities for users (tenants) and for admins.



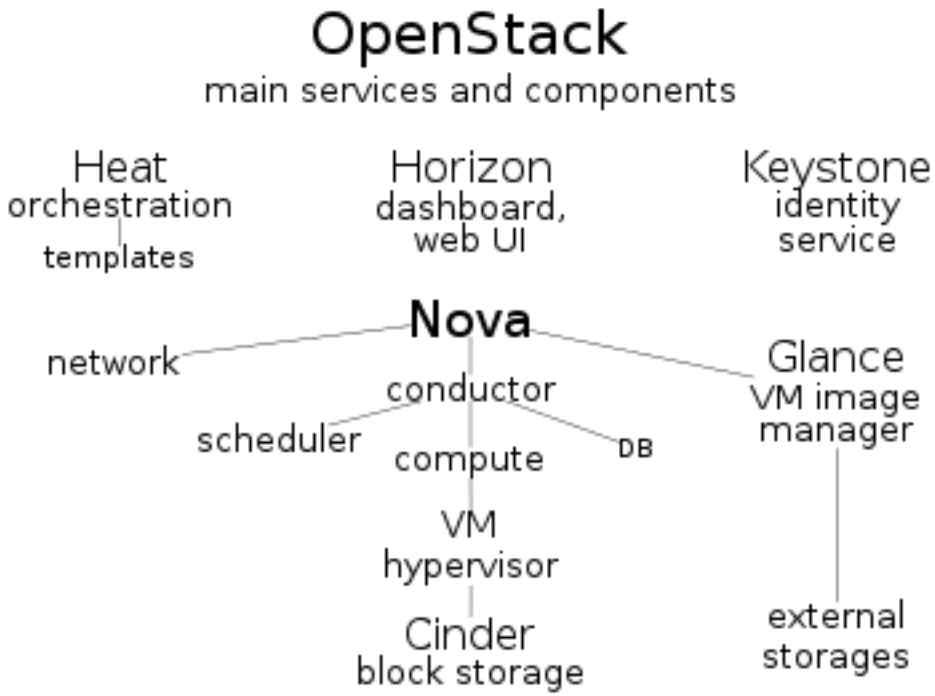
Compute service (Nova)

The OpenStack Compute service (nova) allows you to control an Infrastructure-as-a-Service (IaaS) cloud computing platform.

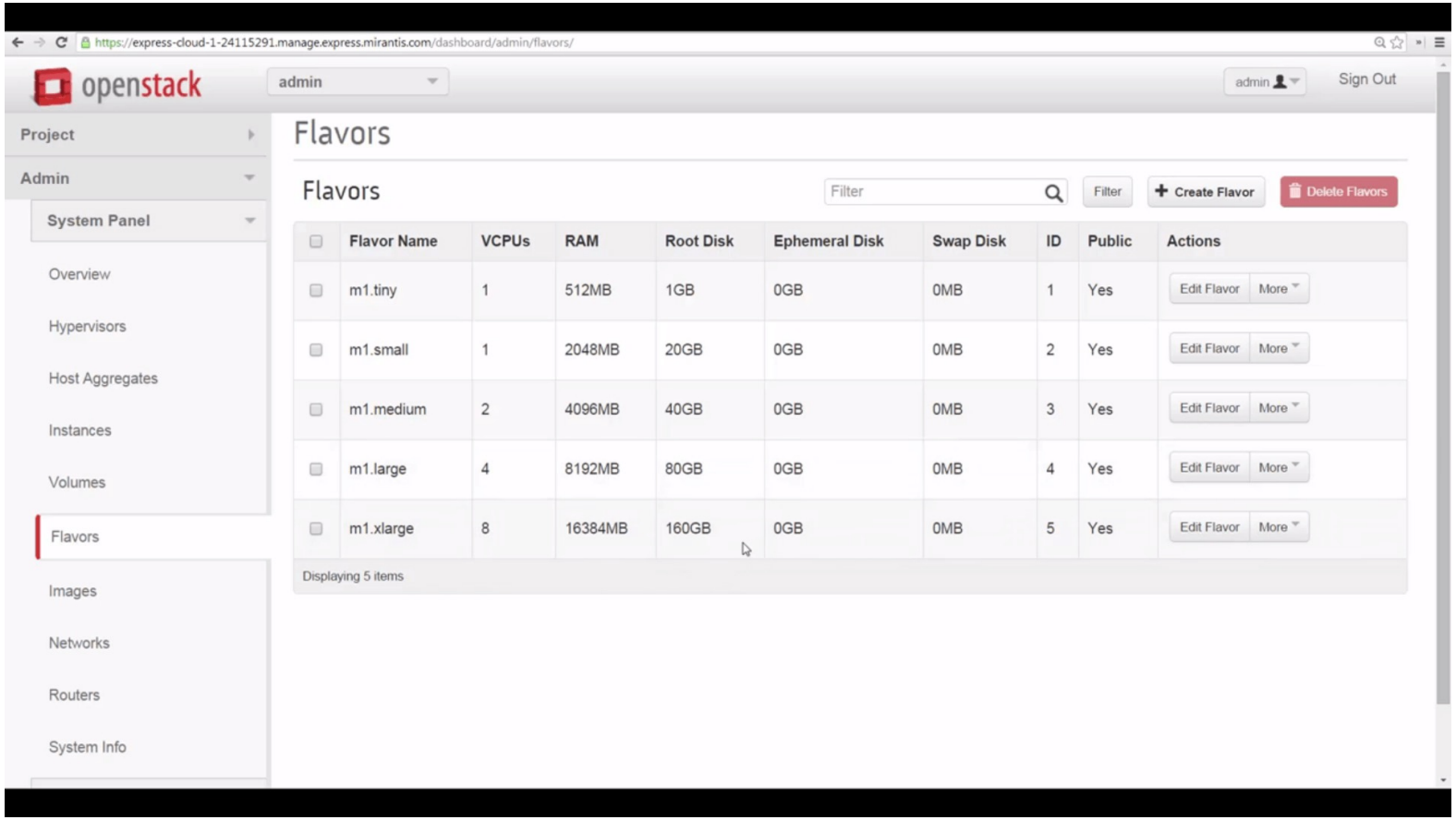
It gives you control over instances and networks, and allows you to manage access to the cloud through users and projects.

Compute does not include virtualization software.

Instead, it defines drivers that interact with underlying virtualization mechanisms that run on your host operating system, and exposes functionality over a web-based API.

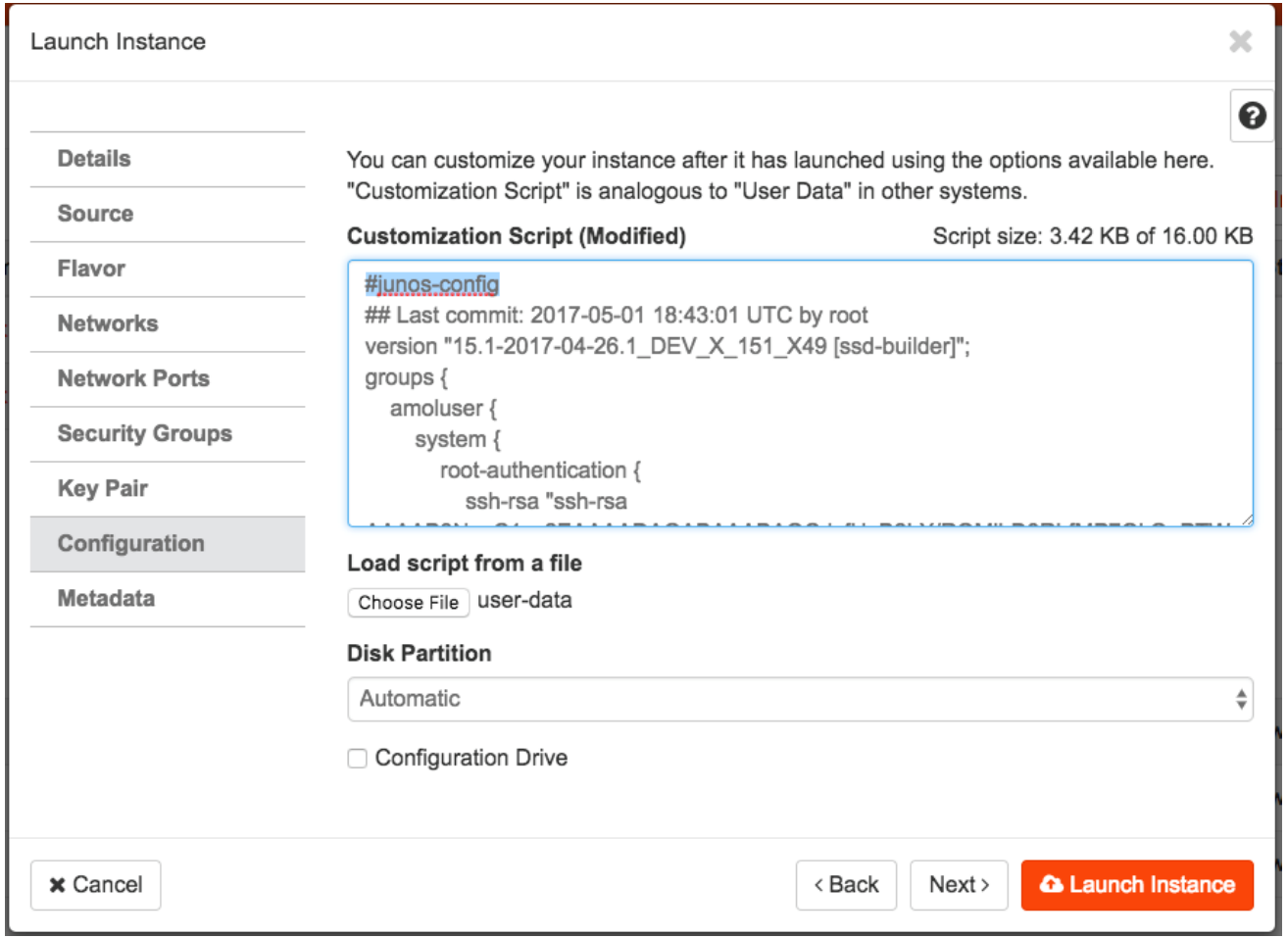


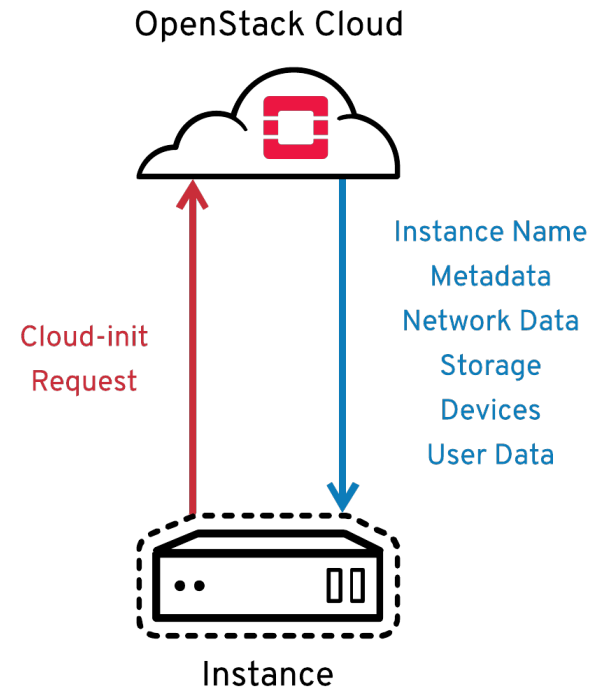
Compute service (Nova) - Flavors



Group different resources to assign to your VM under a name. Most common resources: RAM, DISK, Number of CPU (and usually a cost per Hour or Month or Year).

Compute service (Nova) – cloud-init





https://cloudinit.readthedocs.io/en/latest/

Compute service (Nova) - Security Group

Manage Security Group Rules: default (51d0e9bc-6fbd-40f7-a536-24a284290a79)

+ Add Rule | Delete Rules

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
Egress	IPv6	Any	Any	::/0	-	Delete Rule
Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	Any	Any	-	default	Delete Rule
Ingress	IPv6	Any	Any	-	default	Delete Rule

Displaying 4 items

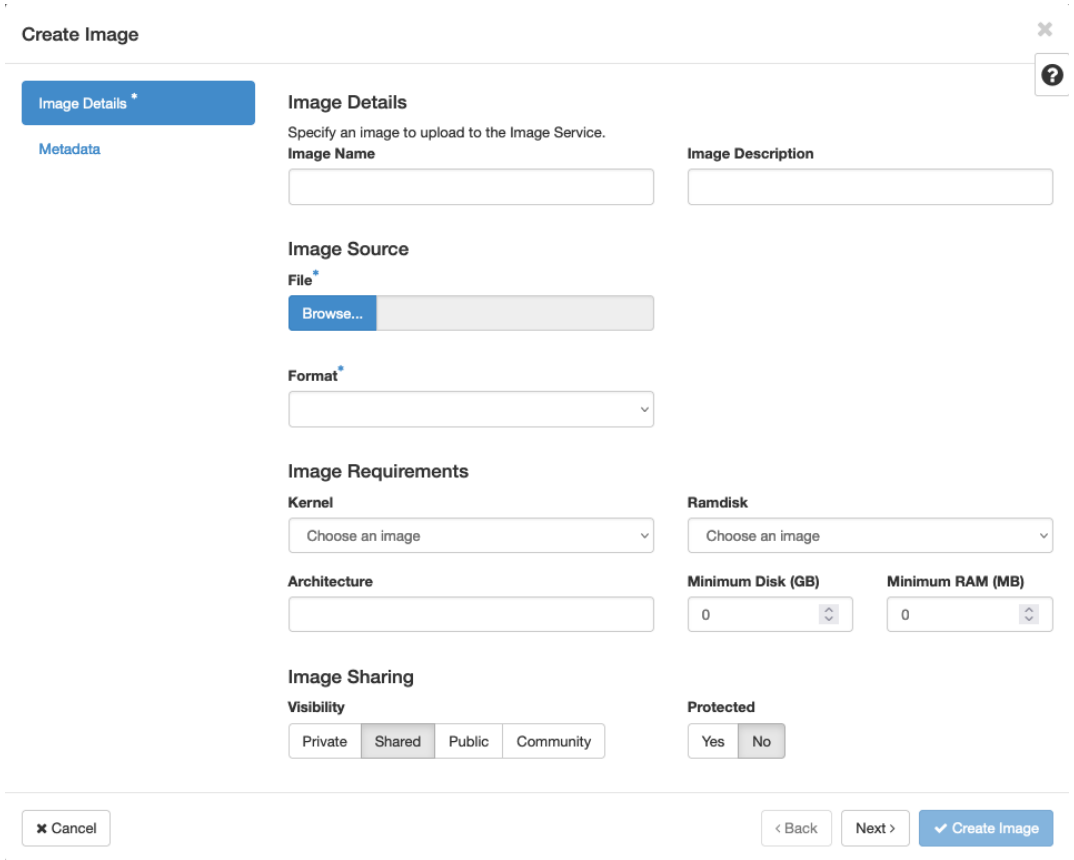
Think about them as a Firewall Rules managed by OpenStack for Group of Hosts.
Default policy is deny. Everything must be allowed explicitly.
An host can be part of multiple Security Groups.

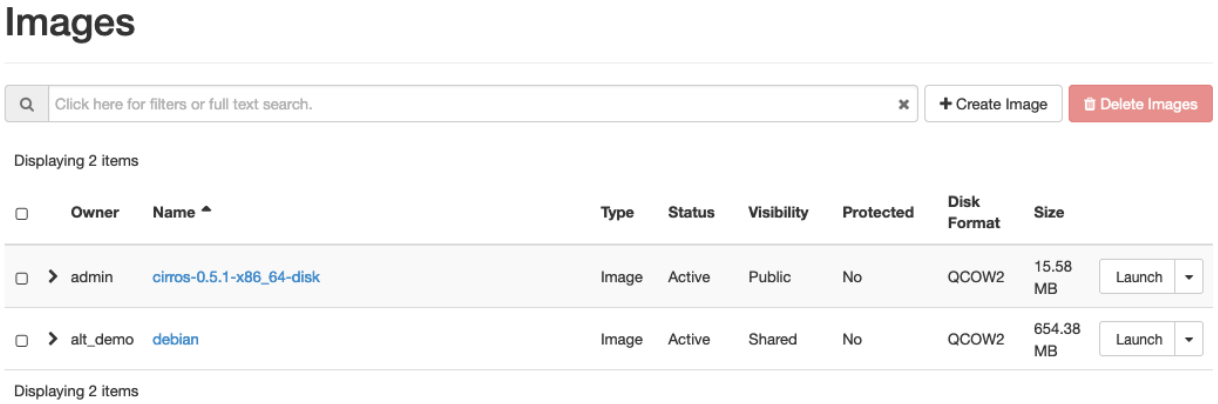
Compute service (Nova) - SSH Access

SSH keys are injected in the Instances (Virtual Machines) by OpenStack

Image service (Glance)

The Image service (glance) enables users to discover, register, and retrieve virtual machine images.





It offers a REST API that enables you to query virtual machine image metadata and retrieve an actual image.

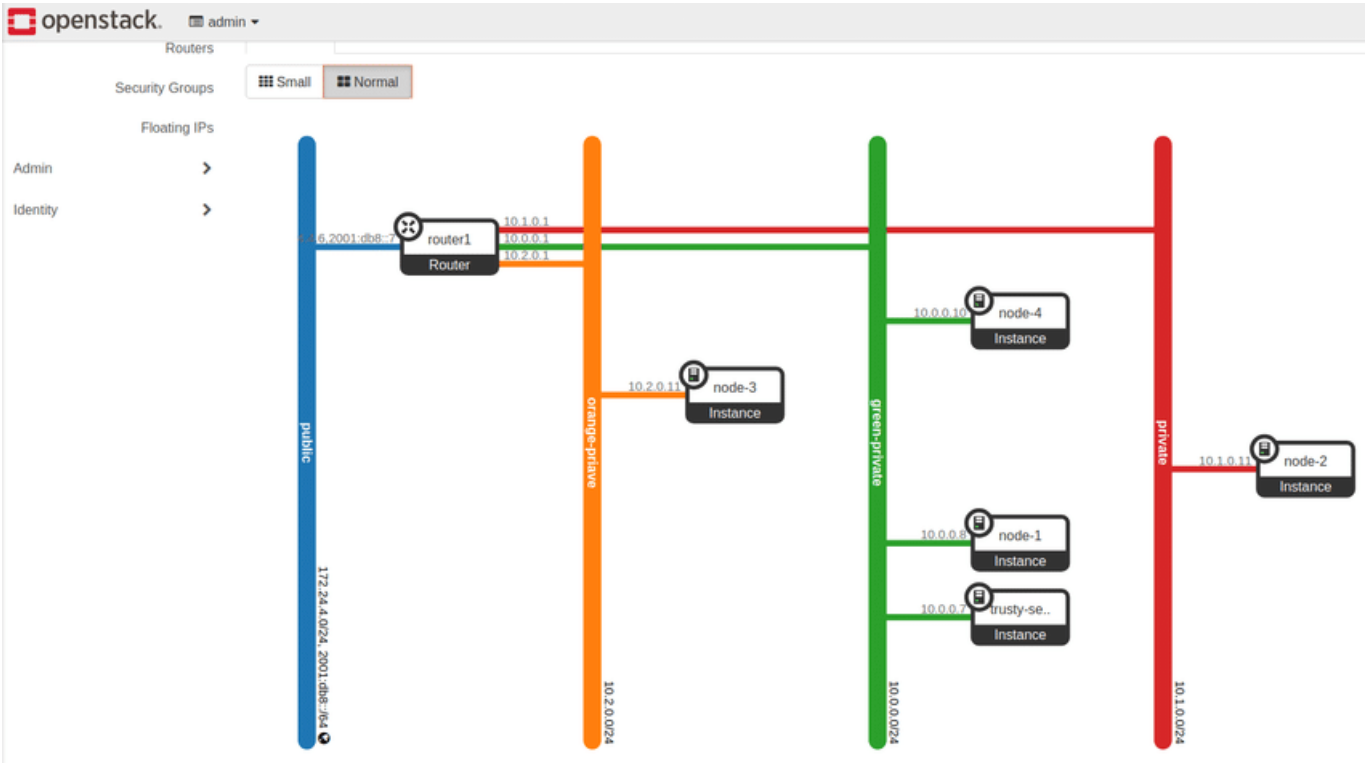
You can store virtual machine images made available through the Image service in a variety of locations, from simple file systems to object-storage systems like OpenStack Object Storage.

Networking service (Neutron)

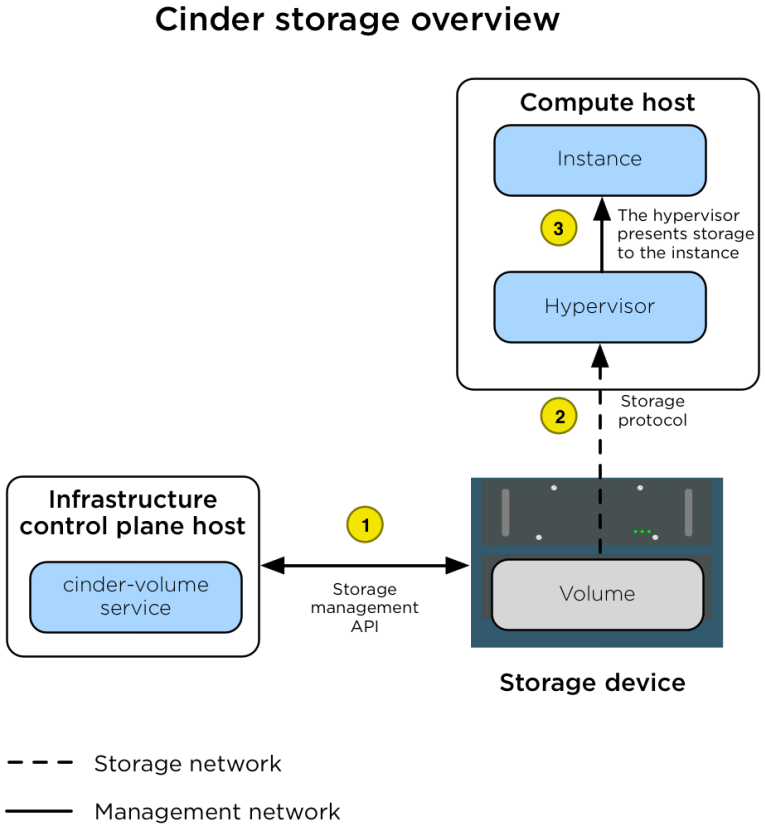
The Networking service, code-named neutron, provides an API that lets you define network connectivity and addressing in the cloud.

The Networking service enables operators to leverage different networking technologies to power their cloud networking.

The Networking service also provides an API to configure and manage a variety of network services ranging from L3 forwarding and NAT to load balancing, edge firewalls, and IPsec VPN.

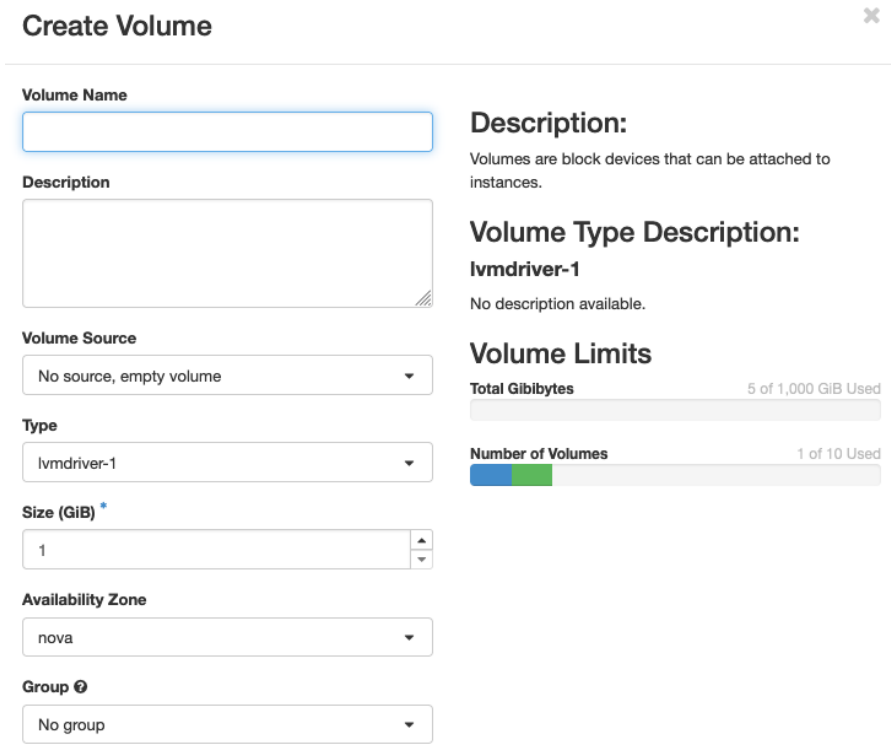


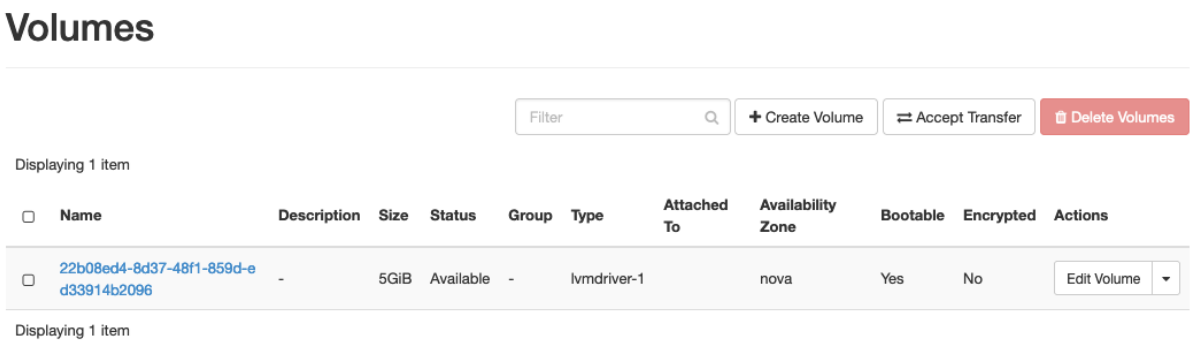
Block Storage service (Cinder)



The OpenStack Block Storage service (cinder) works through the interaction of a series of daemon processes named cinder-* that reside persistently on the host machine or machines.

Can be used as main instance disk or as an external disk (like a USB key)



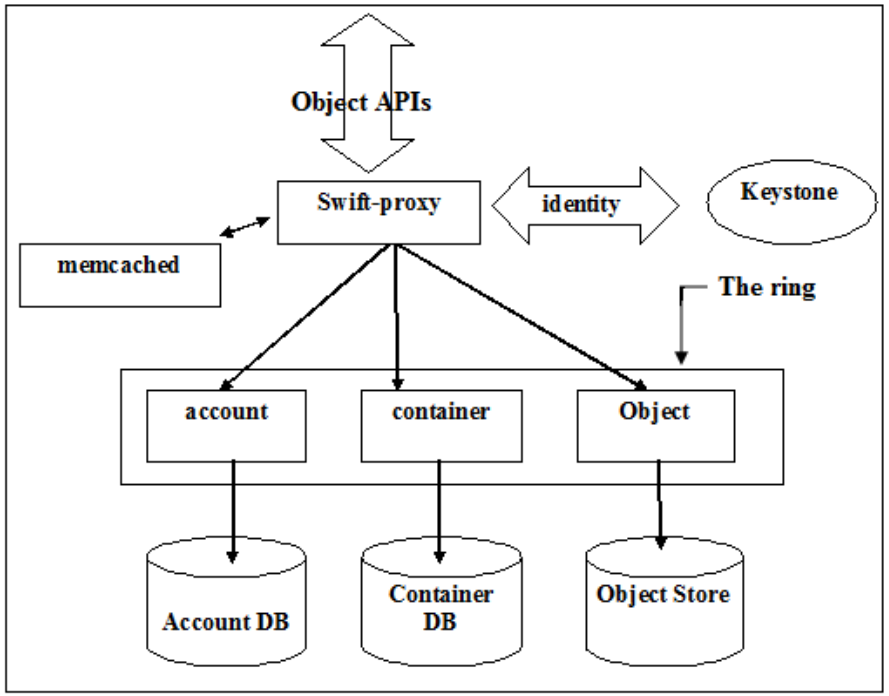


Object Storage Service (Swift)

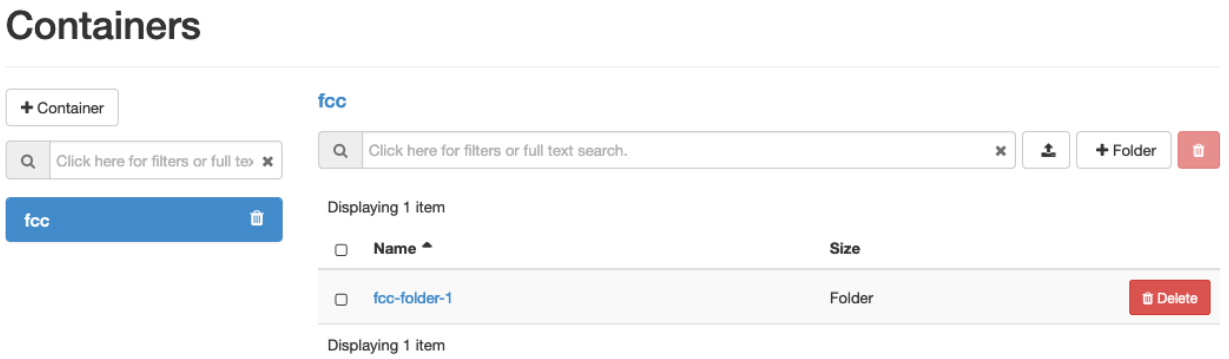
OpenStack Object Storage (swift) is used for redundant, scalable data storage using clusters of standardized servers to store petabytes of accessible data.

It is a long-term storage system for large amounts of static data which can be retrieved and updated via REST-API.

Object Storage uses a distributed architecture with no central point of control, providing greater scalability, redundancy, and permanence.



- Unstructured data such as music, images, and videos
- Backup and log files
- Large sets of historical data
- Archived files



Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, May 9th 2023

From manual GUI and CLI management to Infrastructure Automation

- **IaC (Infrastructure as Code)**
  - The process of managing and/or provisioning computer data centers through declarative machine-readable definition files.
- **CM (Configuration Management)**
  - The process for establishing and maintaining consistency of a product by maintaining systems, such as computer hardware and software, in a specific desired state
- **DevOps Workflow & Pipelines**
  - Set of automated processes and tools that allows both developers and operations teams to work cohesively to build and deploy code to a production environment.
    - CI: Continuous Integration (eg: every commit)
    - CD: Continous Deployement / Continous Delivery (eg: some companies run it every X hours ;D)

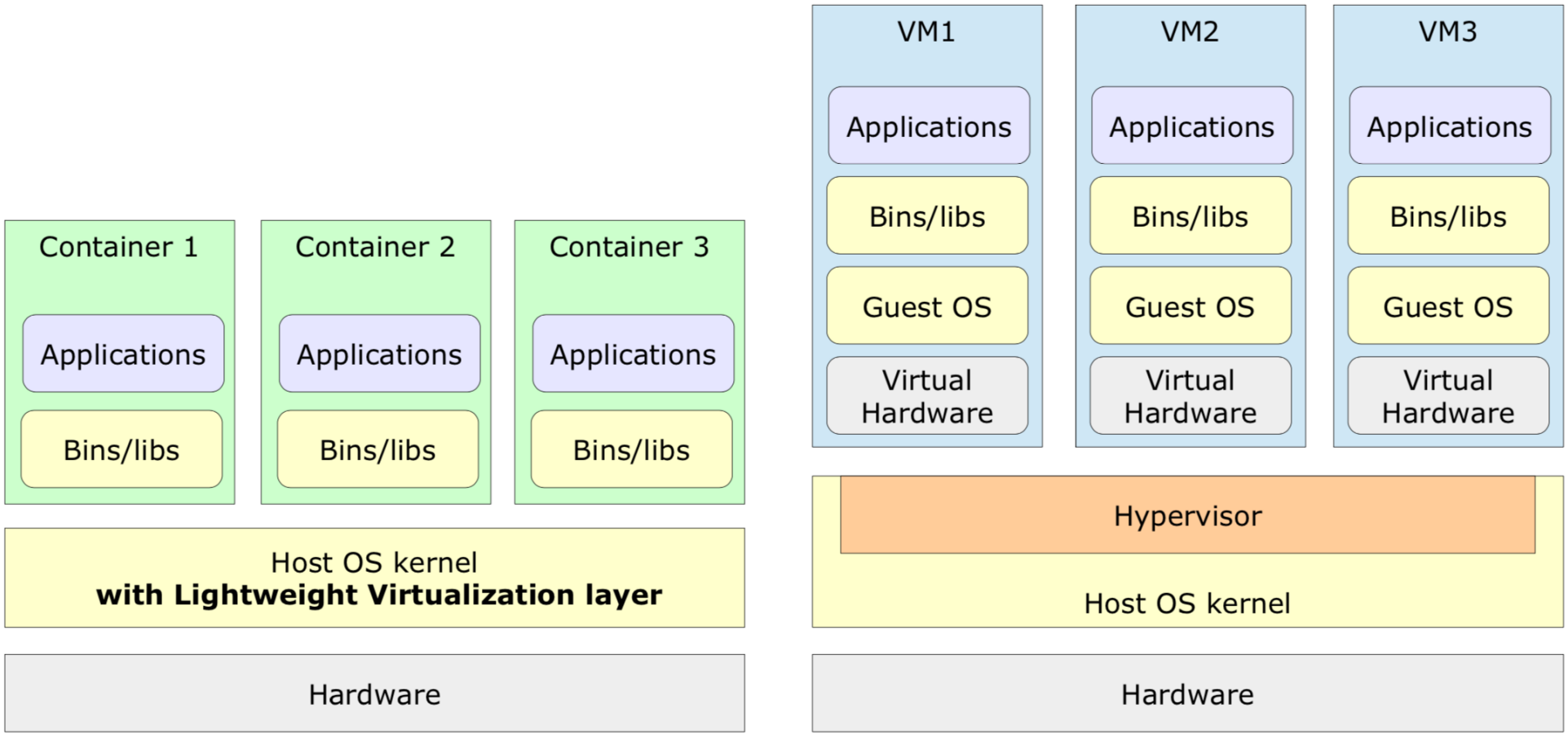
Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

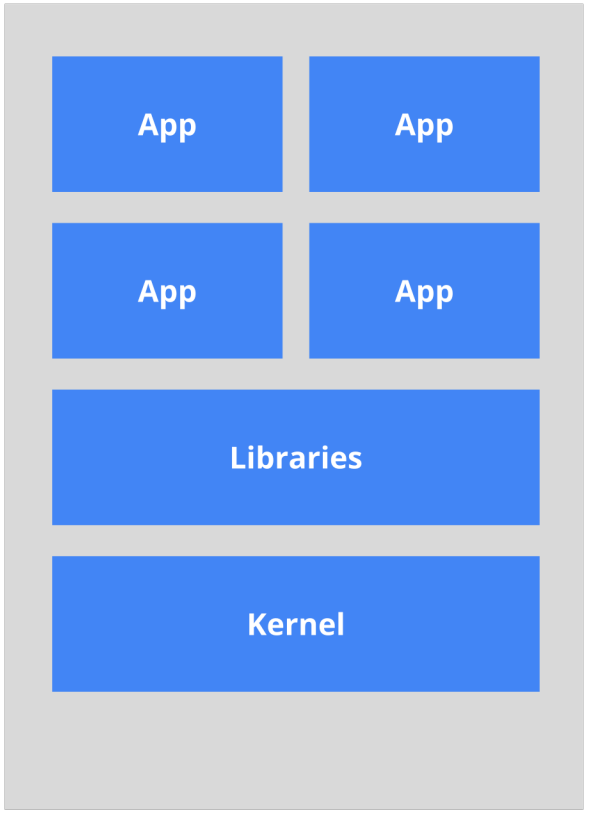
Trento, May 12th 2023

Containers vs Hypervisors (Infrastructure)



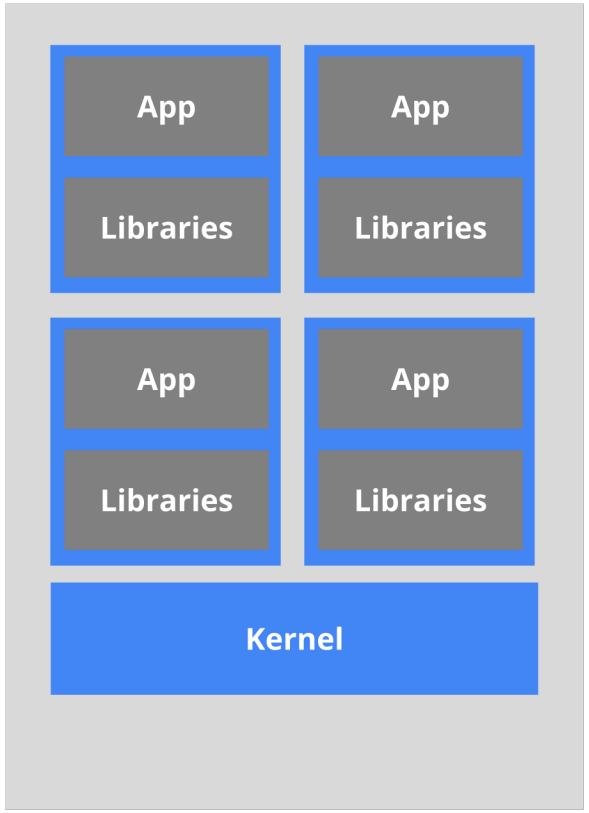
Containers VS Hypervisors (Software)

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



Small and fast, portable
Uses OS-level virtualization

Containers are an application-centric way to deliver high-performing, scalable applications on the infrastructure of your choice.

Containers benefits 1/3

- **Agile application creation and deployment**: Increased ease and efficiency of container image creation and deployment compared to VM.
- **Continuous development, integration, and deployment**: Provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
- **Dev and Ops separation of concerns**: Create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.

Containers benefits 2/3

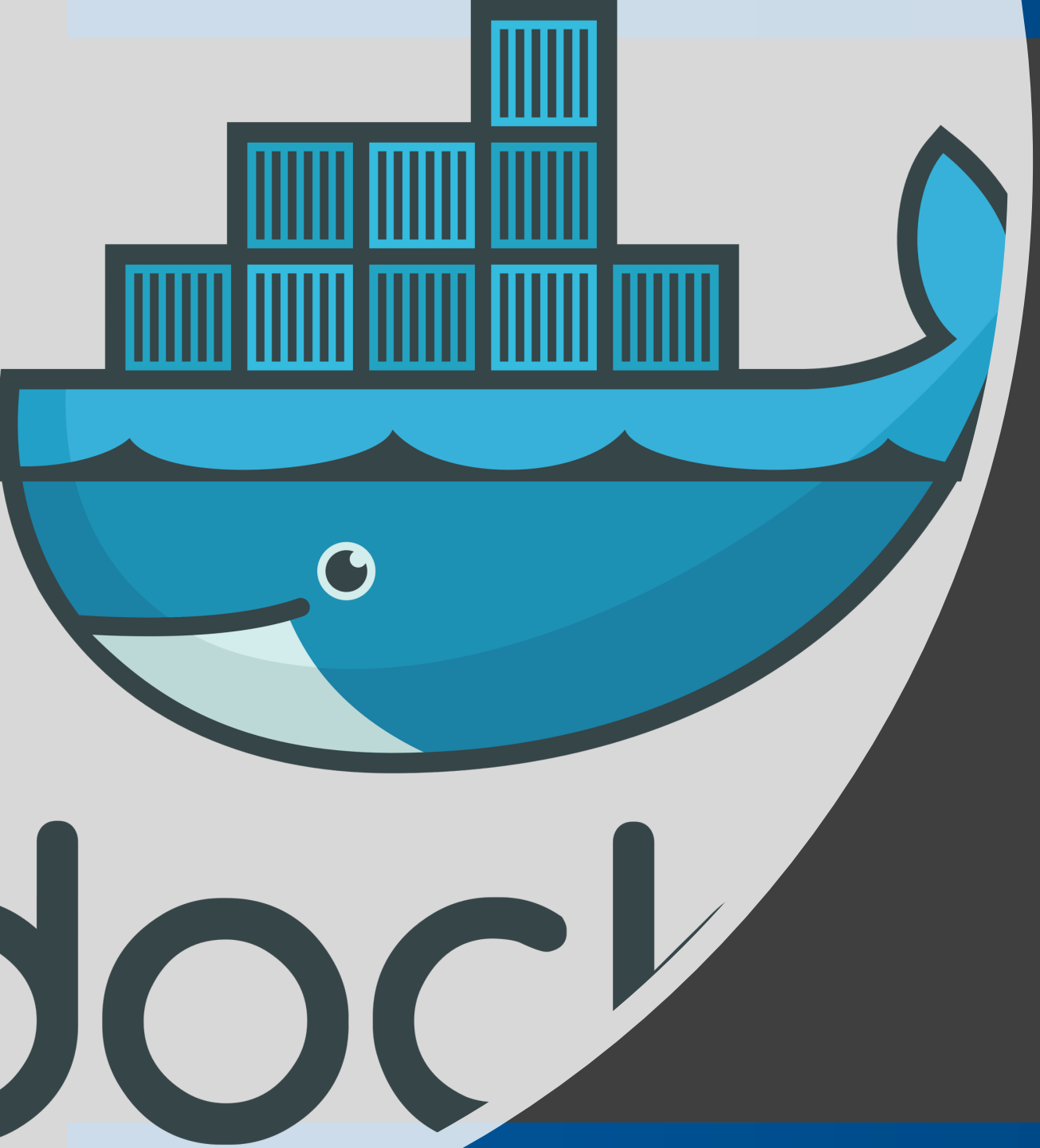
- **Observability** Not only surfaces OS-level information and metrics, but also application health and other signals.
- **Environmental consistency across development, testing, and production**: Runs the same on a laptop as it does in the cloud.
- **Cloud and OS distribution portability**: Runs on Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine, and anywhere else.

Containers benefits 3/3

- **Application-centric management**: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- **Loosely coupled, distributed, elastic, liberated micro-services**: Applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- **Resource utilization**: High efficiency and density.

Containers disadvantages

- Less isolation/security
  - Others weaknesses: Images distribution, Image scanning
- A solution
  - Combine Virtual Machines and Containers
- Kata containers
  - Kata Containers is an open source community working to build a secure container runtime with lightweight virtual machines that feel and perform like containers, but provide stronger workload isolation using hardware virtualization technology as a second layer of defense. **[**https://katacontainers.io/**]**



Docker

Docker

- OS level virtualization (*lightweight*)
- Relies on Linux kernel features: **cgroups** and **namespaces**
- Layered filesystem (similar as git commit)
  - Images as packaged containers derived incrementally from a pre-existing one
- Enable:
  - DevOps
  - Microservice architecture
  - Portability
- https://www.docker.com/ (docs: https://docs.docker.com/)
- The most popular but not the first:
  - A Brief History of Containers: From the 1970s Till Now

Docker: Images VS Containers

- **Docker Images [ref]**:
  - A read-only template with instructions for creating a Docker container.
    - Often, an image is *based on* another image, with some additional customization. *For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application.*
  - You might create your own images or you might only use those created by others and published in a registry.
  - To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image.
  - It is the object that makes your application portable.

Docker: Images VS Containers

- **Docker Containers [ref]**:
  - A runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
  - By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
  - A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

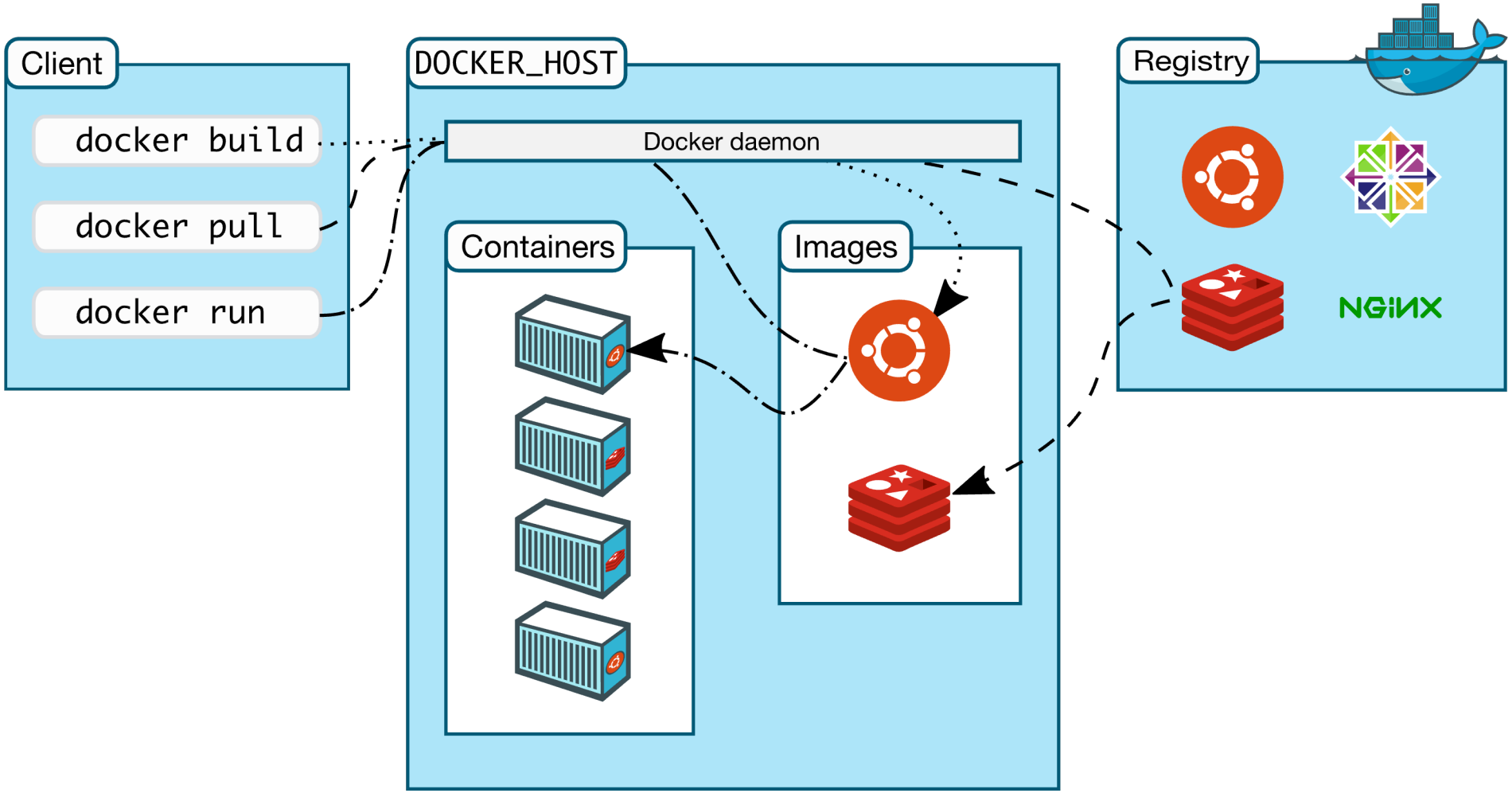
Docker: Images VS Containers

To use a computer science metaphor, if an image is a class, then a container is an instance of a class, in other words a runtime object.

Docker main components

- **Docker daemon** - the main process that manages containers
- **Docker Host** – the host (physical or virtual) where Docker daemon runs
- **Docker client** - for communicating with the Docker daemon
- **Docker registry** - image repository
  - https://hub.docker.com/

Docker Host Overview



Docker related Tools

- **Docker compose** - for deploying multi-container apps
  - https://docs.docker.com/compose/
- **Docker machine** - for setting up and manage remote Docker Hosts
  - https://docs.docker.com/machine/overview/
- **Docker swarm** - container orchestrator
  - https://docs.docker.com/engine/swarm/

Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

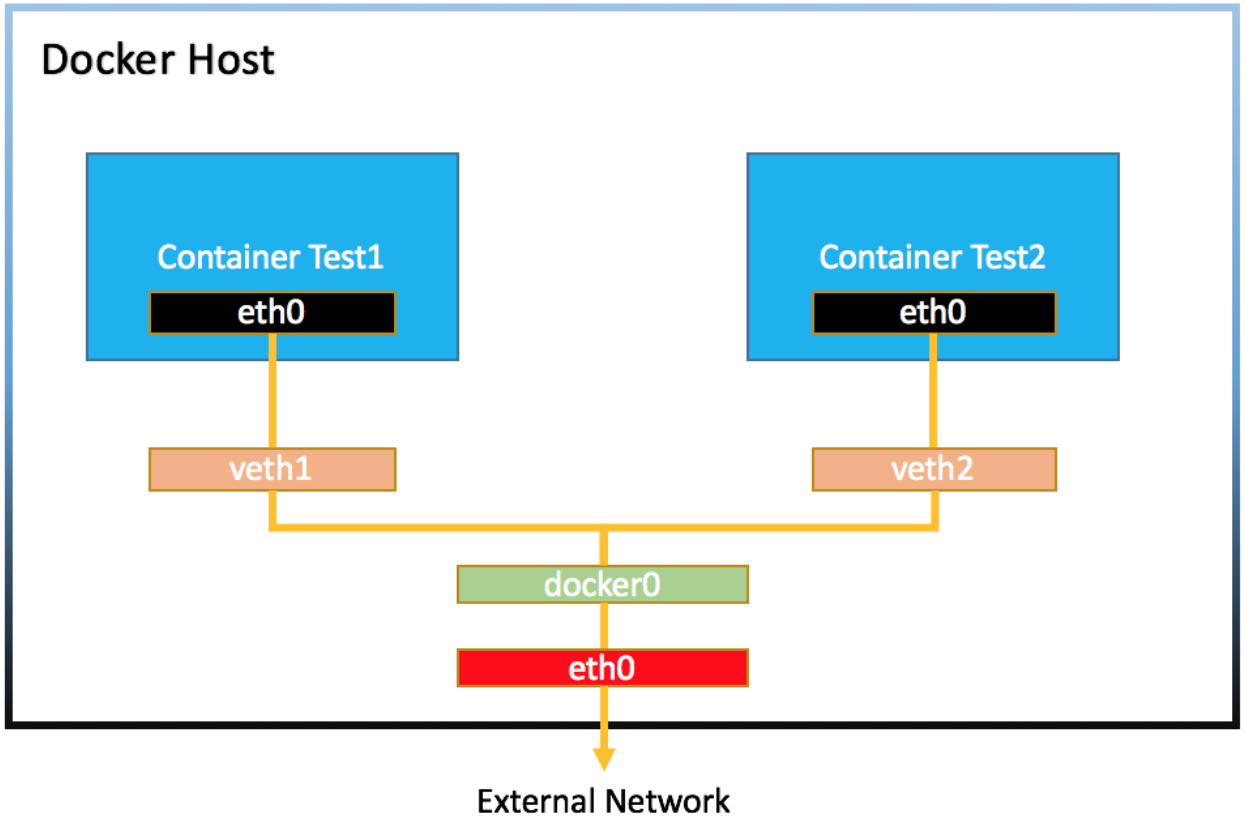
Trento, May 16th 2023

Docker Networking

- Docker offers many network functionalities:
  - Bridge Network (single host, default and user defined)
    - `docker network create mynet`
  - Host Network (no isolation, `--net=host`)
  - None (no network, `--net=none`)
  - Overlay Network (among different hosts)

- DNS service
  - Across containers on same network
  - Default using container name
  - `--alias` to customise

Docker default bridged network (docker0) schema



Docker Volumes

How to attach storage space to containers?

- Containers has an Ephemeral disk

- Docker offers few storage features:
  - Reference docs [ref]
  - Mount a data volume from a Docker Host on a container or create a Docker volume

Cloud Native

- *«**Cloud native** is a term used to describe container-based environments. **Cloud native** technologies are used to develop applications built with services packaged in containers, deployed as microservices and managed on elastic infrastructure through agile DevOps processes and continuous delivery workflows.» [ref]*

- **Cloud native** is about patterns to build software that scale on elastic infrastructure in fast way. *[ref]*

- Cloud Native Computing Foundation (CNCF)

- CN is about (not only) Dev and Ops (DevOps), two main concepts:
  - The Twelve Factors Methodology (Dev)
  - Pets vs Cattle (Ops)

The Twelve Factors

- Set of rules written by people working at the Heroku platform (https://www.heroku.com/)
- Metodology for writing apps (any language) that uses backing services (database, queue, memory cache, etc.)
- As of interset for: i) any developer building applications which run as a service and for ii) ops engineers who deploy or manage such applications.
- Reference: https://12factor.net/

The (first 3) Twelve Factors

I. Codebase

One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

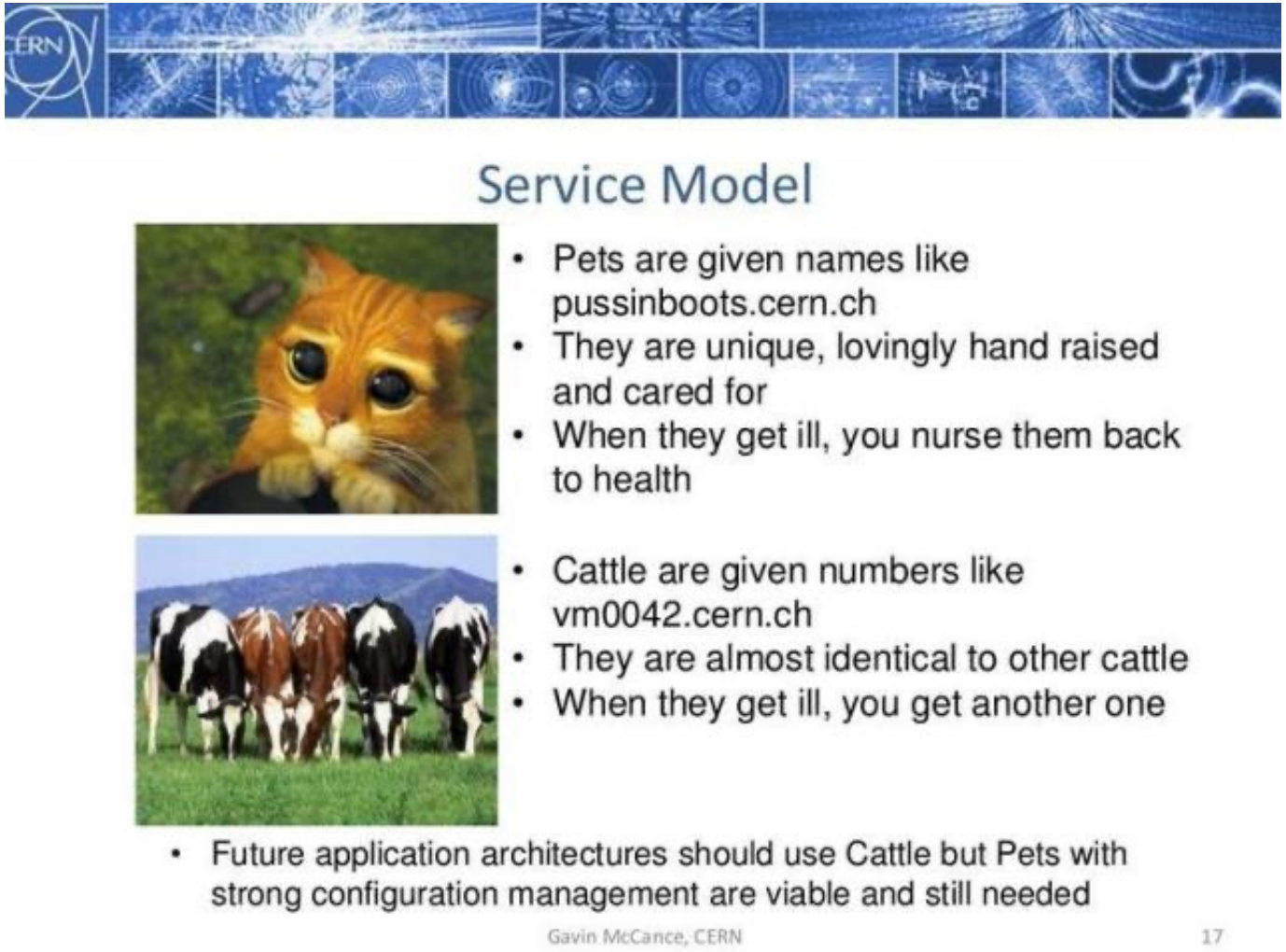
III. Config

Store config in the environment

Read by yourself the others and try to apply them next time you write SW

Pets and Cattle

- First concept from Bill Baker about **Scaling SQL Server**
  - The History of Pets vs Cattle and How to Use the Analogy Properly [ref]
- DevOps Concepts: Pets vs Cattle [ref]



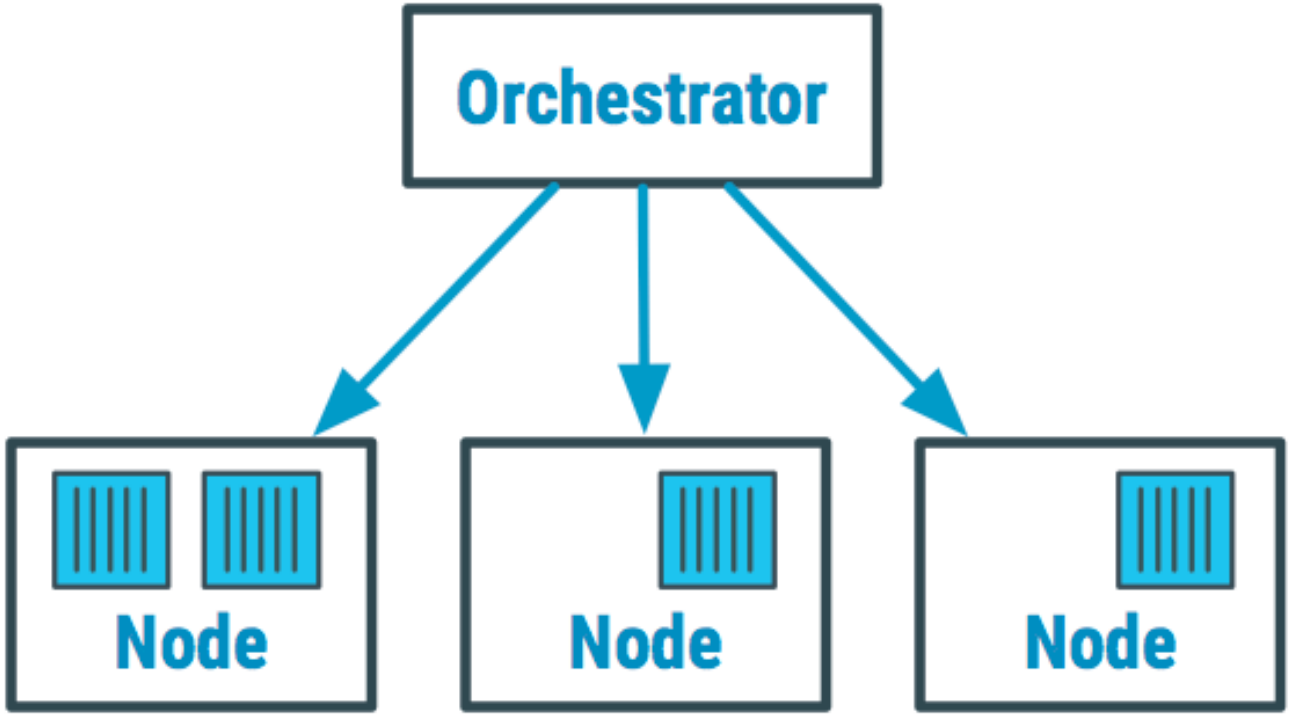
Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, May 16th 2023

Container Orchestration



Container Orchestration 1/2

In *Development* and *Quality Assurance (QA)* environments, we can get away with running containers on a single host to develop and test applications.

However, when we go to *Production*, we do not have the same liberty, as we need to ensure that our applications:

- Are fault-tolerant
- Can scale, and do this on-demand
- Use resources optimally
- Can discover other applications automatically, and communicate with each other
- Are accessible from the external world
- Can update/rollback without any downtime.

Container Orchestration 2/2

- Container Orchestrators are the tools which group hosts together to form a cluster, and help us fulfill the requirements mentioned before.

- Everything at Google runs in a container [ref]
  - In 2014 Google was starting over two billion containers per week (~3300 containers per second)

- Clear and simple explanations of containers orchestration:
  - https://www.youtube.com/watch?v=HDt_iN1hINA
  - https://www.youtube.com/watch?v=kBF6Bvth0zw

Q: Why use Container Orchestrators ?

A: It is all about SCALING

BRING MULTIPLE HOSTS TOGETHER AND MAKE THEM PART OF A CLUSTER

SCHEDULE CONTAINERS TO RUN ON DIFFERENT HOSTS

HELP CONTAINERS RUNNING ON ONE HOST REACH OUT TO CONTAINERS RUNNING ON OTHER HOSTS IN THE CLUSTER

BIND CONTAINERS AND STORAGE

BIND CONTAINERS OF SIMILAR TYPE TO A HIGHER-LEVEL CONSTRUCT, LIKE SERVICES, SO WE DON'T HAVE TO DEAL WITH INDIVIDUAL CONTAINERS

KEEP RESOURCE USAGE IN-CHECK, AND OPTIMIZE IT WHEN NECESSARY

ALLOW SECURE ACCESS TO APPLICATIONS RUNNING INSIDE CONTAINERS.

What is Kubernetes, aka k8s ?

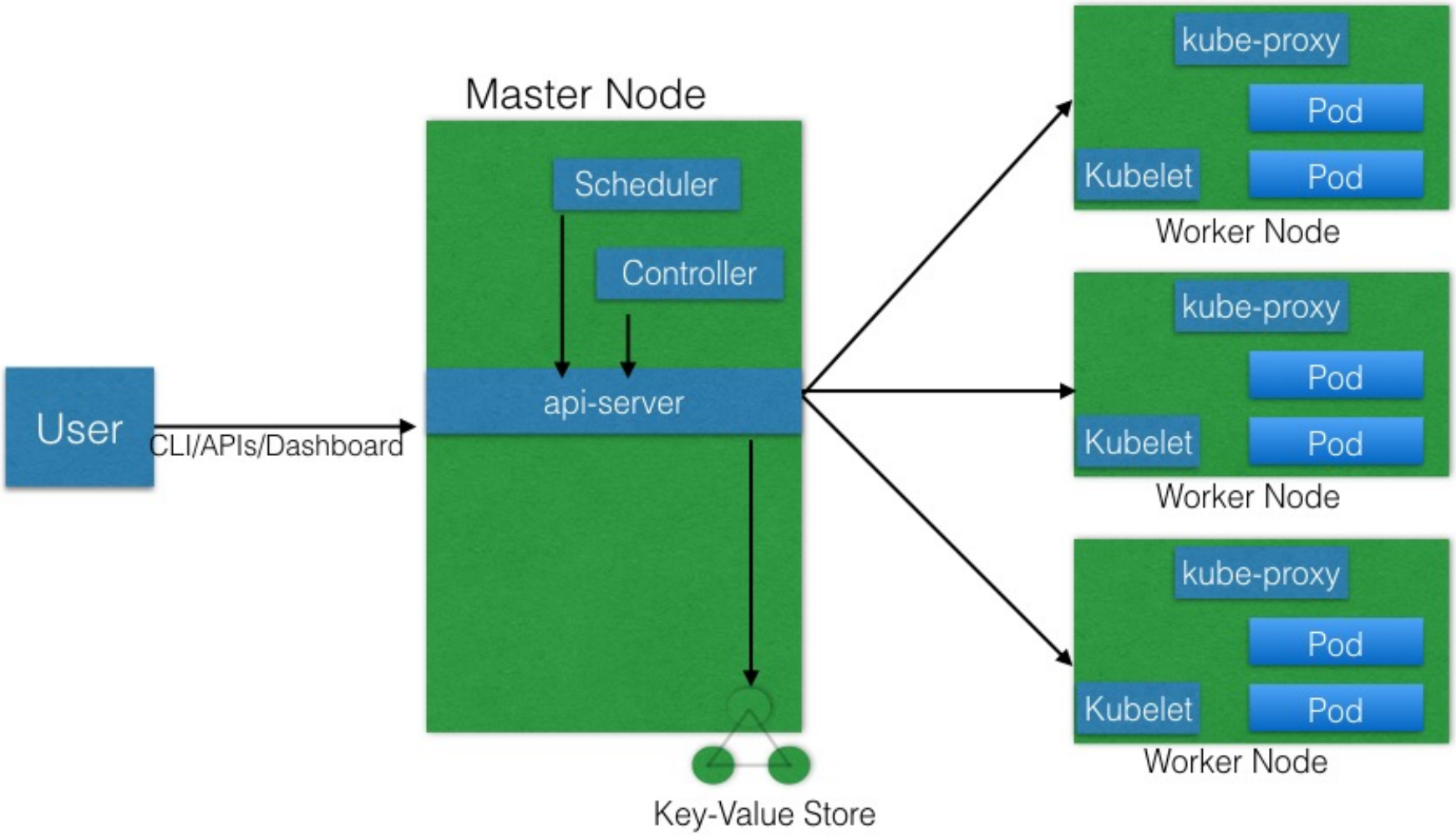
- From k8s website: *"Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.»*

- Highly inspired by the Google Borg system [ref]

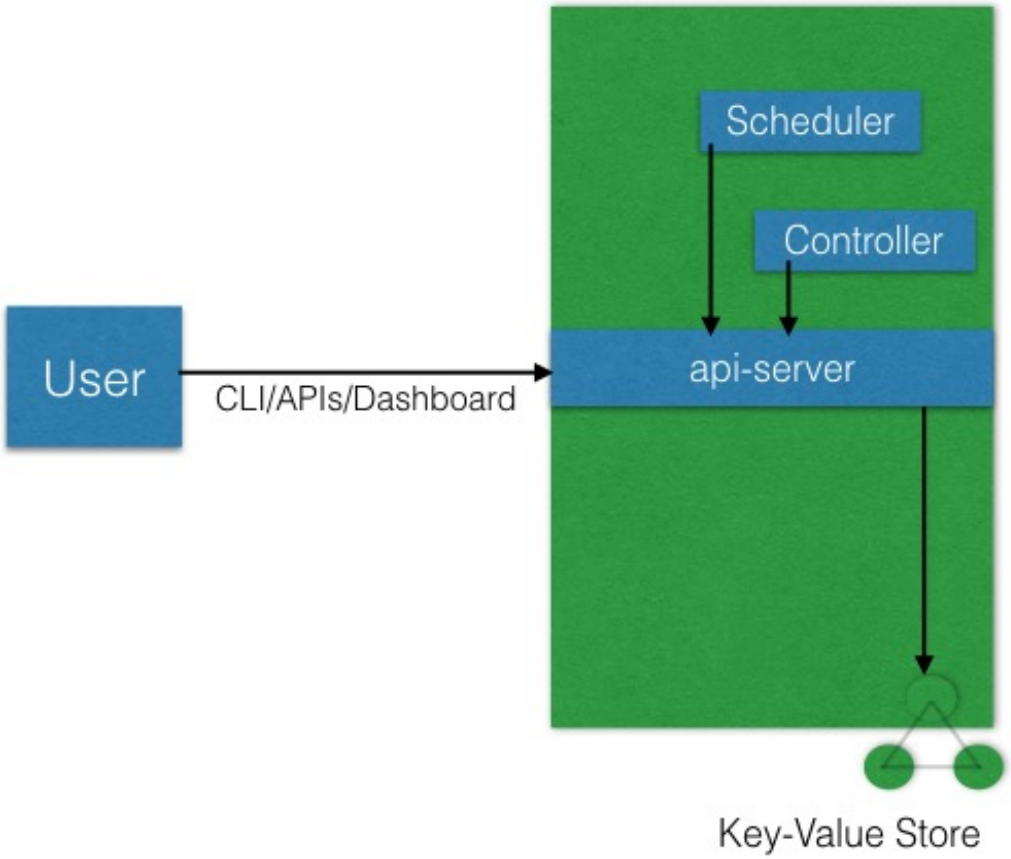
- Started by Google and (from v1.0 release in July 2015) donated to the CNCF

- k8s community recently defined it: a *"Platform to build other platforms» → CRD (Custom Resource Definition)*

Kubernetes Architecture



Master Node



- The Master Node is responsible for managing the Kubernetes cluster, and it is the entry point for all administrative tasks. We can communicate to the Master Node via the CLI, the GUI (Dashboard), or directly via APIs
- If we have more than one Master Node, they would be in a HA (High Availability) mode, and only one of them will be the leader, performing final operations and decisions
- To manage the cluster state, Kubernetes uses **etcd**, and all Master Nodes connect to it. **etcd** is a distributed key-value store

Master Node Components

- **API Server:** A user/operator sends REST commands to the API Server, which then **validates** and **processes** the requests. After executing the requests, the resulting **state of the cluster is stored** in the distributed key-value store.

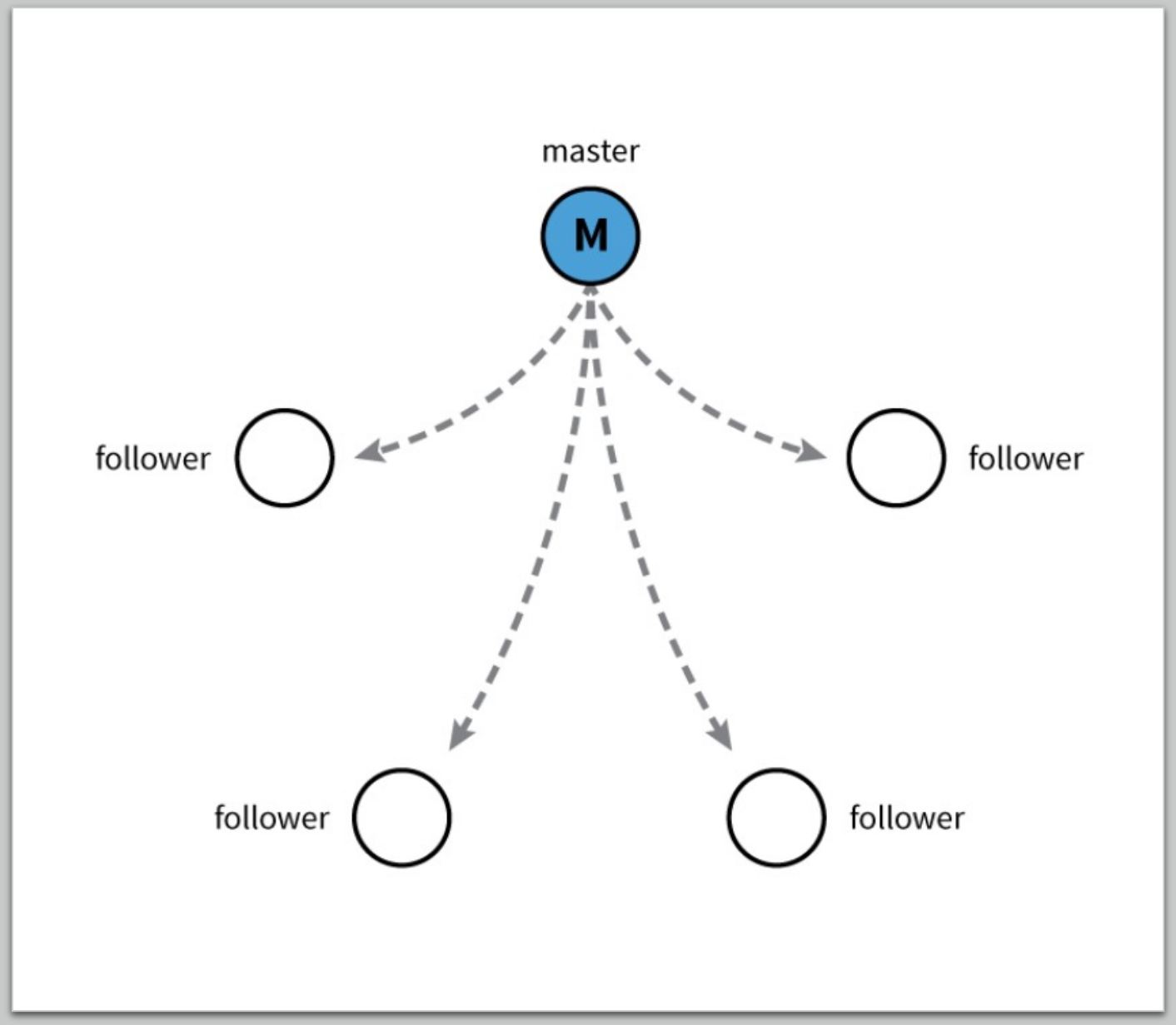
- **Scheduler:** Schedules the work to different Worker Nodes. The Scheduler has the resource usage information for each Worker Node. Before scheduling the work, it also takes into account the quality of the service requirements, data locality, affinity, anti-affinity, etc. eg: `disk==ssd`

- **Controller Manager:** Manages different non-terminating control loops, which regulate the state of the Kubernetes cluster. Each control loop knows about **the desired state** of **the objects it manages**, and watches their **current state** through the API Server. If the current state of the object does not meet the desired state, then it takes corrective steps to make sure that the current state is the same as the desired state.

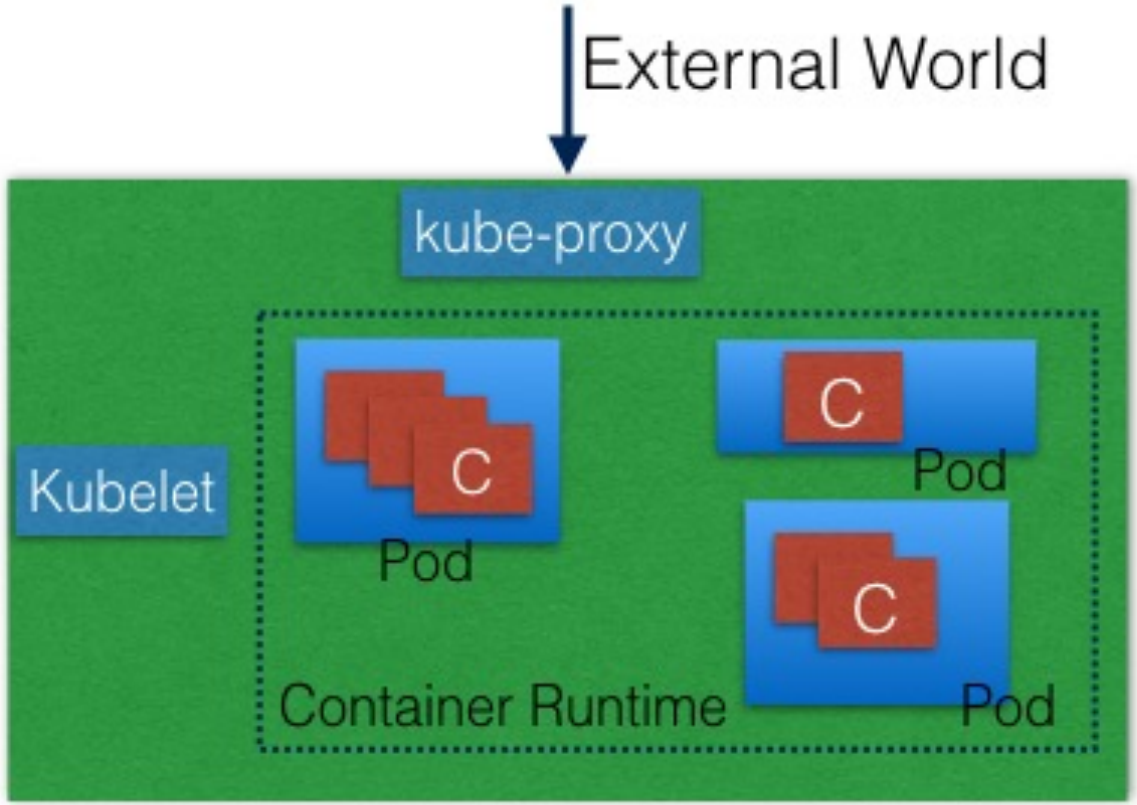
- **etcd:** Distributed key-value store which is used to store the cluster state. It can be part of the Kubernetes Master, or, it can be configured externally, in this case, Master Nodes would connect to it.

State Management

- **etcd** is a distributed key-value store based on the Raft Consensus Algorithm.
- Allow a collection of machines to work as a coherent group that can survive failures (of some members)
- At any given time one node is the master, the rest are followers
- Any node can be the Master
- etcd playground: http://play.etcd.io/play



Worker Node



- A Worker Node is a machine (VM, physical server, etc.) which runs the applications by means of Pods and is controlled by the Master Node.
- Pods are scheduled on the Worker Nodes, which have the *necessary tools* to run and connect them.
- A Pod is **the scheduling unit** in Kubernetes. It is a logical collection of one or more containers which are always scheduled together.
- To access the applications from the external world, (normally) we connect to Worker Nodes and not to the Master Node/s.

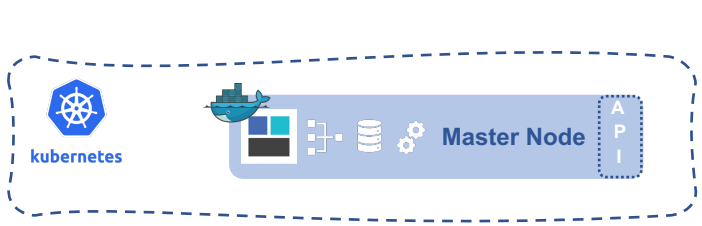
Worker Node Components

- **Container Runtime:** To run containers, we need a Container Runtime on the Worker Node. By default, Kubernetes is configured to run containers with Docker (*containerd from 2019*). It can also run containers using the rkt Container Runtime.

- **kubelet:** An agent which runs on each Worker Node and communicates with the Master Node. It receives the Pod definition via various means (primarily, through the API Server), and runs the containers associated with the Pod. It also makes sure the containers which are part of the Pods are healthy at all times.

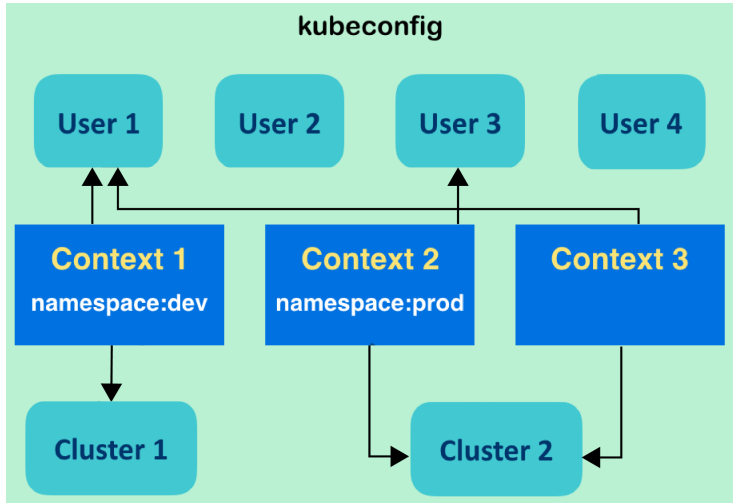
- **kube-proxy:** Instead of connecting directly to Pods to access the applications, we use a **logical construct called Service as a connection endpoint**. A Service groups related Pods, which it load balances when accessed.
  kube-proxy is the network proxy which runs on each Worker Node and listens to the API Server for each Service endpoint creation/deletion. For each Service endpoint, **kube-proxy** sets up the routes so that it can reach to it.

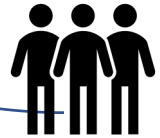
Communicate with your cluster via API - kubectl



kubeconfig

```yaml
apiVersion: v1
clusters:
  - cluster:
      certificate-authority-data: XXX
      server: https://127.0.0.1:33961
    name: kind-kind
contexts:
  - context:
      cluster: kind-kind
      user: kind-kind
    name: kind-kind
current-context: kind-kind
kind: Config
preferences: {}
users:
  - name: kind-kind
    user:
      client-certificate-data: XXX
      client-key-data: XXX
```





kubectl

Kubernetes Object Model

- Kubernetes has a very rich (and extensible) object model with which it represents different persistent entities in the cluster.
  Those entities describes for example:
  - What containerized applications we are running and on which node
  - Application resource consumption
  - Different policies attached to applications, like restart/upgrade policies, fault tolerance, etc.

- Within each object:
  - **We** declare our intent or desired state using the **`spec`** field
  - **Kubernetes** records the actual state in the **`status`**

- At any given point in time, the Kubernetes Control Plane (by means of controllers) tries to **match the object's actual state to the object's desired state**

- Examples of Kubernetes objects are *Pods, Deployments, ReplicaSets,* etc.

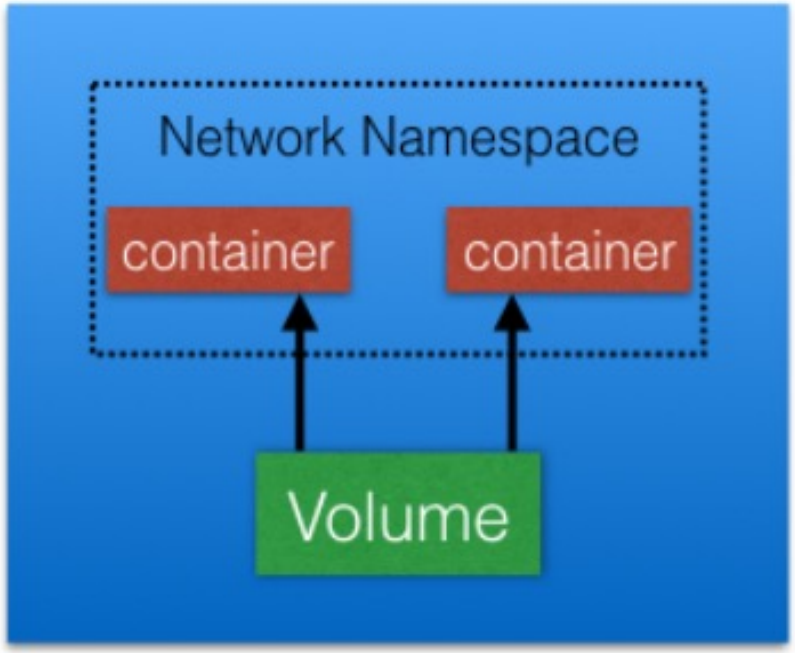
- Most of the time, we provide an object's definition in a **.yaml** file, which is converted by **kubectl** in a JSON payload and sent to the API Server.

```yaml
1  kind: Pod
2  apiVersion: v1
3  metadata:
4    name: clock
5    namespace: default
6  spec:
7    containers:
8    - name: clock
9      image: jpetazzo/clock
```

A simple Pod object: pods/simple-pod.yaml

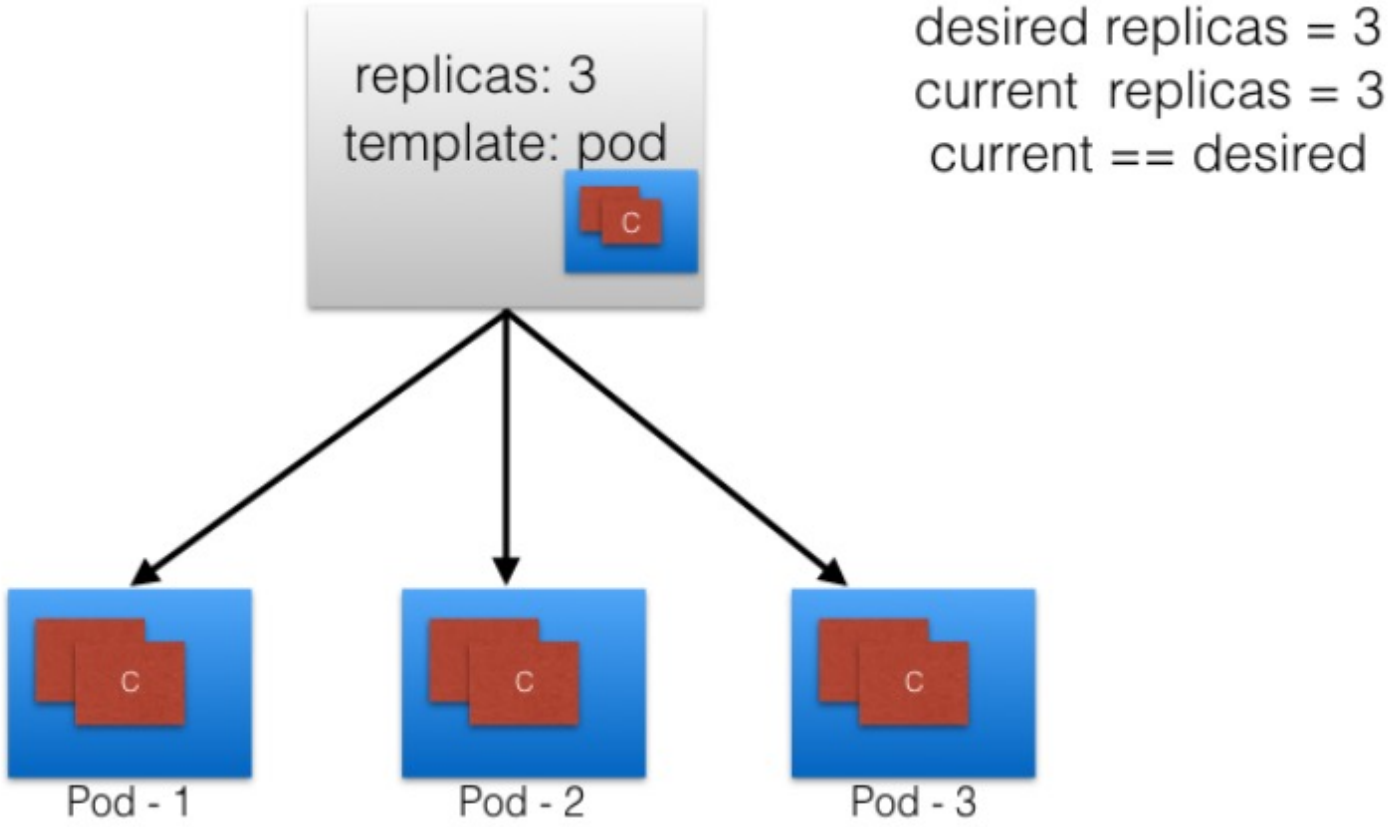
With the **apiVersion** field in the example above, we mention the API endpoint on the API Server which we want to connect to.

Try `kubectl explain RESOURCE_NAME`



Pod

- The **smallest and simplest Kubernetes object**.
- It is the unit of deployment in Kubernetes, which represents a single instance of the application (usually a microservice).
- Is a logical collection of one or more containers, which:
  - **Are scheduled together on the same host**
  - **Share the same network namespace**
  - **Mount the same external storage (Volumes).**
- Pods are ephemeral in nature, and they do not have the capability to self-heal by themselves.
- We use controllers to manage them. Examples of controllers are:
  - Deployments
  - ReplicaSets
  - ReplicationControllers



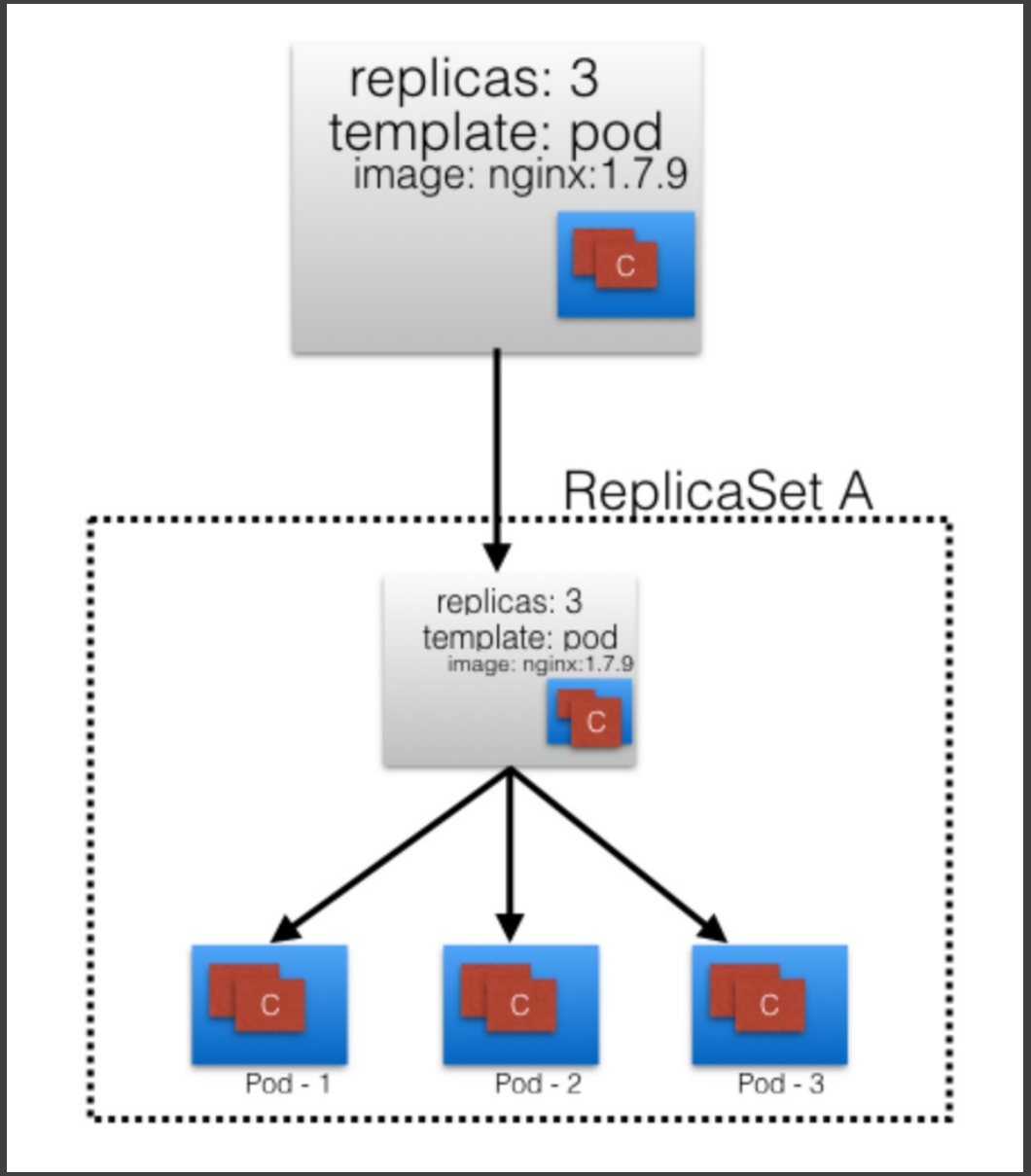
ReplicaSets

- A ReplicaSet Controller is part of the Master Node's Controller Manager. It makes sure the specified number of replicas for a Pod is running at any given point in time looking at the ReplicaSet resources.
  - If there are more Pods than the desired count, the ReplicationController would kill the extra Pods
  - If there are less Pods, then the ReplicationController would create more Pods to match the desired count.
- Generally, we don't deploy a Pod independently, as it would not be able to re-start itself, if something goes wrong. We always use controllers like ReplicaSet to create and manage Pods.

- A ReplicaSet (rs) is the next-generation ReplicationController. ReplicaSets support both equality- and set-based Selectors, whereas ReplicationControllers only support equality-based Selectors. Currently, this is the only difference.

Deployments

- Deployment objects provide declarative updates to Pods and ReplicaSets. The DeploymentController is part of the Master Node's Controller Manager, and it makes sure that the current state always matches the desired state.
- On top of ReplicaSets, Deployments provide features like Deployment recording, with which, if something goes wrong, we can rollback to a previously known state.



Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, May 23rd 2023

Labels and Selectors

- Labels are key-value pairs that can be attached to any Kubernetes objects (e.g. Pods)
  - Labels are used to organize and group a subset of objects
  - Labels do not provide uniqueness to objects

- Examples: `app=webserver, app=database, env=dev, env=prod, env=qa`

- With Selectors, we can select a subset of objects.
  - **Equality-Based Selectors:** Filters based on label keys and values
    - Operators: `=, ==, !=` eg: `env==dev`
  - **Set-Based Selectors:** Filters based on a set of values
    - Operators: `in, noin, exists` eg: `env in (dev,qa)`

ConfigMaps & Secrets (1/2)

- While deploying an application, we may need to pass runtime parameters like configuration details, passwords, etc.

- We need to decouple configuration details from container image allowing to pass them as key-value pairs to k8s objects or system components.

- We also want to pass them as reference, controlling the usage and hiding the content

Do you remember the 3° rule of the 12° Factor rules?

ConfigMaps & Secrets (2/2)

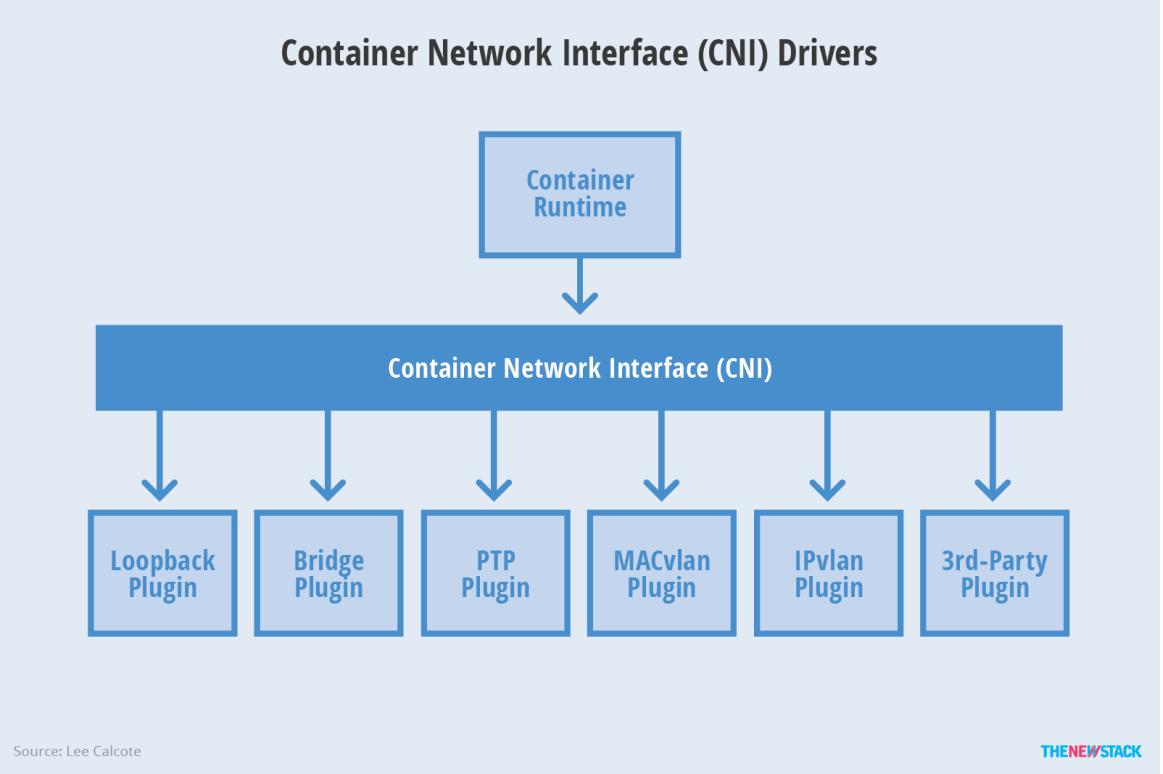
- In such cases, we can use the **ConfigMap** API resource.
- Similarly, when we want to pass sensitive information, we can use the **Secret** API resource.
- Both ConfigMaps and Secrtes can be created and retrieved in various ways
  - Created from literal values, from files and from directory of files…
  - Used via `ENV_VARS`, Volumes…

Networking Requirements

- To have a fully functional cluster, during installation phase, we need to make sure of the following requirements:
  - A unique IP address is assigned to each Pod
  - Containers in a Pod can communicate to each other
  - The Pod is able to communicate with other Pods in the cluster
  - If configured, the application deployed inside a Pod is accessible from the external world.
- All of the above are networking challenges which must be addressed.

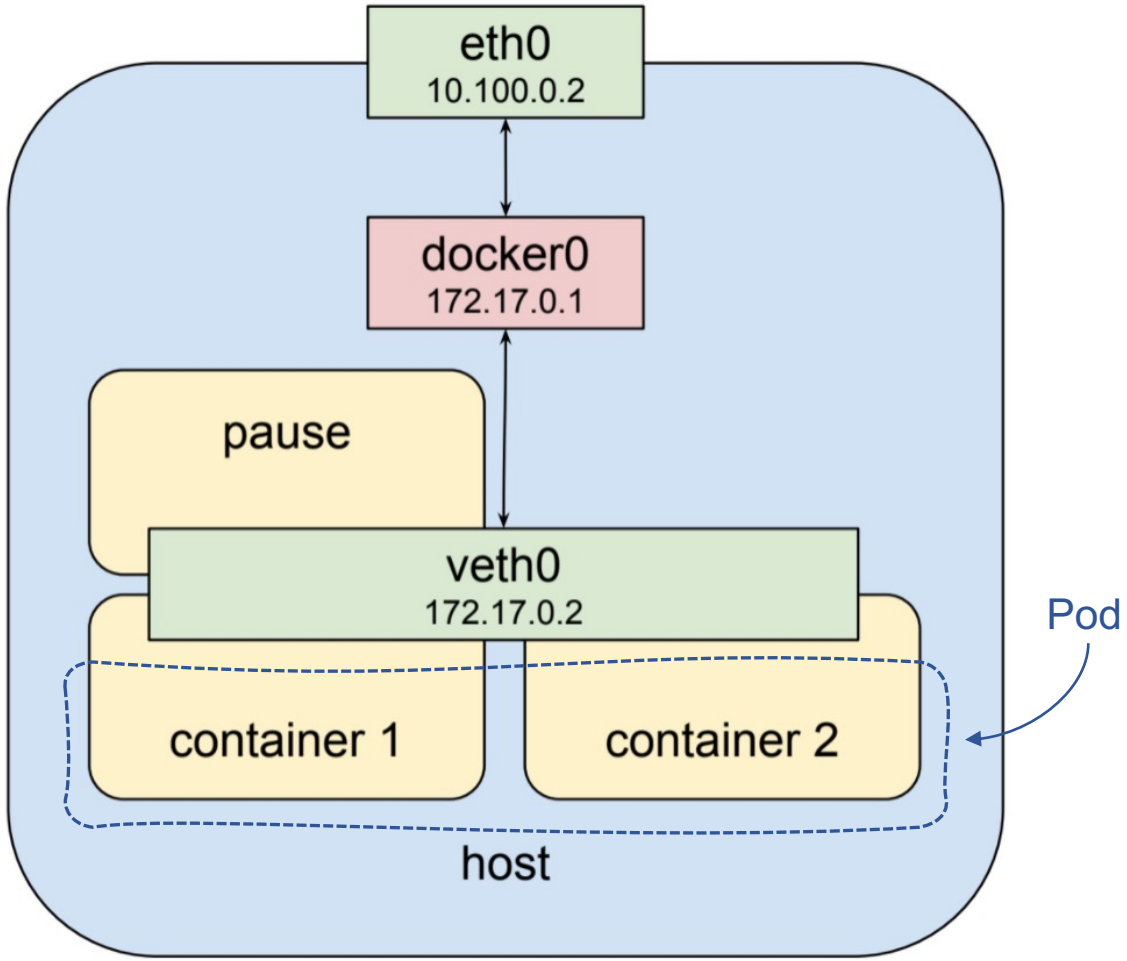
Container Network Interface (CNI)

- In Kubernetes, each Pod gets a unique IP address.
- For container networking, there are two primary specifications:
  - Container Network Model (CNM), proposed by Docker
  - Container Network Interface (CNI), proposed by CoreOS (used by k8s)
- The Container Runtime offloads the IP assignment to CNI, which connects to the underlying configured plugin, like Bridge or MACvlan, to get the IP address. Once the IP address is given by the respective plugin, CNI forwards it back to the requested Container Runtime.
- More info on TKNG (*The Kubernetes Networking Guide*)



Container-to-Container communication inside a Pod

- With the help of the underlying Host OS, all of the Container Runtimes generally create an isolated network entity for each container that they starts.
- On Linux, that entity is referred to as a Network Namespace. These Network Namespaces can be shared across containers, or with the Host Operating System.
- Inside a Pod, containers share the same Network Namespaces, so that they can reach each other via localhost.
- `pause` is a special container that provides a virtual network interface for the other containers to communicate.



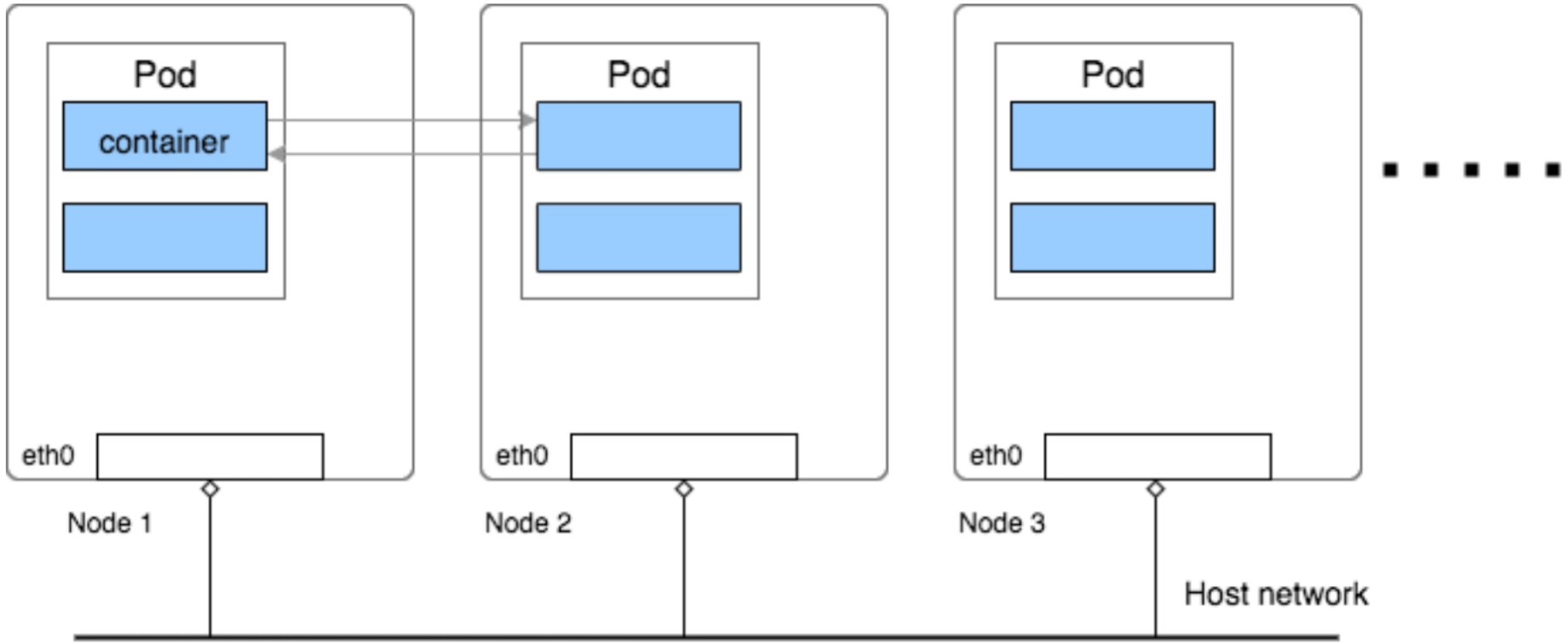
Pod-to-Pod Communication Across Nodes

- In a clustered environment, the Pods can be scheduled on any node.

- We need to make sure that the Pods can communicate across the nodes, and all the nodes should be able to reach any Pod.

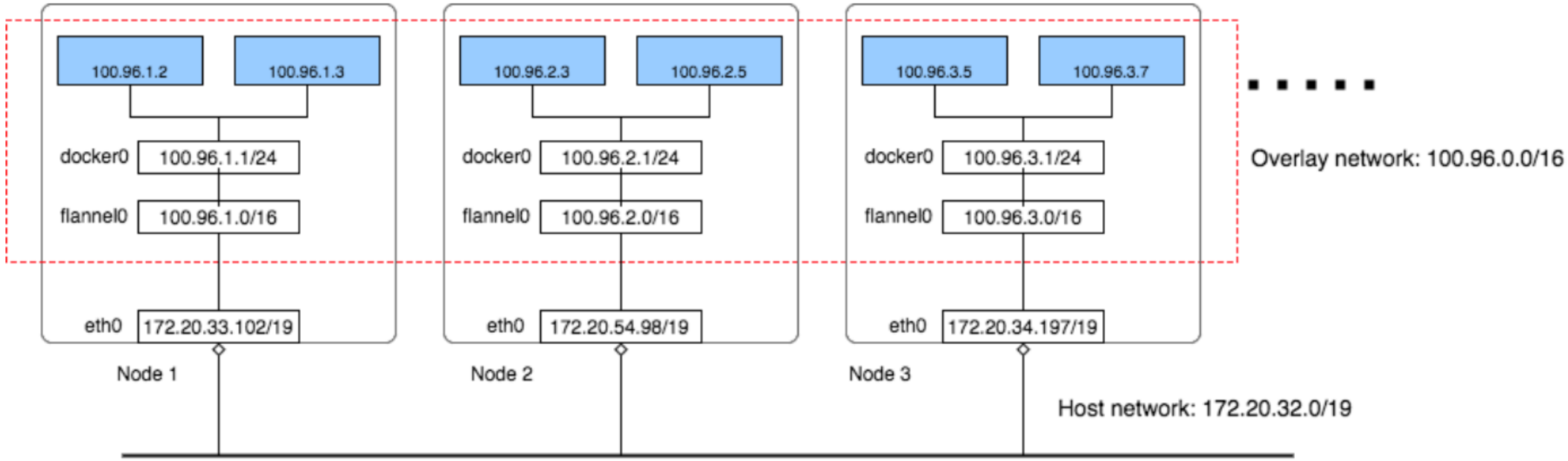
- Kubernetes also puts a condition that there shouldn't be any Network Address Translation (NAT) while doing the Pod-to-Pod communication across Hosts. We can achieve this via:
  - Routable Pods and nodes, using the underlying physical infrastructure, like Google Container Engine
  - Using Software Defined Networking, like Flannel, Weave, Calico, Kindnet, etc.

k8s Networking



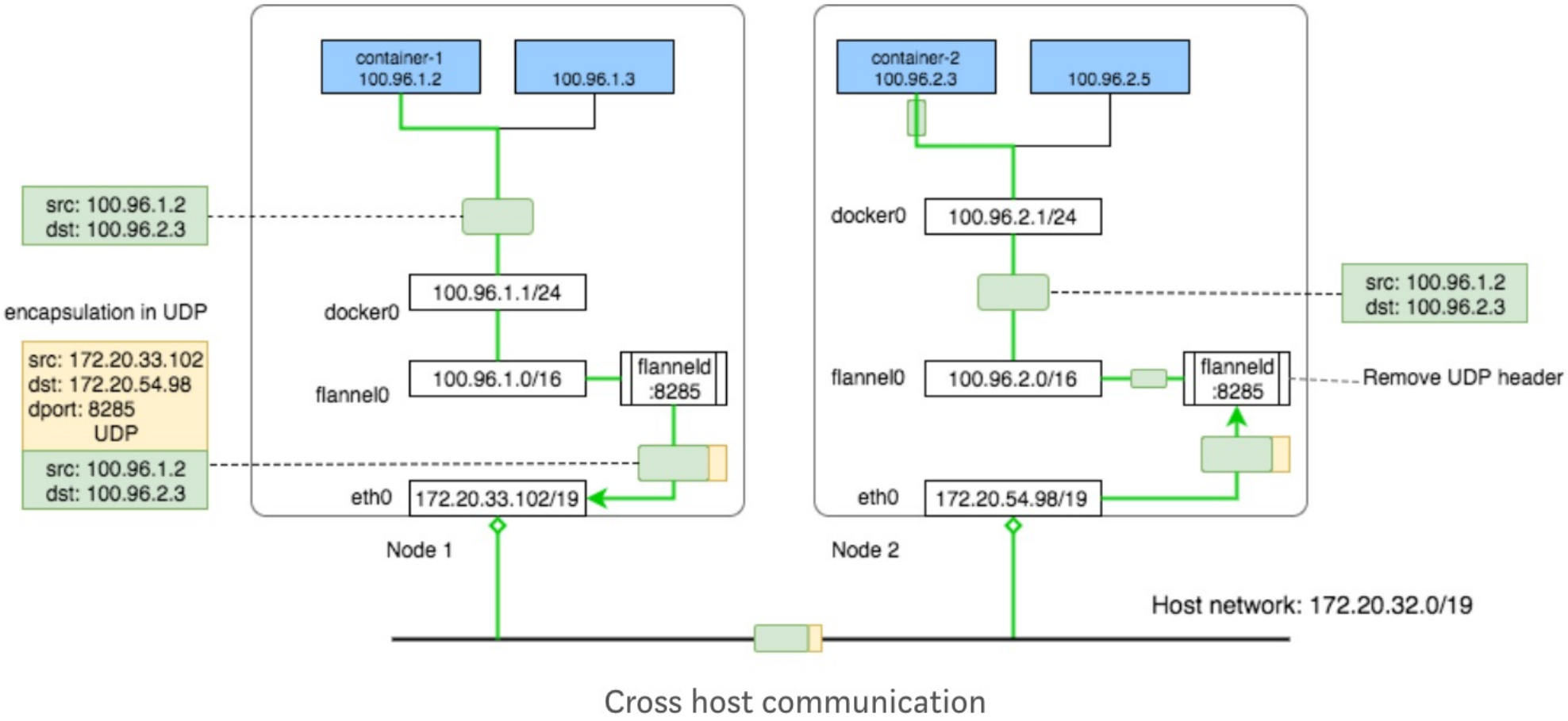
Pods in Kubernetes

Overlay Network



Flannel overlay network

Network Plugin - Flannel



Flannel uses UDP encapsulation in order to encapsulate generic packets into UDP packets.
Many implementations: IPSec, VXLan, others

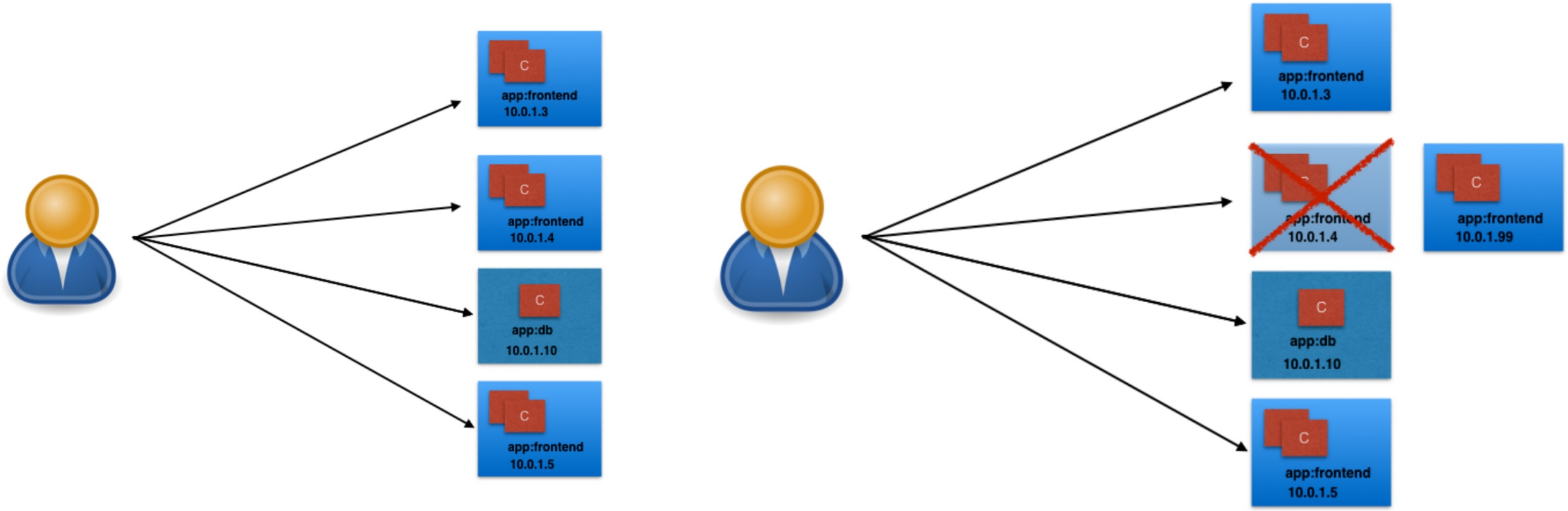
External World-to-Pod Communication

- To access our applications from outside the cluster we expose our services to the external world using the **Service** resource and **kube-proxy**.

- **Service** [ref] is an high-level abstraction, which logically groups Pods and add policies to access them. This grouping is achieved via Labels and Selectors

Services

- Scenario in which a user/client is connected to a Pod using its IP address.
- Unexpectedly, the Pod to which the user/client is connected dies, and a new Pod is created by the controller.
- The new Pod will have a new IP address, which will not be known automatically to the user/client of the earlier Pod.

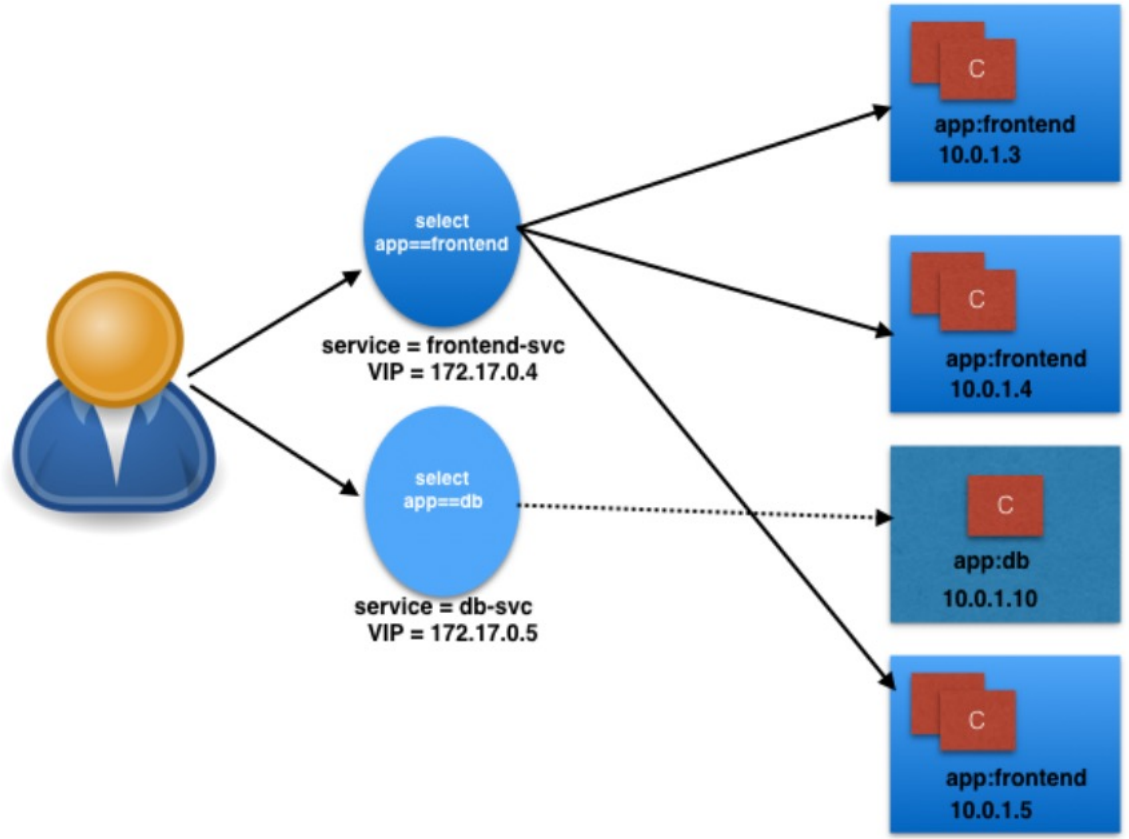


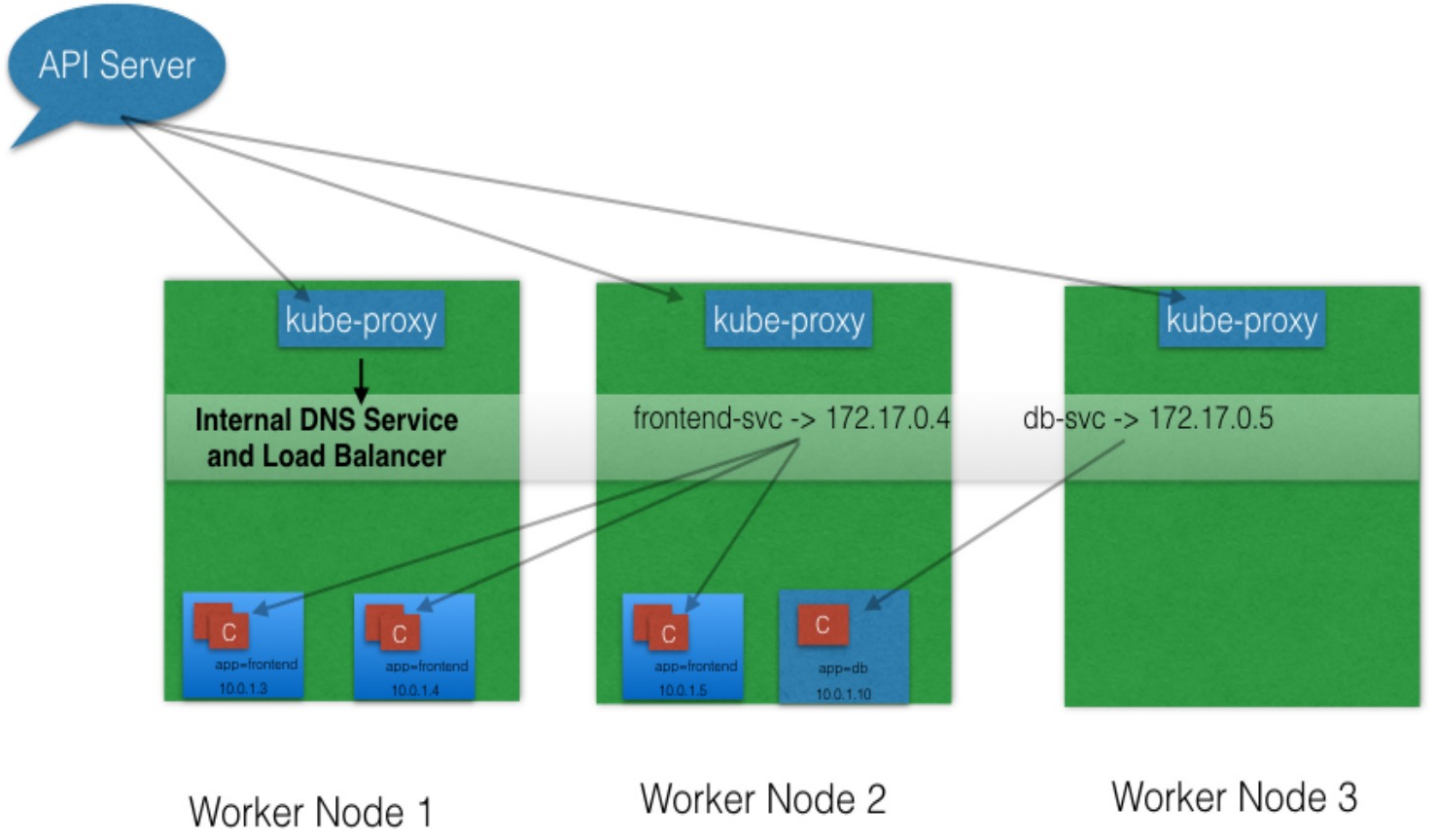
Services 1/3

- Kubernetes provides:
  - A higher-level abstraction called Service, which logically groups Pods
  - A policy to access them
- This grouping is achieved via Labels and Selectors.

- Using Selectors (**app==frontend** and **app==db**), we can group them into two logical groups: one with 3 Pods, and one with just one Pod.

- We can assign a name to the logical grouping, referred to as a **service name**.
- In our example, we have created two Services: **frontend-svc** and **db-svc**, and they have the **app==frontend** and the **app==db** Selectors, respectively.





Services 3/3

- All of the Worker Nodes run a daemon called kube-proxy, which watches the API Server on the Master Node for the addition and removal of Services and Endpoints.
- For each new Service, on each node, **kube-proxy** configures the IPTables rules to capture the traffic for its ClusterIP and forwards it to one of the endpoints.
- When the Service is removed, **kube-proxy** removes the IPtables rules on all nodes as well.

Type of Services

- **ClusterIP**: Exposes the service on a cluster-internal IP. Choosing this value makes the service only reachable from within the cluster. This is the default ServiceType.

- **NodePort**: Exposes the service on each Node's IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service will route, is automatically created. You'll be able to contact the NodePort service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

- **LoadBalancer**: Exposes the service externally using a cloud provider's load balancer. NodePort and ClusterIP services, to which the external load balancer will route, are automatically created.

- **ExternalName**: Maps the service to the contents of the externalName field (e.g. `foo.bar.example.com`), by returning a CNAME record with its value. No proxying of any kind is set up.

Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, June 06th 2023

Storage in k8s [ref]

- Container are ephemeral in nature, so if it crashes all data stored in it is deleted. `kubelet` restarts it but without any of the old data inside.
- To overcome this k8s uses Volumes.
- A Volume is essentially a directory backed by a storage medium. The storage medium and its content are determined by the Volume Type.
- A volume is shared among the containers of the same Pod
- Depending on its type, a volume:
  - has the same lifetime as the Pod - **Ephemeral Volumes**
  - it outlives the containers of the Pod - **Persistent Volumes**

Type of Volumes (1/2)

- Volume Type decides the properties of the directory, types are:
  - **emptyDir**: An empty Volume is created for the Pod as soon as it is scheduled on the Worker Node. The Volume's life is tightly coupled with the Pod. If the Pod dies, the content of `emptyDir` is deleted forever.
  - **hostPath**: With the `hostPath` Volume Type, we can share a directory from the host to the Pod. If the Pod dies, the content of the Volume is still available on the host.
  - **gcePersistentDisk**: With the gcePersistentDisk Volume Type, we can mount a Google Compute Engine (GCE) persistent disk into a Pod.

Type of Volumes (2/2)

- Volume Type decides the properties of the directory, types are:
  - **awsElasticBlockStore**: With the awsElasticBlockStore Volume Type, we can mount an AWS EBS Volume into a Pod.
  - **nfs**: With nfs, we can mount an NFS share into a Pod.
  - **iscsi**: With iscsi, we can mount an iSCSI share into a Pod.
  - **secret**: With the secret Volume Type, we can pass sensitive information, such as passwords, to Pods
  - ….

Volume management and API

- Kubernetes resolves the problem of the storage with the Persistent Volume Subsystem

- It provides APIs for users and administrators to manage and consume storage:

  – To manage the Volume → PersistentVolume (PV) API resource type

  – To consume the Volume → PersistentVolumeClaim (PVC) API resource type

PersistentVolume and StorageClass

- A Persistent Volume is (usually) a network attached storage in the cluster

- A persistent Volume can be:
  - **Statically** provisioned by the administrator.
  - **Dynamically** provisioned based on the `StorageClass` resource.

- A StorageClass contains pre-defined provisioners and parameters to create a Persistent Volume.

PersistentVolumeClaim

- A PersistentVolumeClaim (PVC) is a request for storage by a user. It follows the below lifecycle:
  1. Users request for Persistent Volume (PV) resources based on size, access modes, etc. via a PVC
  2. Once a suitable Persistent Volume is found, it is bound to a Persistent Volume Claim
  3. After a successful bind, the PersistentVolumeClaim resource can be used in a Pod.
  4. Once a user finishes its work, the attached Persistent Volumes can be released. The underlying Persistent Volumes can then be reclaimed and recycled for future usage.

Namespaces

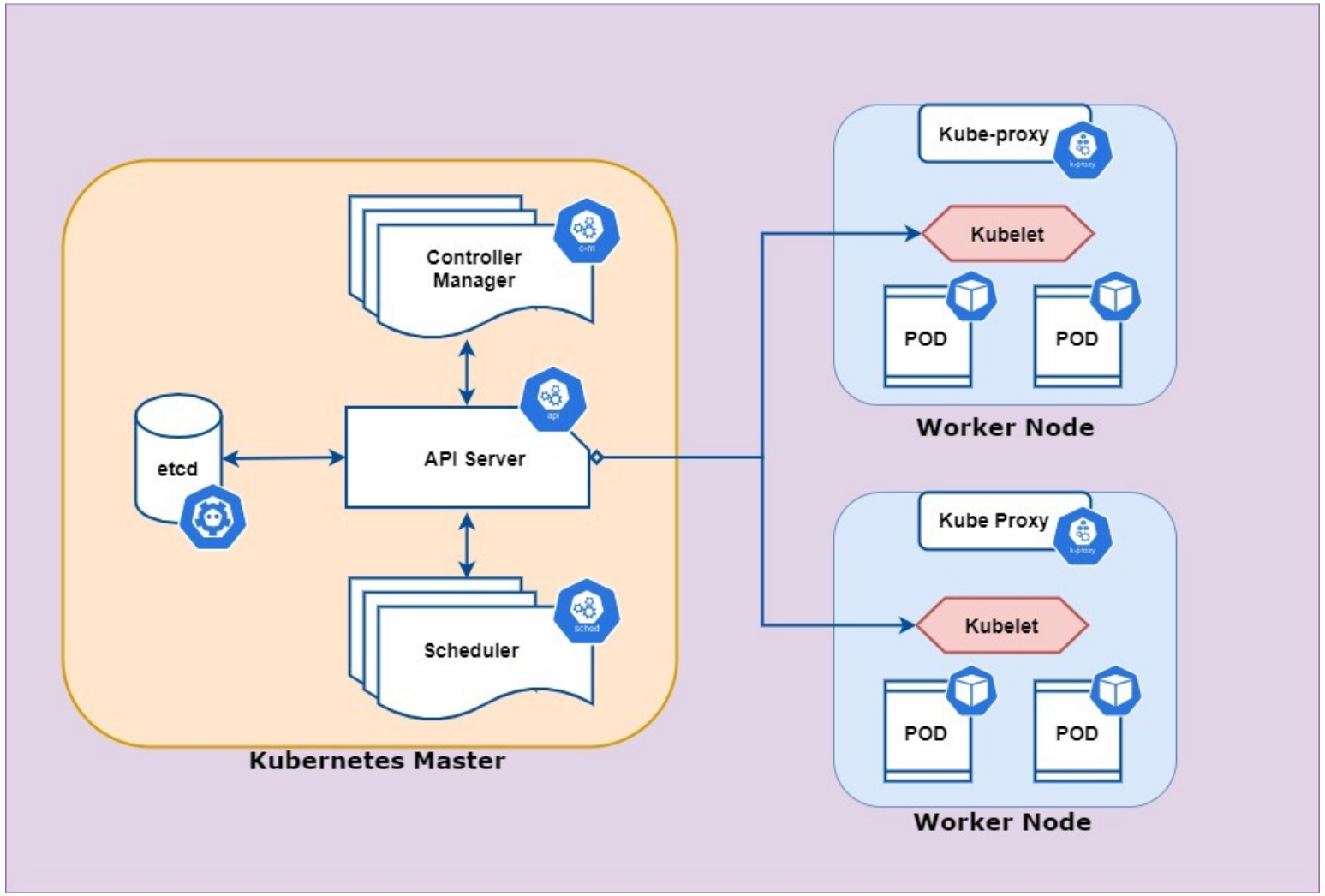
- If we have numerous users whom we would like to organize into teams/projects, we can partition the Kubernetes cluster into «sub-clusters» using Namespaces

- The names of the resources/objects created inside a Namespace are unique, but not across Namespaces

- Generally, Kubernetes creates two default namespaces:
  - **kube-system:** contains objects created by k8s system
  - **default:** contains objects which belong to any other user

- Using Resource Quotas, we can divide the cluster resources within Namespaces.

k8s scheduler



- Control plane process
- Works with controller-manager through the API-server
- Run in master node(s)
- Run in `kube-system` namespace
- Assigns Pods to Nodes
- Uses constraints and available resources
- Algorithms can be extended/customised

Pod Placement

- The `kube-scheduler` selects a node for the pod in a 2-step operation:
  - Filtering
  - Scoring

- Filtering uses some scheduler constraints
  - NodeSelector
  - Affinity and anti-Affinity

- Scoring sorts the remaining nodes to choose the most suitable Pod placement, based on active scoring rules

```
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: lb-example2
    spec:
      nodeSelector:
        region: "EDGE"
      containers:
      - name: worker
        image: python:3.6-alpine
        command:
        - "/bin/sh"
```

Other k8s Objects & Features

FONDAZIONE BRUNO KESSLER

- **Deployment Advanced Features:** Autoscaling, Proportional Scaling, Pausing and Resuming
- **Ingress:** Collection of rules that allow inbound connections to reach the cluster Services.
- **Jobs:** Create one or more Pod to perform a given task. It makes sure the task is completed , then terminates the Pods. Cron Job is a Job on a time-based schedule
- **StatefulSets:** For application that require a unique identity, like name, net id, strict ordering, eg: mysql or etcd cluster (*it was called PetSet < 1.5*)
- **DaemonSets:** Special Pod that run on all nodes and is started/deleted automatically when node is added/removed
- **Quota Management:** Limit resource consumption per namespace: Compute, Storage, Object Count
- **CRD:** Create our own API objects and Controller that manages them
- **RBAC:** Authorization mechanism for managing permissions around Kubernetes resources
- **Kubernetes Federation:** Manage multiple Kubernetes clusters from a single control plane

… and many more…

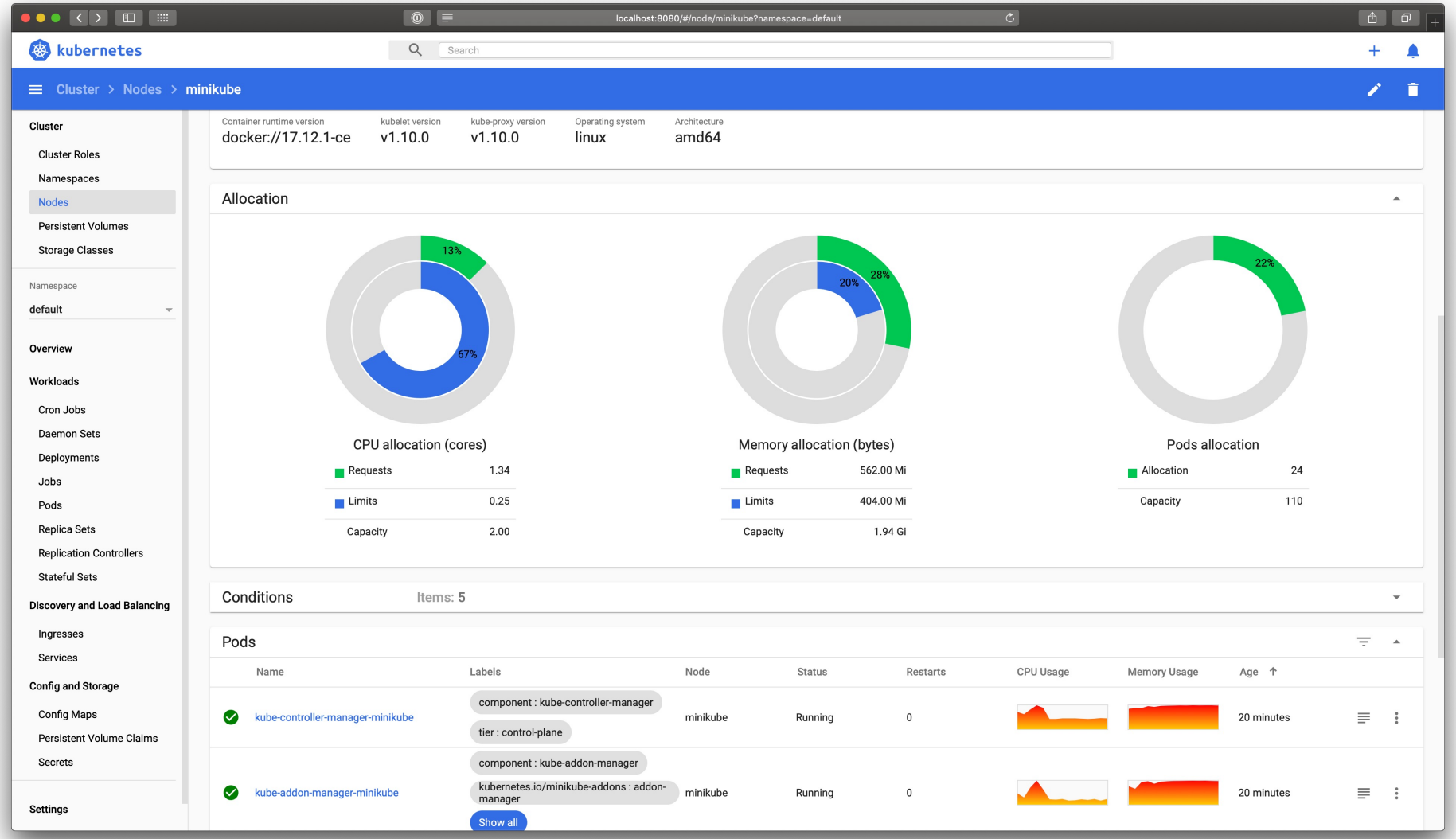
Fog and Cloud Computing
Lab

Daniele Santoro (*dsantoro@fbk.eu*) - Expert Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, June 06th 2023

Kubernetes Dashboard



See the official repository for more information → https://github.com/kubernetes/dashboard

Helm

- Resources in k8s are described as YAML manifests
  - We can bundle all those manifests after templatizing them into a well-defined format, along with other metadata
  - Such a bundle is referred to as Chart
  - These Charts can then be served via repositories, such as those that we have for `rpm` and `deb` packages
- Helm is a package manager (analogous to `yum` and `apt`) for Kubernetes, which can help to define, install, update, delete those Charts (applications) in the Kubernetes cluster.
- Helm is a graduated project in the CNCF
- Introduction to Helm video [ref]

Ingress (1/3)

- Ingress is another method we can use to access our applications from the external world.

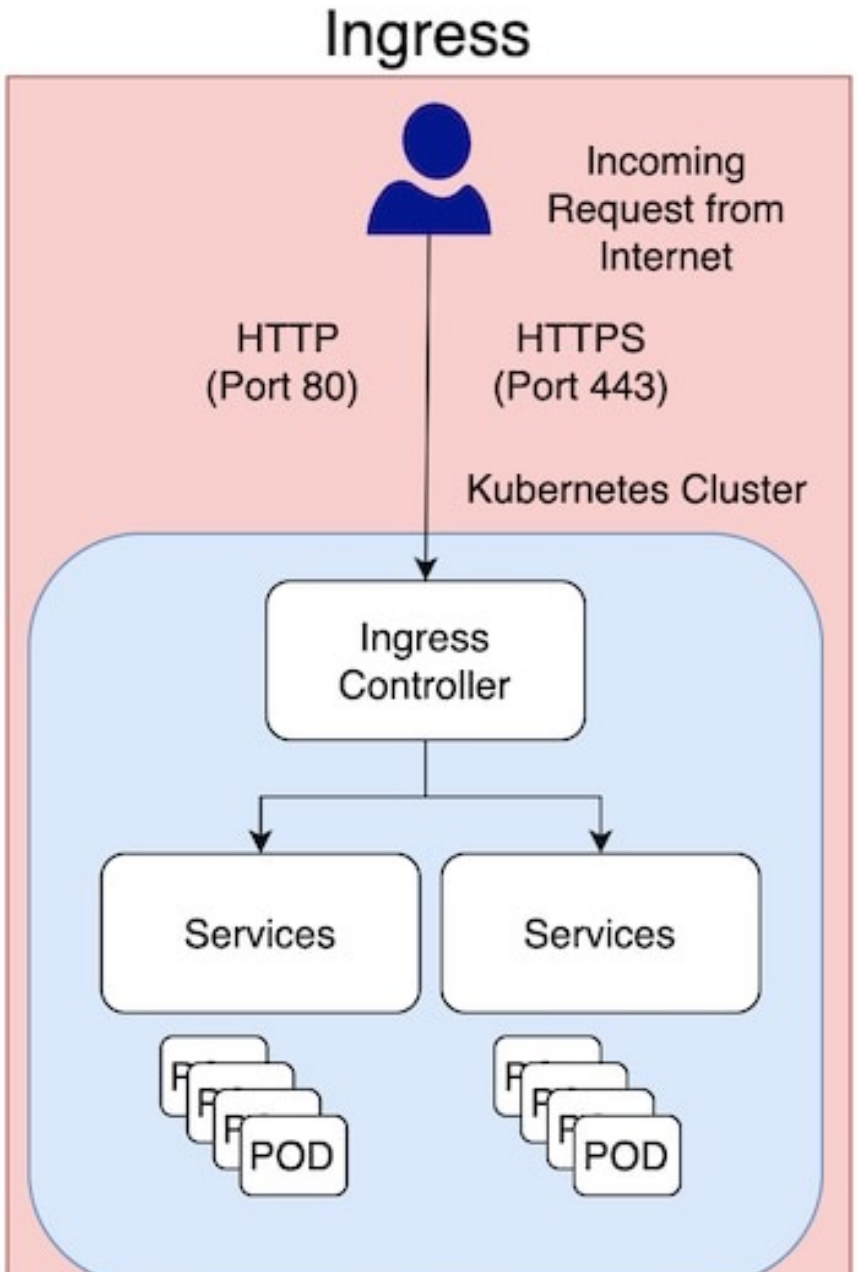
- With Services, routing rules are attached to a given Service.
  - They exist for as long as the Service exists.

- With Ingress we can somehow decouple the routing rules from the application
  - When we update our application we do not worry about its external access rules

Ingress (2/3)

- An Ingress is a collection of rules that allow inbound connections to reach the cluster Services.

- To allow the inbound connection to reach the cluster Services, Ingress configures a **Layer 7 HTTP load balancer** for Services and provides the following:
  - TLS (Transport Layer Security)
  - Name-based virtual hosting
  - Path-based routing
  - Custom rules



Ingress (3/3)

- With Ingress, users don't connect directly to a Service. Users reach the Ingress endpoint (Ingress Controller), and, from there, the request is forwarded to the respective Service.

- Example of Ingress rules are:
  - Name-Based Virtual Hosting Ingress rule (based on the FQDN)
  - Fan Out Ingress rule (based on path)

Ingress Controller

- All of the magic is done using the Ingress Controller
- Once the Ingress Controller is deployed, we can create an Ingress resource using the kubectl create command
- An Ingress Controller is an **application** which
  - Watches the Master Node's API Server for changes in the Ingress resources
  - Updates the Layer 7 load balancer accordingly
- Kubernetes has different Ingress Controllers, and, if needed, we can also build our own
- GKE L7 Load Balancer and Nginx Ingress Controller are examples of Ingress Controllers