Fog and Cloud Computing
Lab

Daniele Santoro *(dsantoro@fbk.eu)* - Expert Research Engineers
Silvio Cretti *(scretti@fbk.eu)* – *Senior* Research Engineers

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

Trento, April 8th 2022

Lab Resources

- Shared Etherpad: https://annuel2.framapad.org/p/6s5u416vo7-9t4b
- White Board: https://tinyurl.com/2p8j7yra
- Interaction:
  - Etherpad
    - *Exercises check, Share Troubleshooting, Questions and Logs*
  - Zoom Chat (for those remotely connected)
    - *Discuss with your colleagues during exercises or directly/privately with me*
  - Rise your Hand (also via Zoom)
    - *If you need my attention or want to speak, don't be shy !!!*
  - Course Forum: https://tinyurl.com/27vmd9pj
    - *Questions and answers could be useful to others, be collaborative*

Lab Resources

- Slides
  - Uploaded before any lesson in Moodle
- Repositories of exercises
  - https://gitlab.fbk.eu/dsantoro/fcc-lab-2022
- Lab Virtual Machine:
  - **Lab VM on Azure (reference for exercises)**
  - Vagrant and VirtualBox on your laptop (possible choice)
    - https://www.virtualbox.org/, https://www.vagrantup.com/ and https://gitlab.fbk.eu/dsantoro/fcc-lab-2022

Today Lesson

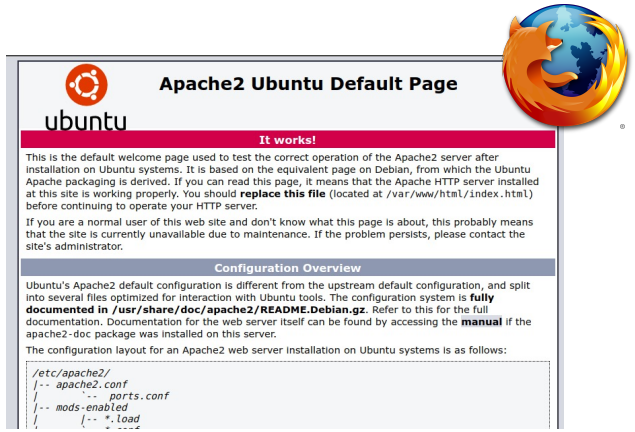
- Recap of Lab Nested VMs and Containers
- Recap of Docker basis
- Cloud Native approach
- Twelve Factor approach
- Docker
  - Images
  - Networking
  - Volumes
  - Multi-Container application (optional)
- IaaS and OpenStack intro ?

Nested VMs and Containers - Networking

SSH Socks tunnel

```
ssh \
-D 4444 \
-p Azure_PORT disi@Azure_URL
```

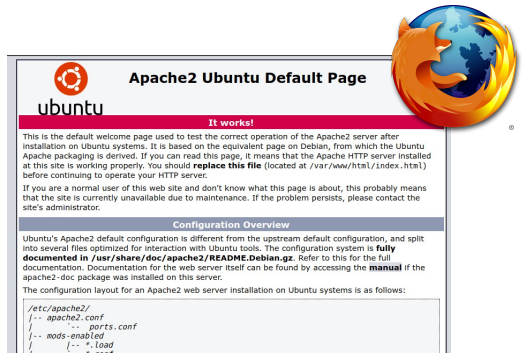
Lab VM



Nested VMs and Containers - Networking

SSH Port Forward

```
ssh -L 4444:localhost:4444 \
-p Azure_PORT disi@Azure_URL
```



SSH Socks tunnel

```
cd fcc-lab-2022
vagrant ssh -- -D 4444
```

Lab VM (host)

Vagrant VM (guest)
docker-host

docker

Nested VMs and Containers - Files

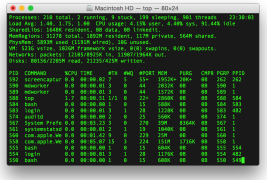
```
cd /home/disi
git clone https://gitlab.fbk.eu/dsantoro/fcc-lab-
cd fcc-lab-2022
```

```
pwd → /home/disi/fcc-lab-2022
vagrant up
vagrant ssh
```

Microsoft Azure

Lab VM (host)

git



Vagrant VM (guest)
docker-host

```
cd /vagrant
ls
pwd → /vagrant
```

docker

Shared via Synced Folder

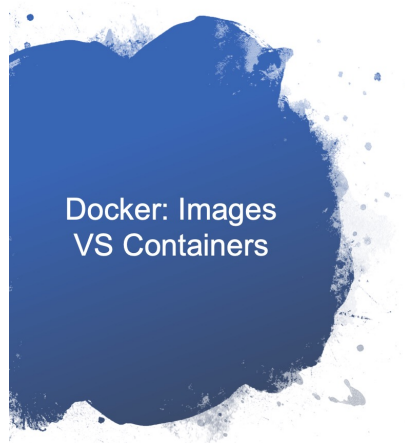
Quick Docker Recap

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/)

Docker: Images VS Containers

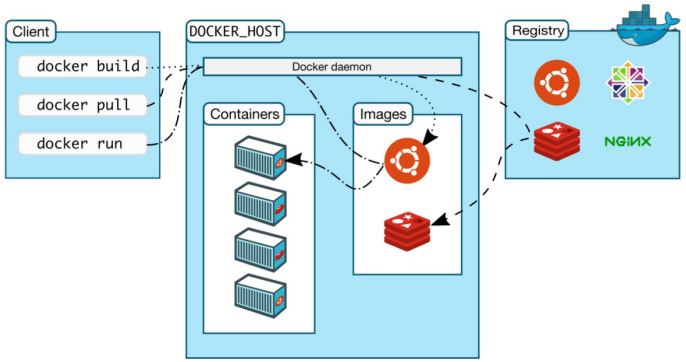
- **Docker Images [ref]**:
  - A read-only template with instructions for creating a Docker container.
    - Often, an image is *based on* another image, with some additional customization. *For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and 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.



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.

21

Docker Host Overview



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.

20

Container Lifecycle

- Create - **docker create** <image>
- Start - **docker start** <container id>
- Stop - **docker kill** , **docker stop** <container id>
- Restart - **docker restart** <container id>
- Remove - **docker rm** <container id>

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

Exercise 12 – Build a Docker image using a Dockerfile

- **Time:** ~10 minutes
  - 3 minutes: *Try by yourself*
  - 5 minutes: *Check, Verify, Ask*

- **Description:** After completing the exercise 3 try to build the very same image but using a Dockerfile instead of creating it from a running container. Understand how and why your image is different from the initial one. Give it a name and a tag and optionally upload it on the public Docker registry. Finally start a container based on the custom image.

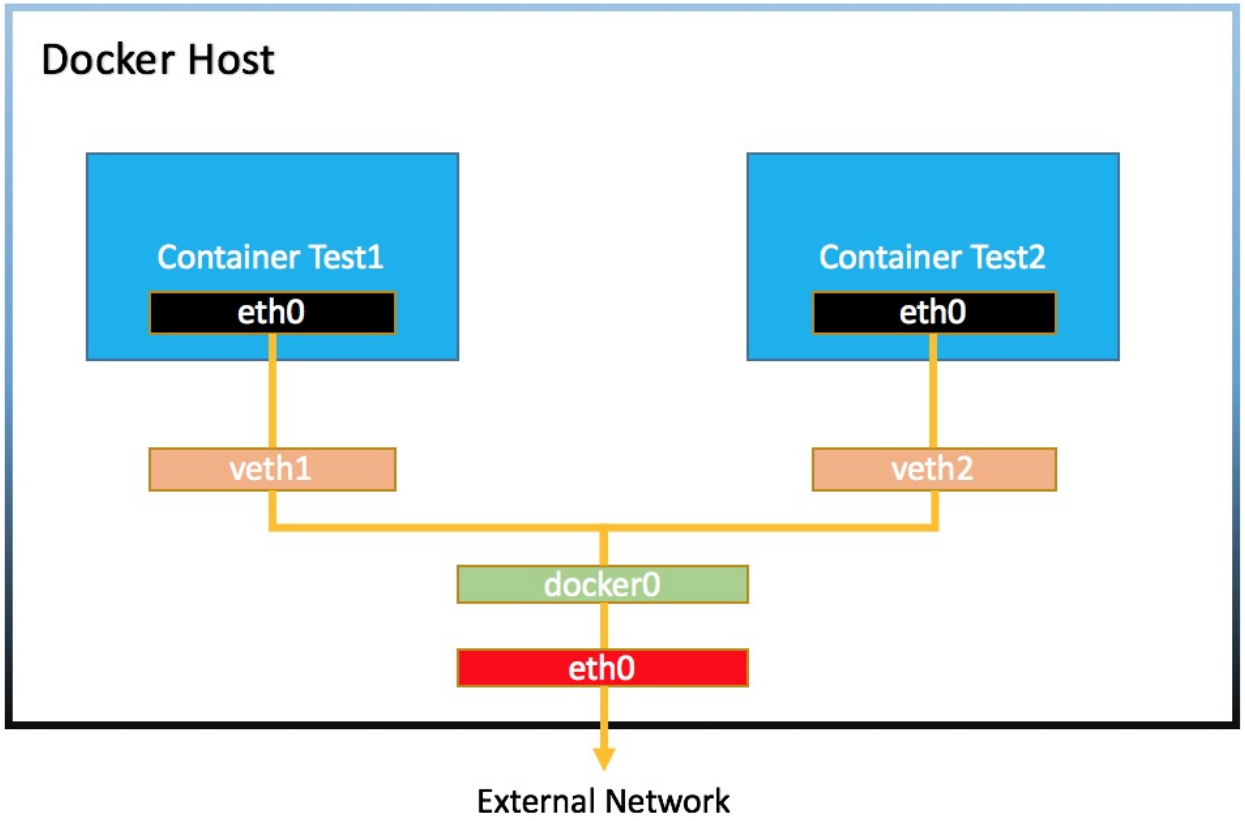
- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e12

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



Exercise 13 – Make two containers talking each others

- **Time:** ~5 minutes
  - 2 minutes: *Try by yourself*
  - 3 minutes: *Check, Verify, Ask*

- **Description:** Create a custom bridged network. Create two different container (a client and a server) and attach them to the same network. Finally use Docker internal DNS to communicate from one to the other.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e13

Interconnecting two (or more) containers

1. Deprecated:

   `docker run --link <container_name>:<alias>`

   Note: Use `docker network connect` instead

2. Use Bridge Network

   Connect two or more containers to the same network (default `docker0` or user defined)

3. Use Overlay Network

   Connect two or more containers across different hosts

Exercise 14 – Run a Service with Docker

- **Time:** ~10 minutes
  - 4 minutes: *Try by yourself*
  - 6 minutes: *Check, Verify, Ask*

- **Description:** Using the `nginx` container image at https://hub.docker.com/_/nginx, setup a simple webserver. Expose its port on the Docker host and connect to it.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e14

Exercise 15 – Run a custom Service with Docker

- **Time:** ~10 minutes
  - 4 minutes: *Try by yourself*
  - 6 minutes: *Check, Verify, Ask*

- **Description:** Using the `nginx` container image at https://hub.docker.com/_/nginx, setup a simple webserver. Replace `nginx` default page with a custom content. Expose its internal port on the port `8080` of the Docker host and connect to it. Finally change the custom content exposed by the webserver.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e15

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

Bind Mount and Volumes

- Mounting a host directory into a container

```
$ docker run -v <host path>:<container path> <image>
```

- Creating a volume and sharing it

```
$ docker volume create my_vol
$ docker run -v my_vol:/data <image>
```

Exercise 16 – Run a custom Docker Service with persistency

- **Time:** ~7 minutes
  - 3 minutes: *Try by yourself*
  - 4 minutes: *Check, Verify, Ask*

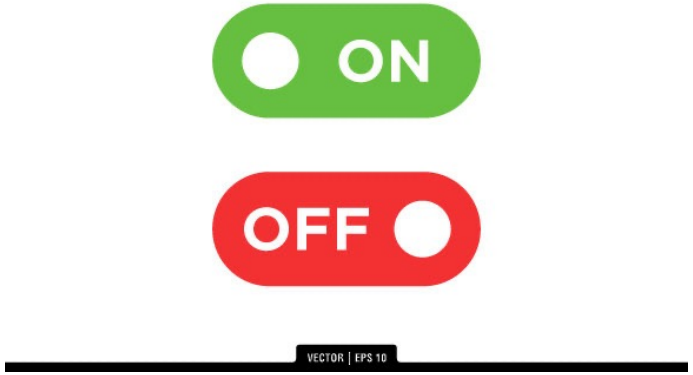
- **Description:** Adapt last exercise using a Bind mount or a Docker Volume to add persistency.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e16

Optional – Deploy a Multi-Container Application

- Let's try to set a web site powered by wordpress:
- Run a mysql container setting the root passwd (-e MYSQL_ROOT_PASSWORD= any_passwd)
- Run a wordpress container connecting it with the mysql container and mapping port 80 of the wordpress container on port 4567 of the host
- With a web browser navigate to the wordpress site and verify of everything works well.
- Note: *There is a tool dedicated to the deployment of multi container applications that is called* ***Docker Compose****.*

Important Reminder



REMEMBER TO TURN OFF
YOUR LAB VM ON AZURE

Saves

Cloud Computing

- Cloud computing is a model for enabling ubiquitous, convenient, on- demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
- This cloud model is com- posed of five essential characteristics, three service models, and four deployment models.

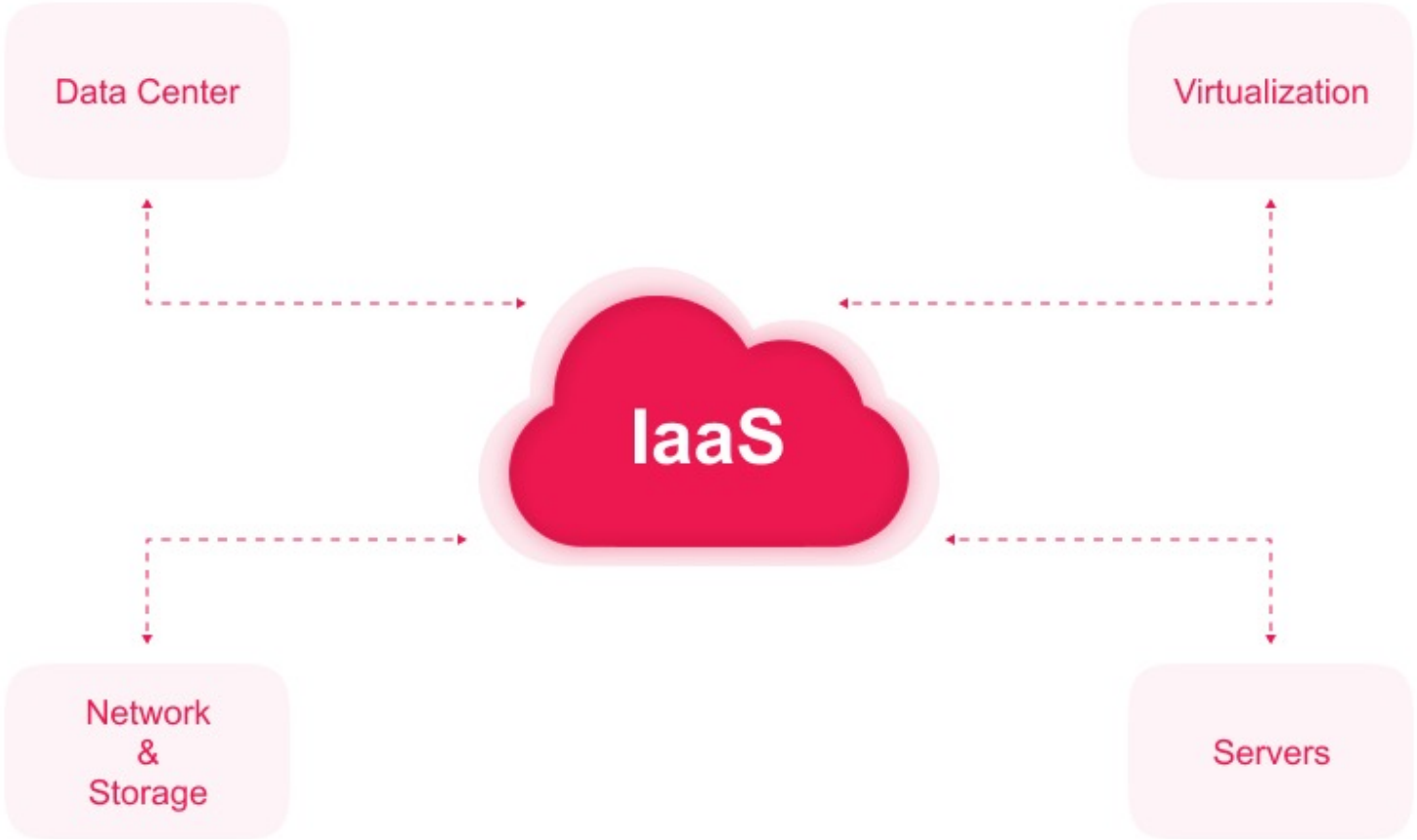
- http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Essential Characteristics
– On-demand self-service – Broad network access –
Resource pooling
– Rapid elasticity
– Measured service

Service Models
– Software as a Serice (SaaS)
– Platform as a Service (PaaS)
– Infrastructure as a Service (IaaS)

Deployment Models:
– Private Cloud
– Community Cloud – Public Cloud
– Hybrid Cloud

IaaS



Open Stack (https://openstack.org/)

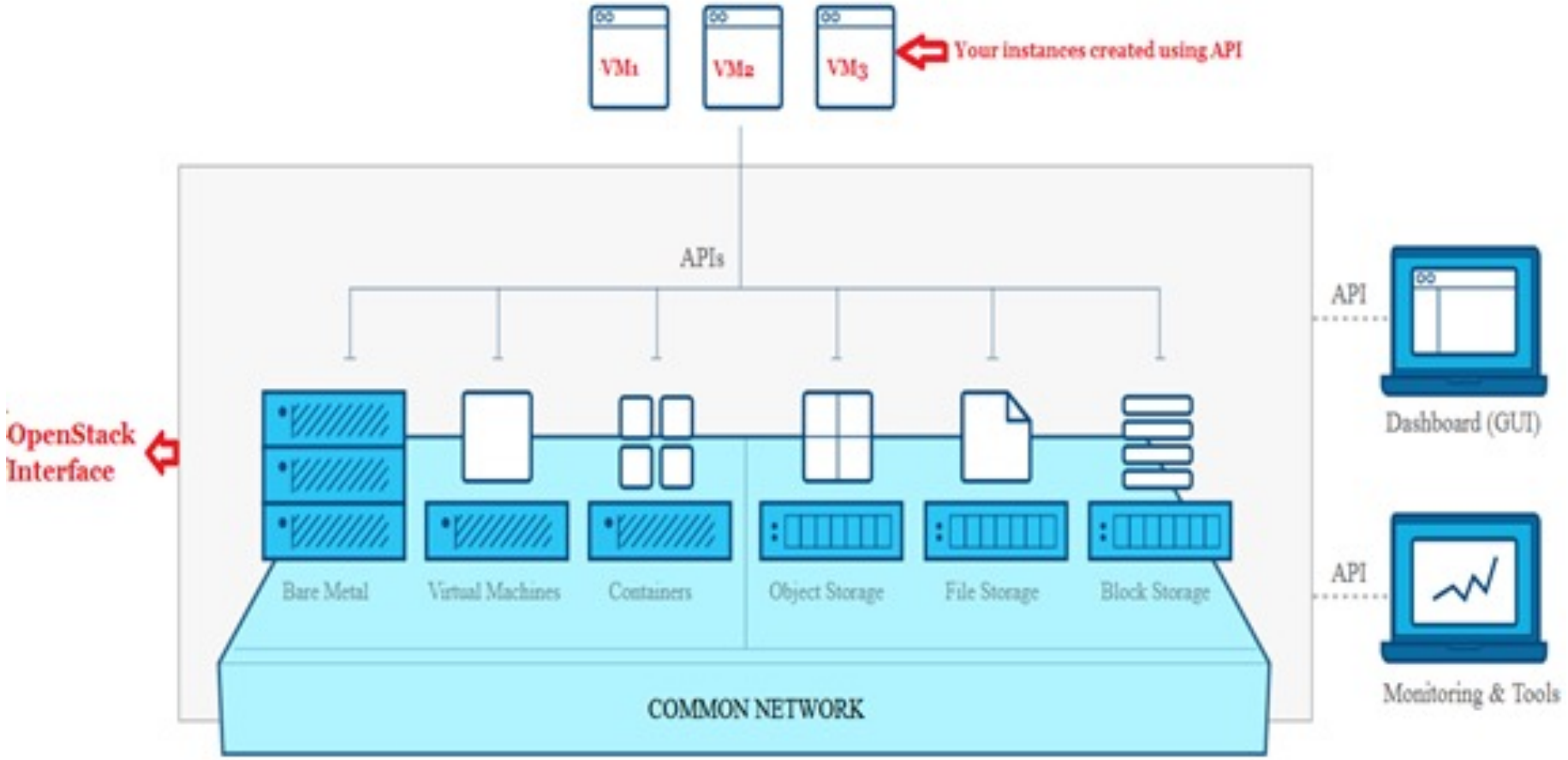
- Open source software for creating private and public clouds.
- OpenStack software controls large pools of compute, storage, and network-ing resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making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

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

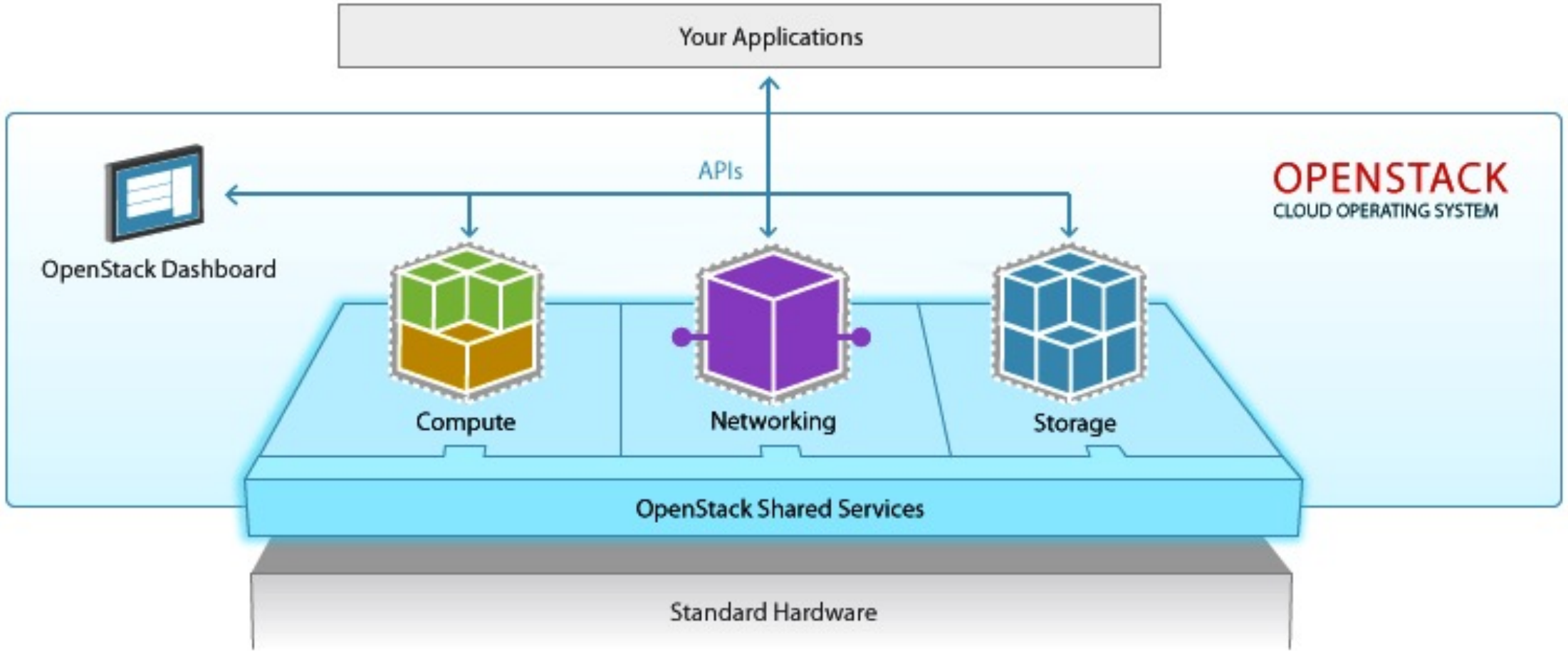
- Four “open”s
  - Open Source, Open Design, Open Development, Open Community
  - More information at the governance page

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

Open Stack Model, Architecture



Open Stack Model, Architecture



Open Stack Components, Services