Dockerfiles and how to get lost

It’s not that you cannot get assisted by AI...

Just be sure to know the error it can make in advance.

Now let’s build a cool data-science container image for humans

Dockerfiles and how to get lost

#SOSC24

INFN
Istituto Nazionale di Fisica Nucleare

Dockerfile

```
1  FROM quay.io/jupyter/tensorflow-notebook:python-3.11
2
3  USER root
4
5  RUN apt update && apt install -y graphviz
6
7  USER jovyan
8
9  RUN conda install -y -c conda-forge dask distributed
10
11 RUN pip3 install boto3 graphviz black papermill nats-python pillow tqdm mlflow  "
12
13 RUN mkdir $HOME/bin $HOME/data
14
15 RUN curl https://dl.min.io/client/mc/release/linux-amd64/mc \
16   --create-dirs \
17   -o $HOME/bin/mc \
18   && chmod +x $HOME/bin/mc \
19   && export PATH=$PATH:$HOME/bin
20
```

```
20
21 RUN conda install -y -c conda-forge \
22   jupyterlab-git \
23   jupyter-resource-usage \
24   nodejs \
25   dask-labextension \
26   jupyter-server-proxy \
27   jupyter-ai \
28   jupyterlab-s3-browser \
29   langchain-openai \
30   jedi-language-server
31
32 RUN jupyter labextension enable jupyterlab_s3_browser \
33   &&  jupyter labextension enable jupyter_resource_usage
34
35 COPY config.py /etc/jupyter/jupyter_server_config.py
36
37 USER root
38
39 RUN usermod -aG sudo jovyan
40 # New added for disable sudo password
41 RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
42
43 RUN wget https://dl.min.io/server/minio/release/linux-amd64/minio \
44   && chmod +x minio \
45   && mv minio /usr/bin/
46
47 COPY init.sh /opt/init.sh
48
49 RUN chmod +x /opt/init.sh
50
51 USER jovyan
```

Let's take a look at the image you are going to use for the school

INFN
Istituto Nazionale di Fisica Nucleare

Containers Orchestration

Diego Ciangottini
INFN

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Introduction

Hand-on session

today

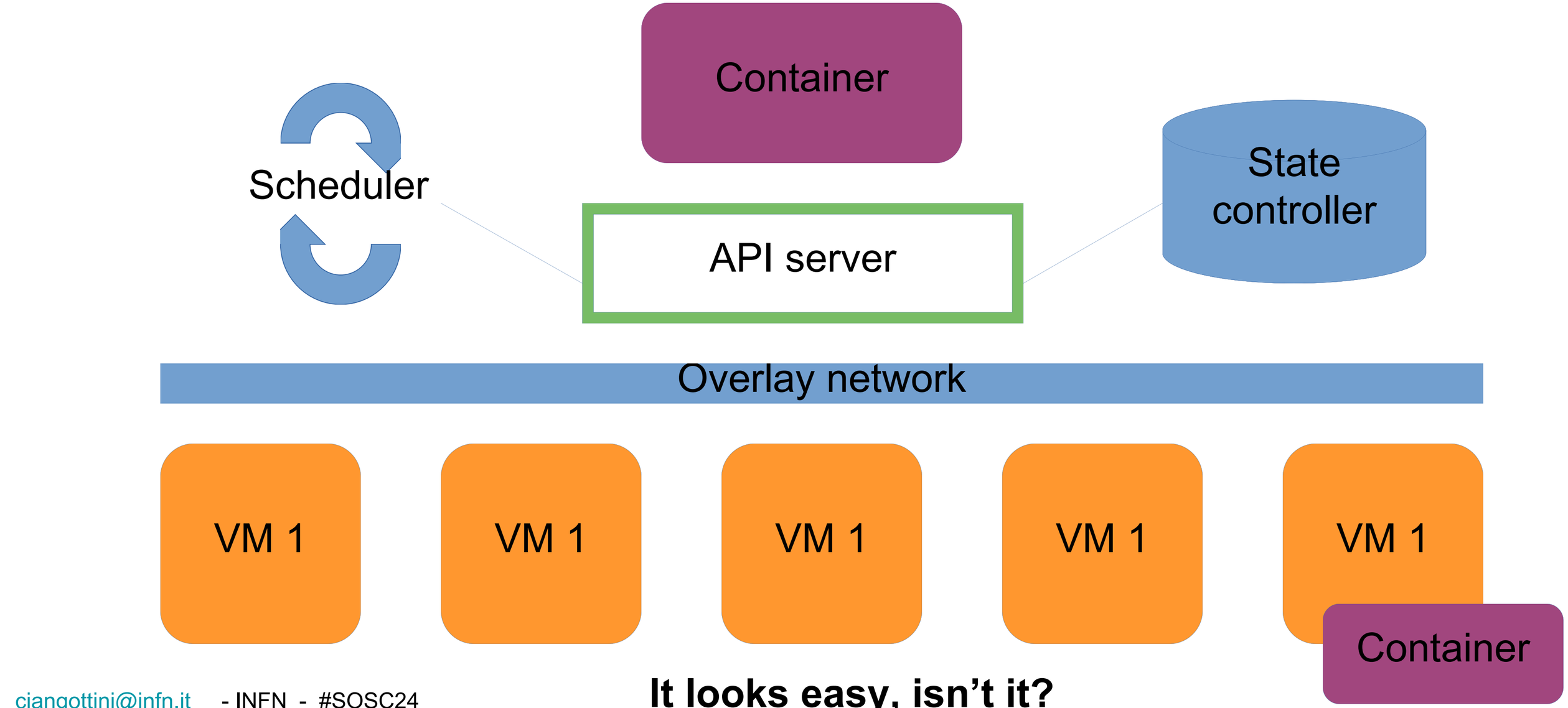
Take-home messagge session

Introduction

- We explored how containers help us to easily create applications that are – as the name says – self-contained.
- We discussed docker applications and explored a bit the docker-compose

- What we need then? we'd explore how to effectively orchestrate many containers across distributed hosts.
    - container orchestration.

Orchestrating containers



It looks easy, isn't it?

Docker swarm

Docker Swarm is the traditional way of orchestrating containers with Docker through docker-compose. Compared to other methods we'll see later, it is **relatively easy to use**. Its main features are:

- Cluster management **integrated with Docker Engine**: no other software than docker is needed.
- **Decentralized design**: this means that any node in a Docker Swarm can assume any role at runtime.
- **Scaling**: the Swarm manager can **automatically scale up and down services**, adding or removing tasks.
- **Desired state reconciliation**: if something happens to a Swarm cluster (e.g. some containers crash), the Swarm manager will try to reconcile the state of the cluster to its desired state (e.g. bringing up some more containers).

Docker Swarm (cont)

Docker Swarm features, continued:

- **Multi-host networking**: the Swarm manager can handle an overlay network spanning your services.
- **Service discovery**: there is a DNS server embedded in each Swarm. The Swarm manager discovers services and assigns to each of them a unique DNS name.
- **Load balancing**: you can specify how to distribute services among nodes.
- Secure by default: the communication among all nodes in a Swarm cluster is protected by the cryptographic protocol called TLS (Transport Layer Security).
- **Rolling updates**: if anything goes wrong, you can roll-back a task to a previous version of the service.

Where to get started

Hands-on with Docker Swarm:

- https://docs.docker.com/engine/swarm/swarm-tutorial/

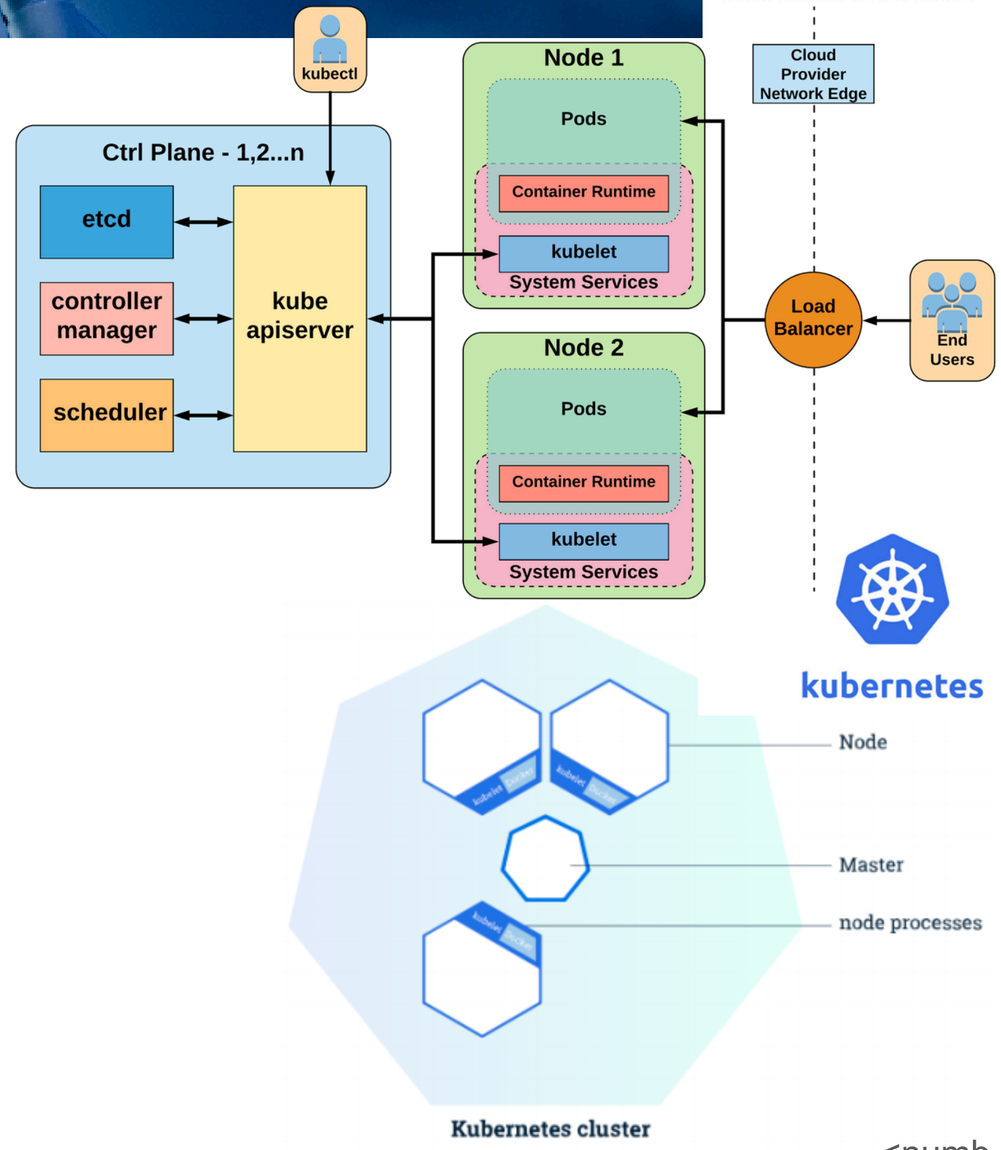
  You will learn how to:

  - Create a Swarm service
  - Deploy a load balancer
  - Create swarm cluster
  - Create swarm service

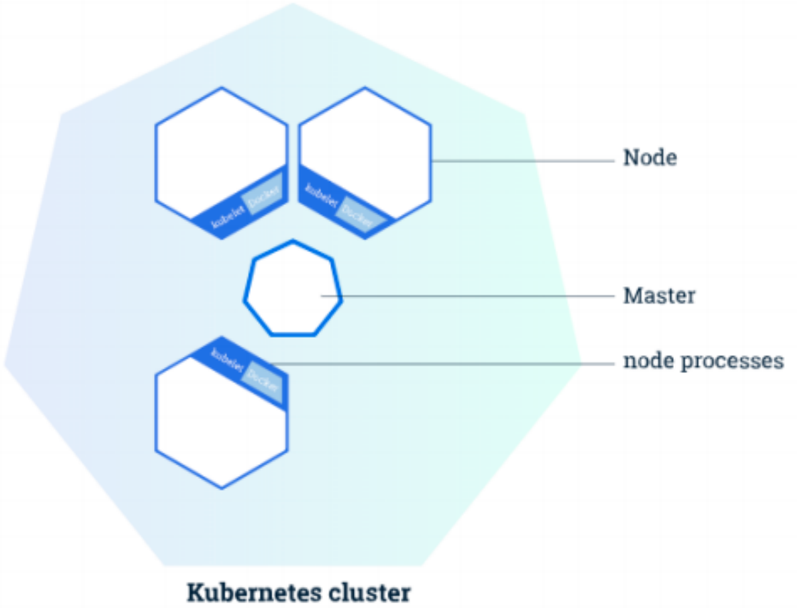
Kubernetes

Kubernetes is an **open-source platform that coordinates a highly available cluster of computers that are connected to work as a single unit**. It is backed by Google and RedHat.

- **Applications** need to be **containerized**.
- Kubernetes automates the distribution and scheduling of application containers across a cluster in a fairly efficient way.
- A Kubernetes cluster can be deployed on either physical or virtual machines.



Kubernetes Cluster Resources



A Kubernetes cluster consists of **two types of resources:**

- **The Master coordinates the cluster**
- **Nodes are the workers that run applications**

The Master is responsible for managing the cluster

- coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

  A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster

Kubernetes Master/node processes

The Kubernetes Master is a collection of three processes that run on a single node in your cluster, which is designated as the master node. These processes are:

- kube-apiserver
- kube-controller-manager
- Kube-scheduler

Each **individual Node in your cluster runs two processes**:

- kubelet, which communicates with the Kubernetes Master.
- kube-proxy, a network proxy which reflects Kubernetes networking services on each node.

Moreover, **each Node runs a container runtime** (like Docker) responsible for **pulling** the container **image from a registry**, **unpacking** the container, and **running the application.**

- A Kubernetes cluster that handles production traffic should have a minimum of three nodes

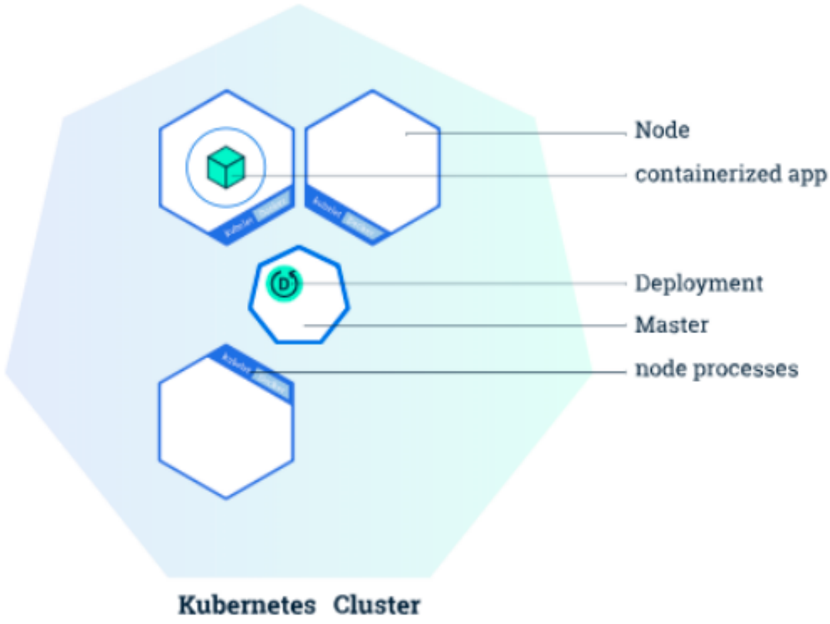
Kubernetes Objects

Kubernetes contains a number of abstractions that represent the state of your system: deployed containerized **applications and workloads**, their associated **network** and **disk** resources, and other information about what your cluster is doing.

- These abstractions are represented by objects in the Kubernetes API.
- The basic Kubernetes objects include:
  - **Volume**
  - **Namespace**
  - **Deployment**
  - **Pod**
  - **Service**

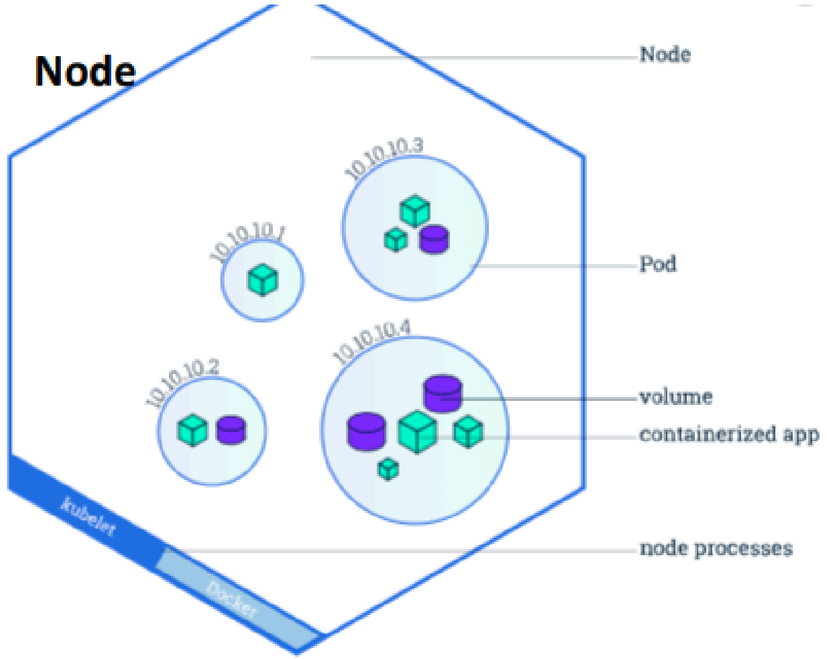
Kubernetes Deployment

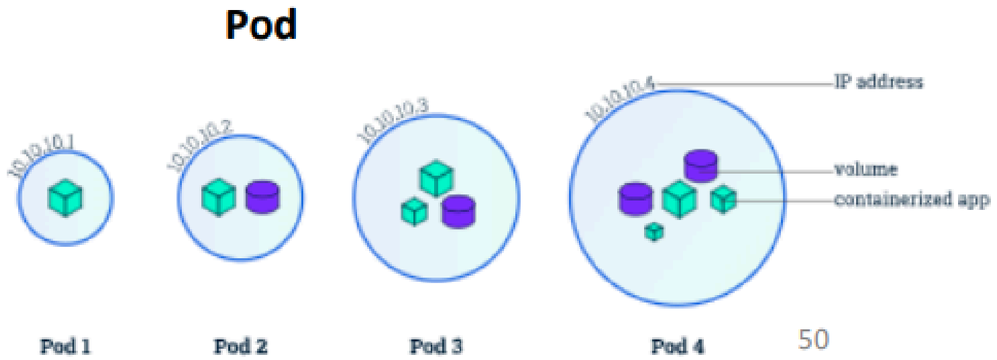
- Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment configuration.
- The Deployment tells Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes master schedules application instances onto individual Nodes in the cluster.
- Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down r is deleted, the Deployment controller replaces it. This provides a self-healing mechanism to address machine failure or maintenance.
- In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to applications



Kubernetes Pod

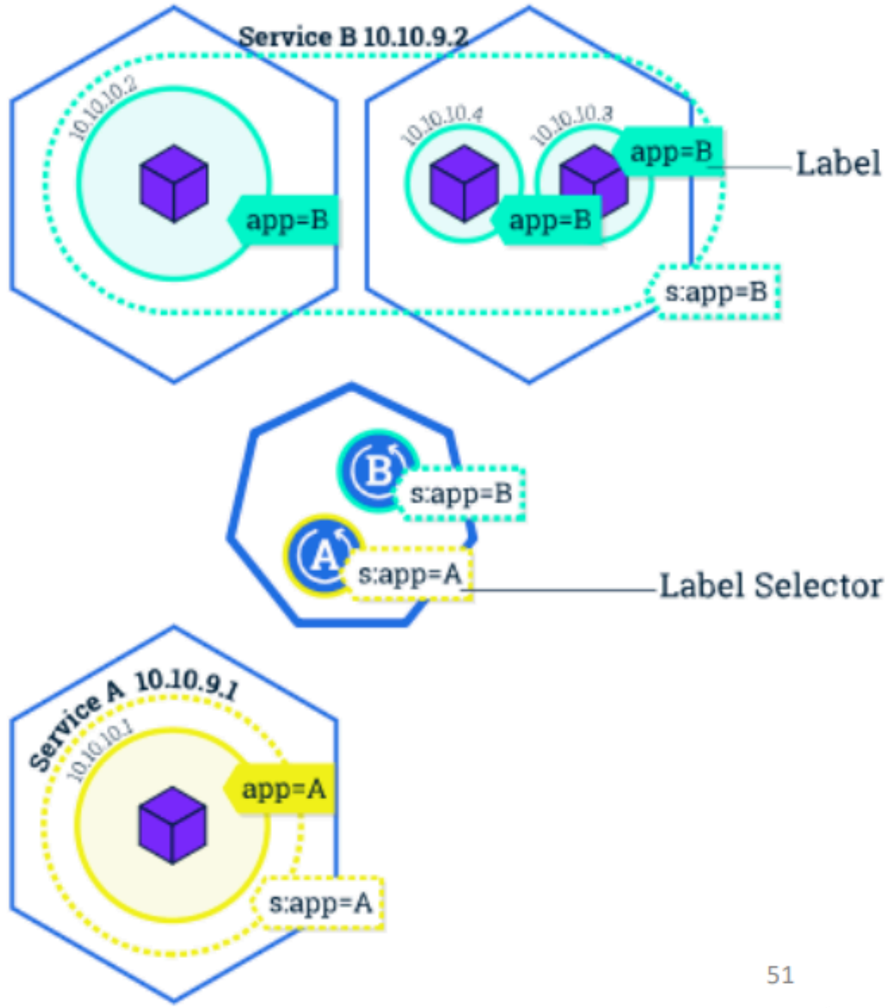
- A Pod is the basic building block of Kubernetes. It represents a running process on your cluster.
- A Pod encapsulates an application container, storage resources, a unique network IP, and options that govern how the container(s) should run.
- Pods that run a single container. The “one -container - per - Pod” model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container, and Kubernetes manages the Pods rather than the containers directly.
- Pods that run multiple containers that need to work together. A Pod might encapsulate an application composed of multiple co - located containers that are tightly coupled and need to share resources. The Pod wraps these containers and storage resources together as a single manageable entity.





Kubernetes Services

- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access it.
- Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic.
- Services match a set of Pods using labels and selectors, a grouping primitive that allows logical operation on objects in Kubernetes.
- Labels are key/value pairs attached to objects and can be used in any number of ways:
  - Designate objects for development, test, and production
  - Embed version tags
  - Classify an object using tags



Kubernetes as a Service

Deploying and managing a Kubernetes cluster is generally not trivial (that's why Minikube was introduced), since it requires effort and several skills.

- **It would be nice to automatize this part as well, and focus just on deploying our containers on a Kubernetes cluster that somebody else instantiates for us.**

  Many Cloud providers give us just that: a Kubernetes as a Service.
- Amazon provides what they call the "Elastic Container Service for Kubernetes", or EKS for short. Other providers have similar offerings.
- **INFN Cloud** also offers its own

  Monitoring tools: Headlamp or Portainer

Tutorial ref K8s

https://kubernetes.io/docs/tutorials/kubernetes-basics/

Here you can see in 4 episode, what it took to build the SOSC platform you are using right now (trivial issues included):
Link to youtube series

On Thursday we will see how all of that can be further simplified!

Docker compose vs Kubernetes

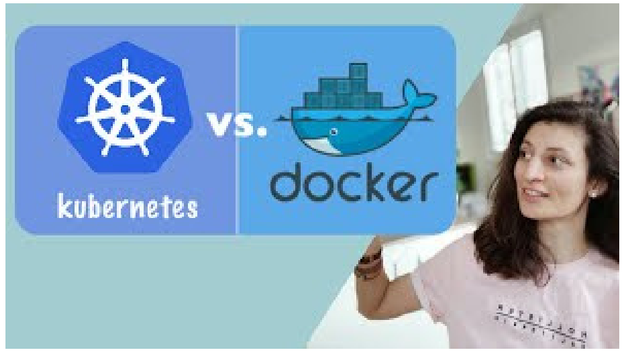
#SOSC24

INFN Istituto Nazionale di Fisica Nucleare

We have seen (with different degrees of in-depth analysis) **the three current major solutions for container or resource orchestration**.
Question now should be: which one to use ?
Some general considerations on when to use what:

- **Docker Swarm/Compose for smaller projects and for testing purposes**. Easy to use if you are already familiar with Docker.
- **For larger, enterprise-like solutions, Kubernetes**. It's also "the Google way of doing it". But mind the rather steep learning curve.

https://www.youtube.com/watch?v=9_s3h_GVzZc



https://labs.play-with-k8s.com