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, May 31st 2022

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
  - has the same lifetime as the Pod
  - it outlives the containers of the Pod

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

- 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.
  - Users request for Persistent Volume resources based on size, access modes, etc. via a PVC
  - Once a suitable Persistent Volume is found, it is bound to a Persistent Volume Claim
  - After a successful bind, the PersistentVolumeClaim resource can be used in a Pod.
  - 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.

Exercise 32 – Volumes

- **Time:** ~15 minutes
  - 6 minutes: *Try by yourself*
  - 9 minutes: *Check, Verify, Ask*

Description: Create two webservers: the first with ephemeral storage and the second with persistent storage and then check the differences.
In the second case the document root is exposed via a Volume.

There will be a built-in storage class so that I can deploy applications that request persistent volume claims.

Ensure that modification on the document root are persistent across pod restarts: If my pod restarts I want that pod to be scheduled such that the persistent volume claim is available again to it.This ensures that if I have to restart my pod will always come back with access to the same data.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e32

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]

Exercise 33 – Helm installation

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

Description: Install Helm and practice with basic commands

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e33

Ingress (1/3)

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

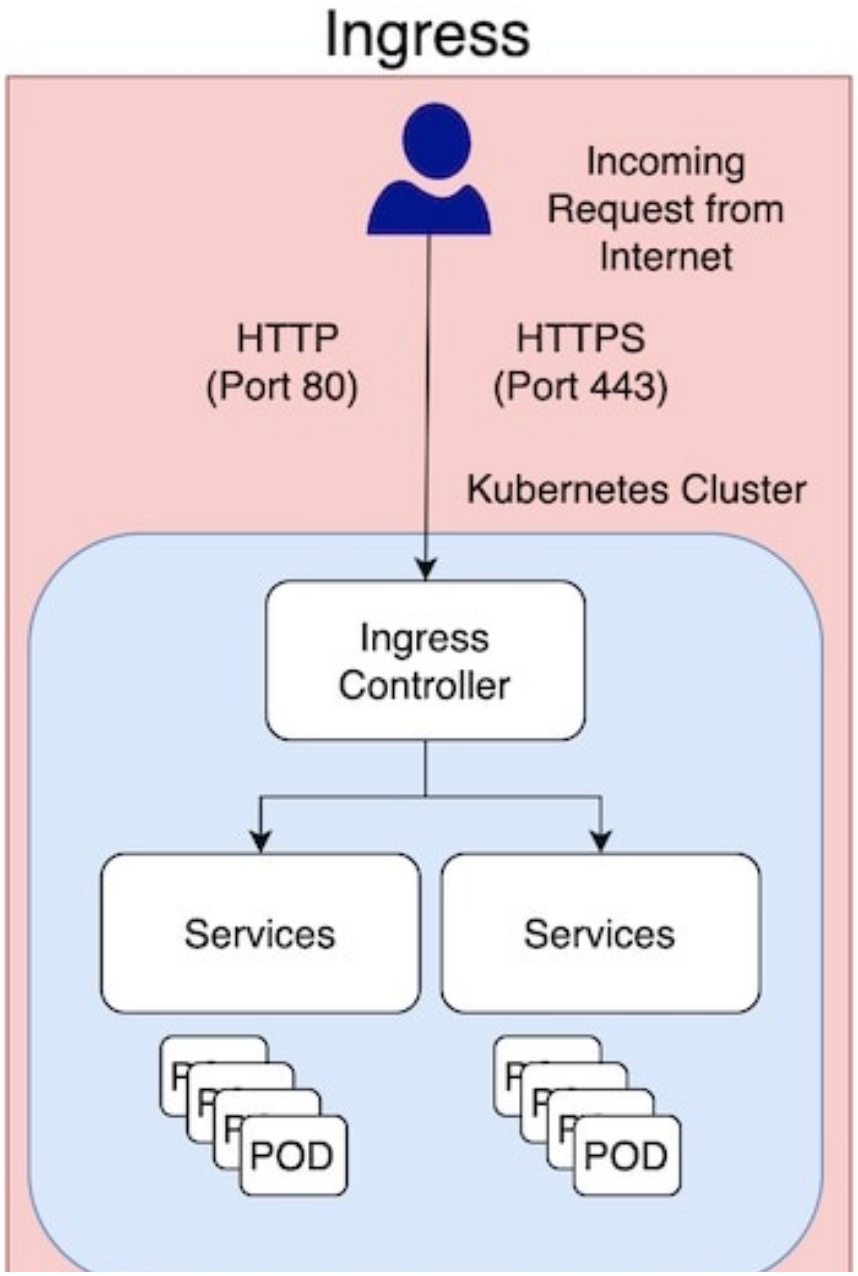
- With Services, routing rules are attached to a given Service.
  - They exist for as long as the Service exists.

- 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

Exercise 34 – NGINX Ingress Controller

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

Description: Install the NGINX Ingress Controller in your kind cluster.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e34

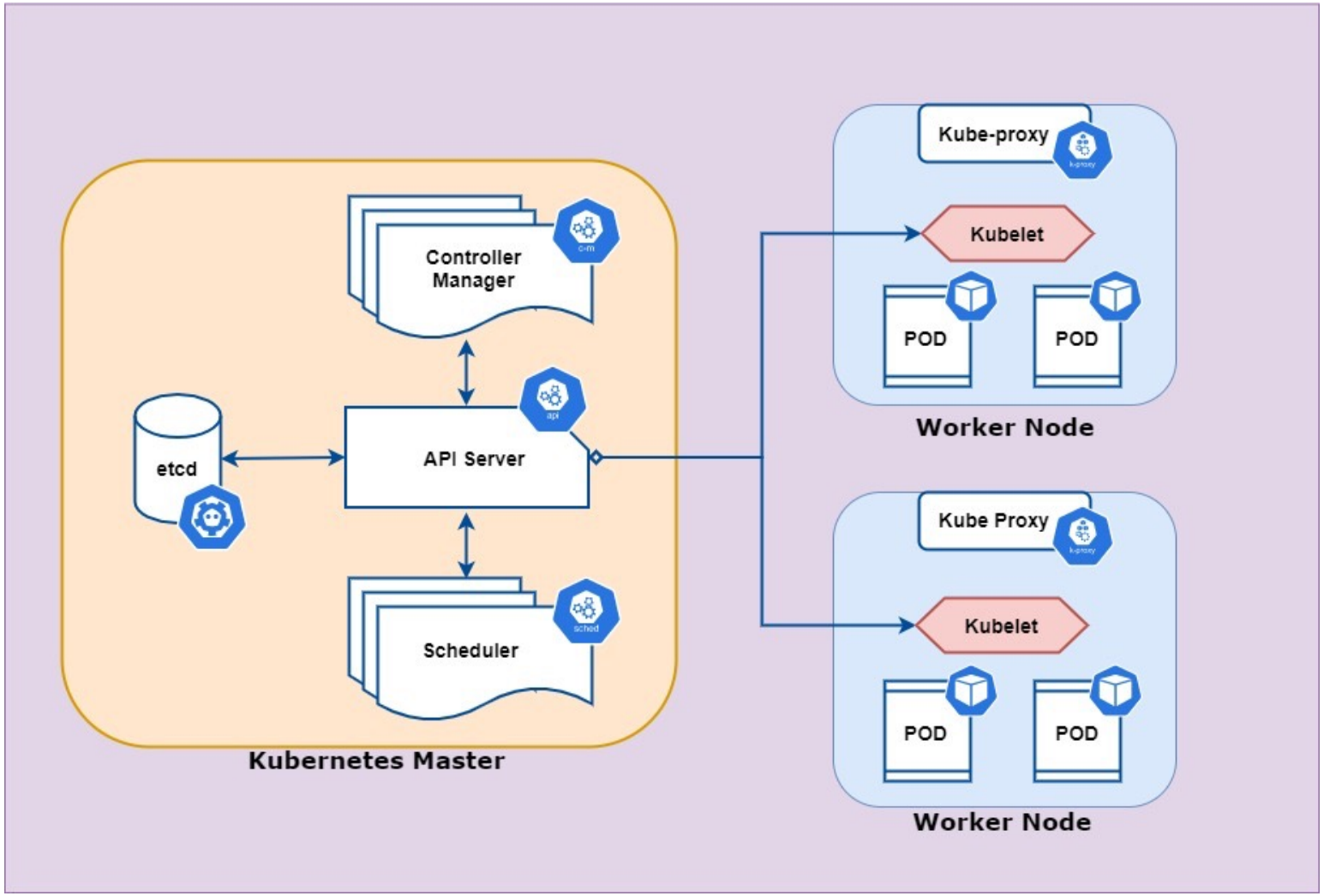
Exercise 35 – Ingress resource usage

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

Description: Create two services which prints different strings. Expose them via two different Services and finally create an Ingress rule that forward the traffic to the right Pod depending on the HTTP path.

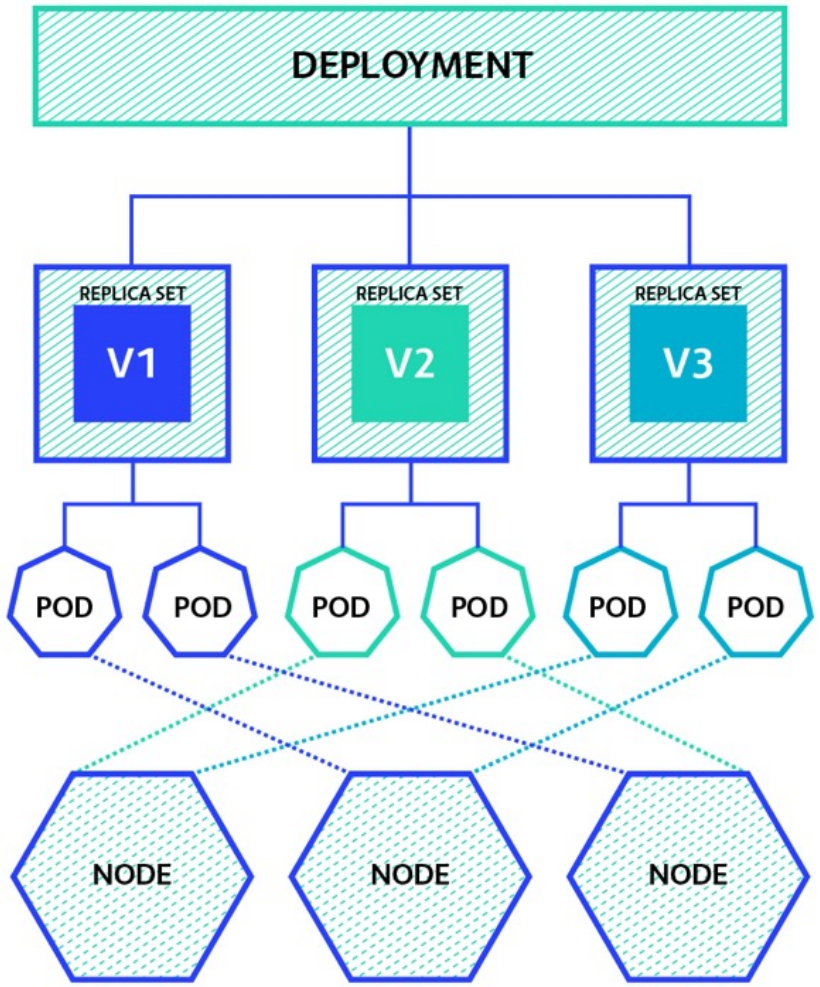
- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e35

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

k8s scheduling example



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

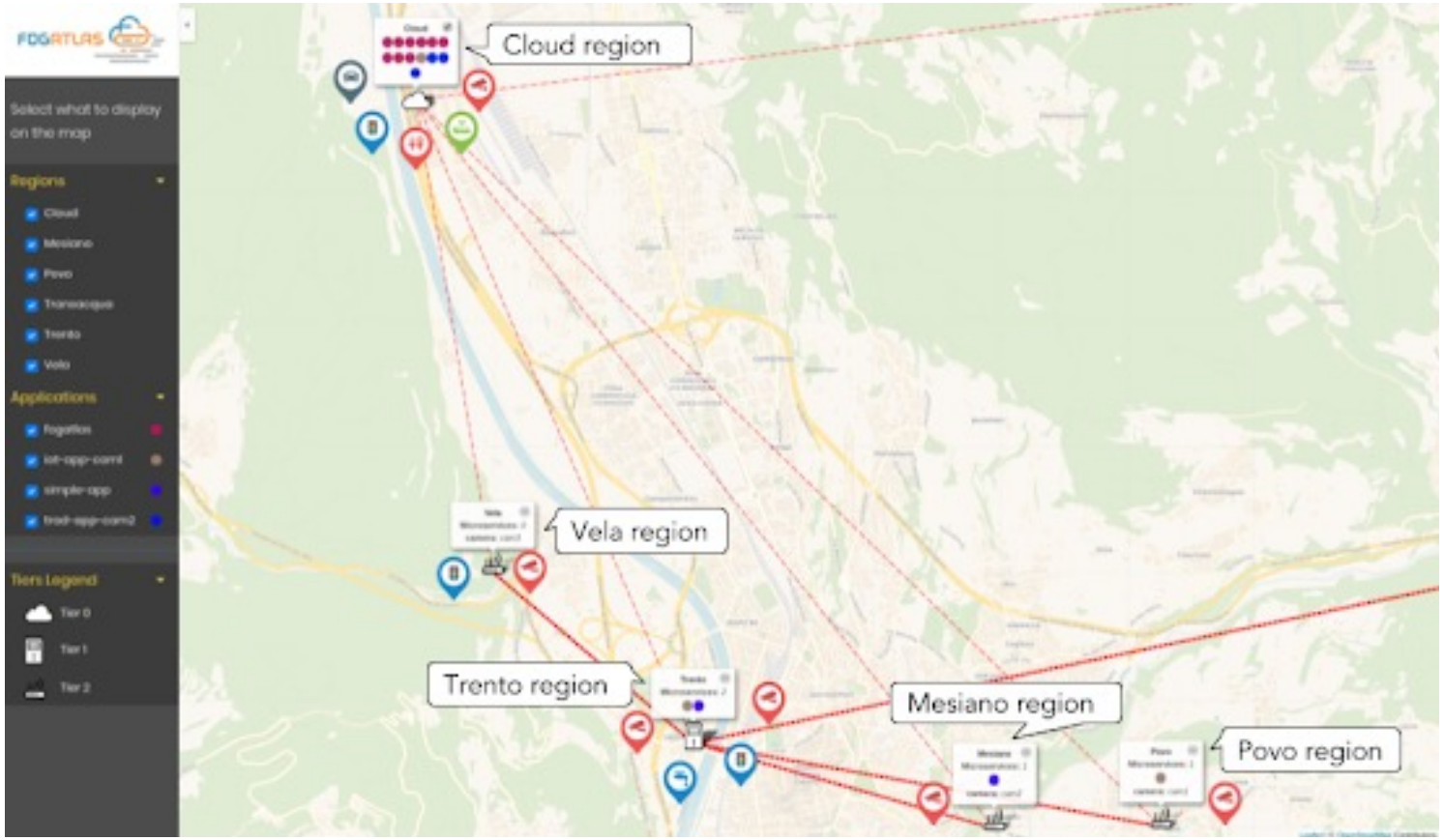
```
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: lb-example2
    spec:
      nodeSelector:
        region: "EDGE"
      containers:
      - name: worker
        image: python:3.6-alpine
        command:
        - "/bin/sh"
```

Exercise 36 – Pod placement

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

Description: Modify the lb-example Deployment adding a `NodeSelector` requiring a label `region=EDGE` constraint and deploy it on a specific node. If this does not work out of the box, try to inspect why the scheduler is not able to complete the request and apply corrective actions

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e36



FOGATLAS

DRIVES APPLICATIONS THROUGH THE FOG

- FogAtlas
- **Cloud4IoT: a heterogeneous, distributed and autonomic cloud platform for the IoT**
- **Foggy: A Platform for Workload Orchestration in a Fog Computing Environment**
- **Cutting Throughput on the Edge:App-Aware Placement in Fog Computing**
- **Other papers here…**