# 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 6th 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

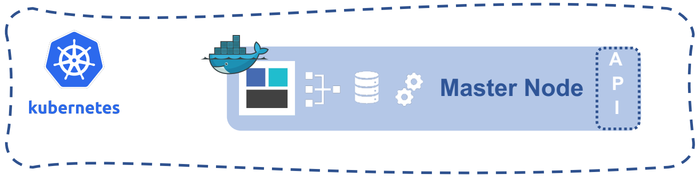
- Slides
  - Uploaded before any lesson in Moodle
- Repositories of exercises
  - https://gitlab.fbk.eu/dsantoro/fcc-lab-2022
- Lab Virtual Machine:
  - **Lab VM on Azure (reference for exercises)**
  - Vagrant and VirtualBox on your laptop (possible choice)
    - https://www.virtualbox.org/, https://www.vagrantup.com/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

# Quick Recap & Today Lesson

- Recap of previous topics
  - Install a (single-node) k8s cluster in kind
  - Objects in k8s
  - Pod, ReplicaSet, Deployment (missing exercises)
- Multi-Node k8s cluster & Networking
  - Install a multi-node cluster
  - Overlay Network
  - Pod-to-Pod communication
  - External-World-to-Pod communication
  - K8s Services
  - Load Balancer

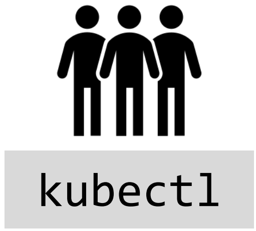
# From Single-Node to Multi-Node Cluster

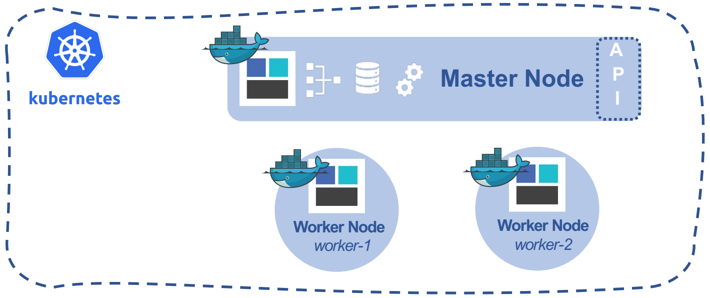
Single Node Cluster




This is also a worker node because kind sets it as a worker by default. Usually, in production clusters the master node does not host any workload.

This is driven by a **taint** on the node:
`node-role.kubernetes.io/master:NoSchedule`




Multi Node Cluster

# Exercise 24 – Create a multi-node k8s cluster

- **Time:** ~5 minutes
  - 5 minutes: Altogether, *Check, Verify, Ask*

**Description:** Delete the current cluster which is composed by a single worker node (the master node) and create a fresh new cluster composed by one master node and two worker nodes. Check cluster and nodes status.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e24

# 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.

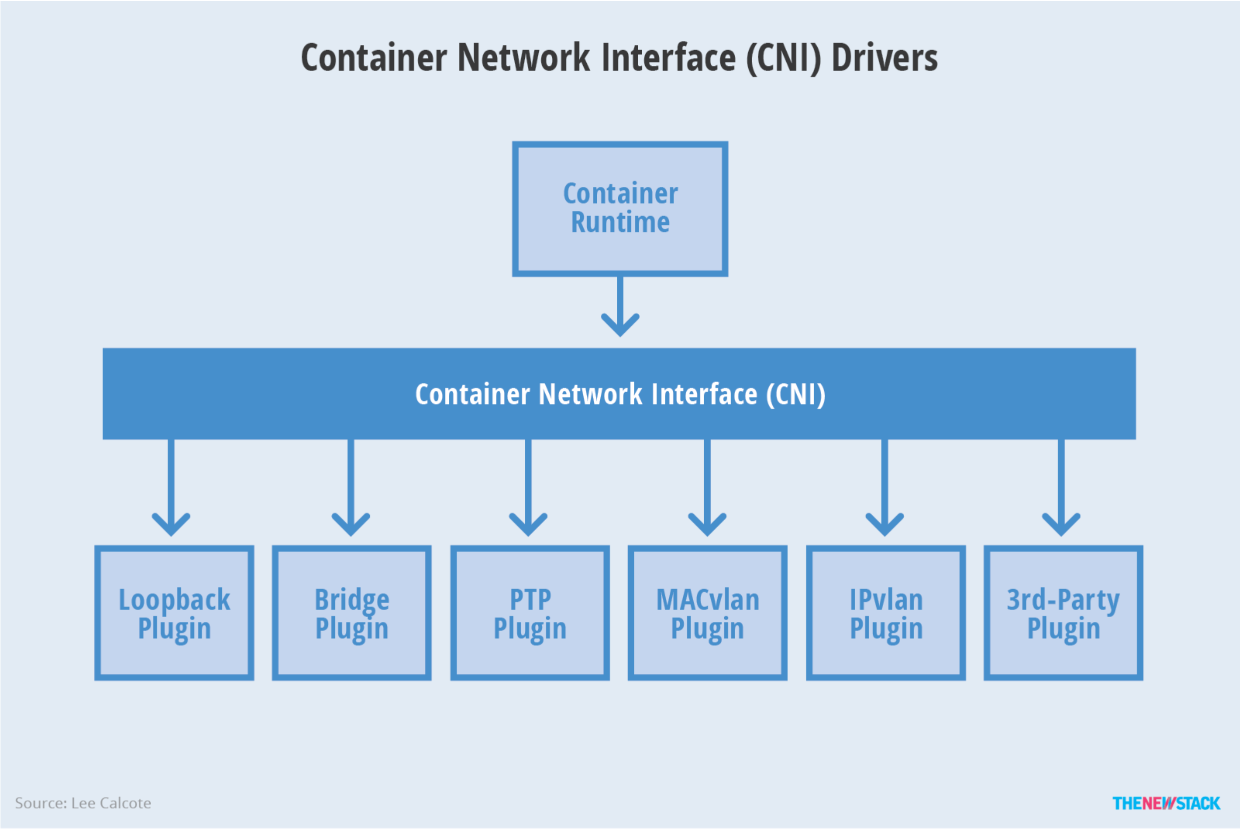
- All of the above are <u>networking challenges which must be addressed.</u>

# Container Network Interface (CNI)

- <u>In Kubernetes, each Pod gets a unique IP address</u>.

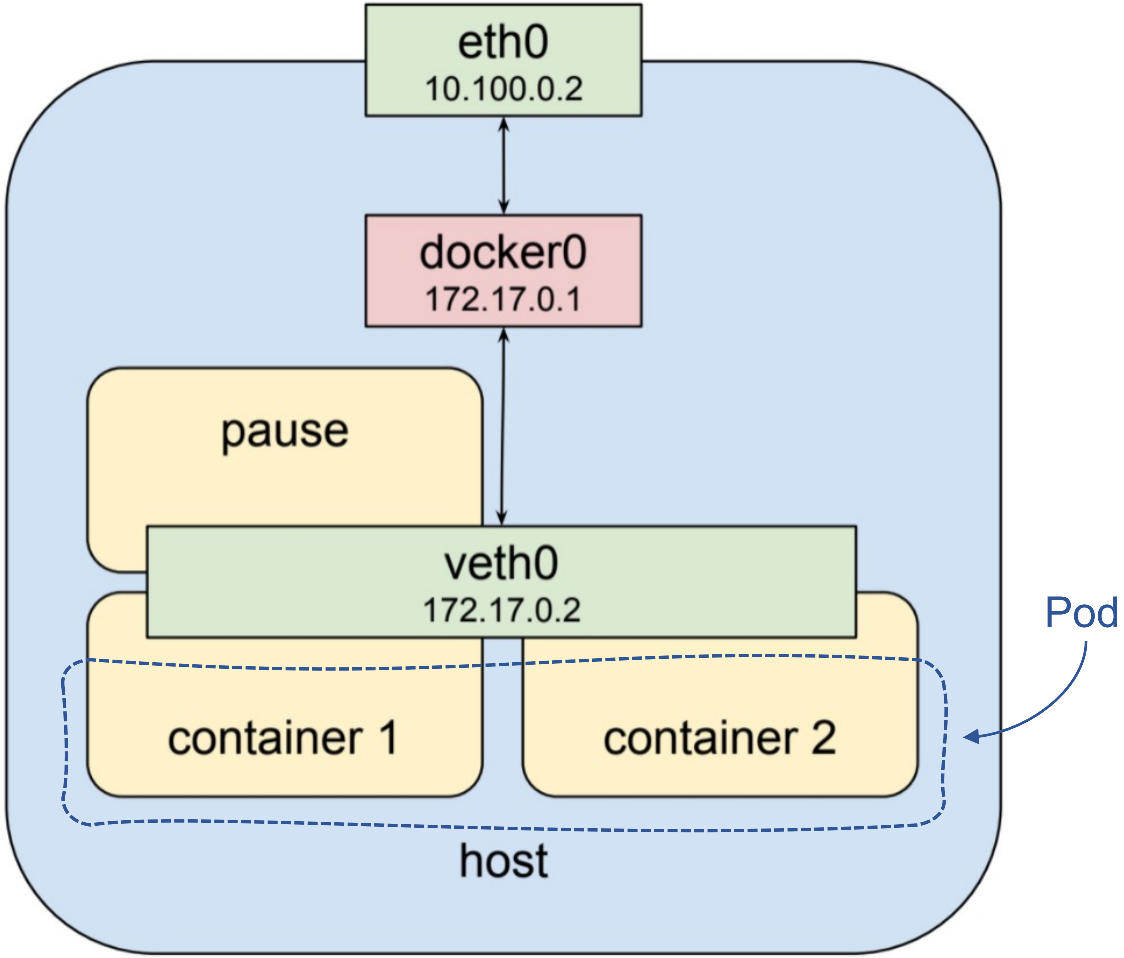
- For container networking, there are two primary specifications:
  - Container Network Model (CNM), proposed by Docker
  - Container Network Interface (CNI), proposed by CoreOS (used by k8s)

- The <u>Container Runtime offloads the IP assignment to CNI</u>, which connects to the underlying configured plugin, like Bridge or MACvlan, to get the IP address. <u>Once the IP address is given</u> by the respective plugin, <u>CNI forwards it back to the requested Container Runtime</u>.



# Container-to-Container Communication Inside a Pod

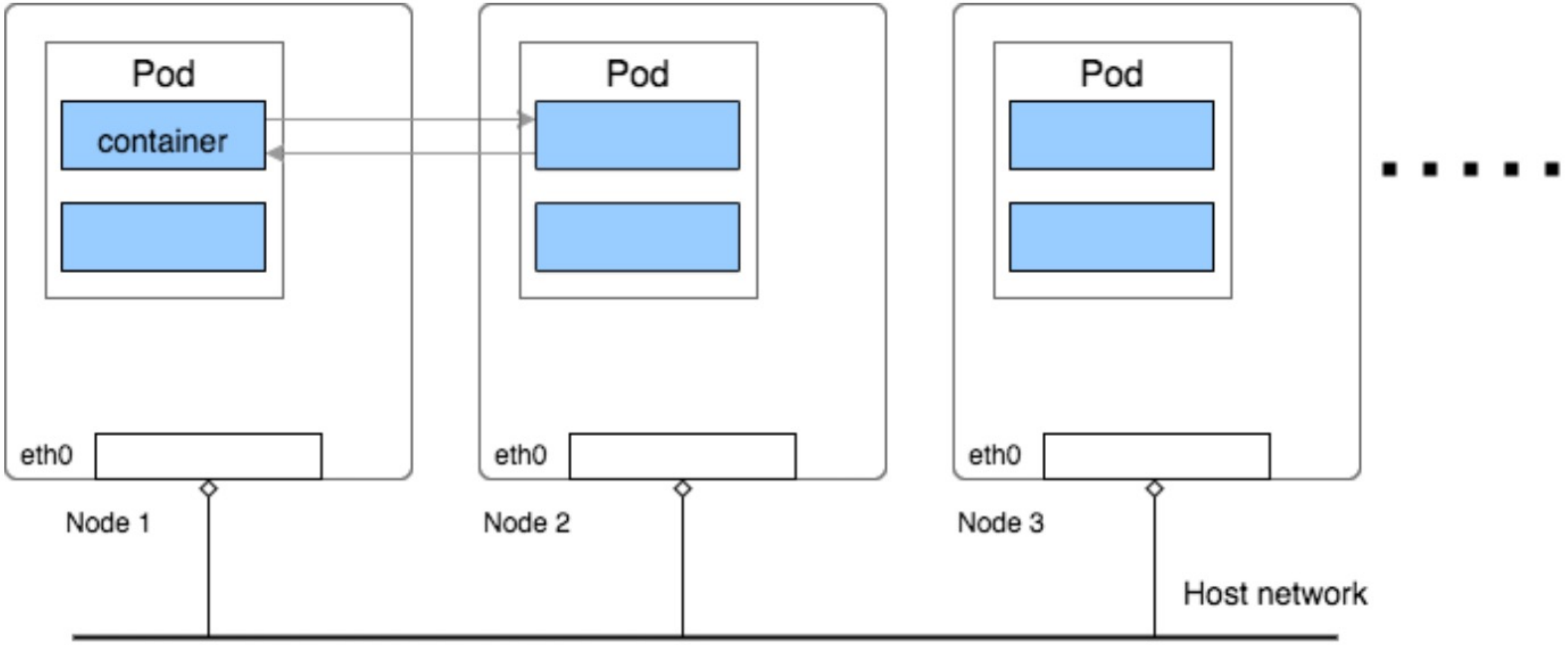
- With the help of the underlying Host OS, all of the Container Runtimes generally create an isolated network entity for each container that they starts.
- On Linux, that entity is referred to as a Network Namespace. These Network Namespaces can be shared across containers, or with the Host Operating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

# k8s Networking




Pods in Kubernetes

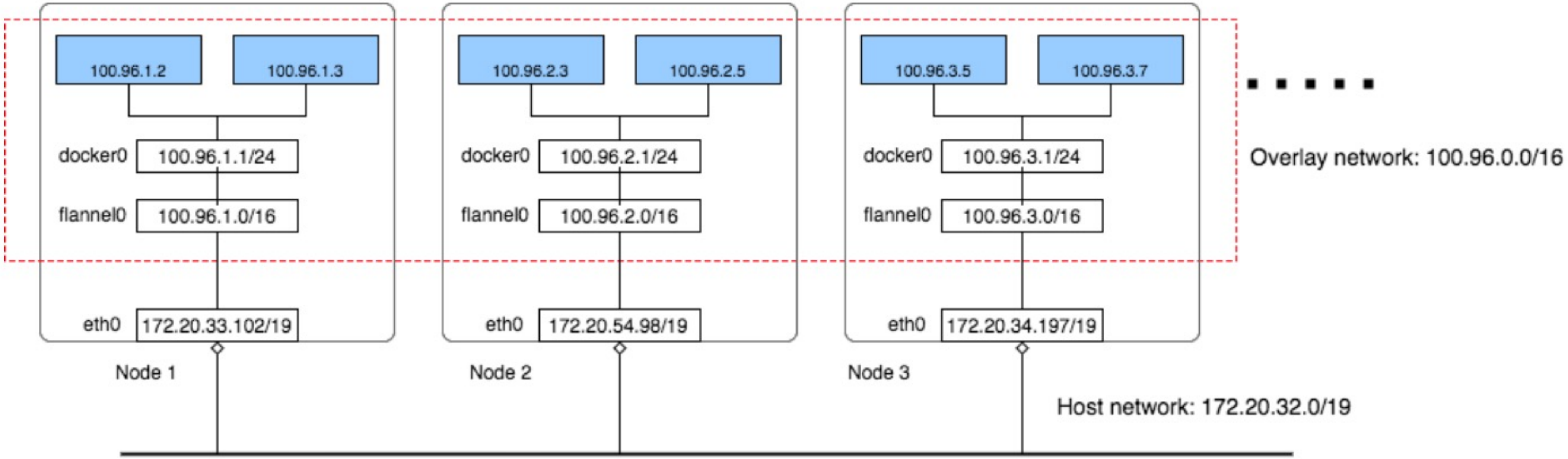
# Exercise 25 - Pod-to-Pod Communications

- **Time:** ~15 minutes
  - 7 minutes: *Try by yourself*
  - 8 minutes: *Check, Verify, Ask*

**Description:** Deploy two microservices on different worker nodes: a client and a server. To ensure the scheduler respects your requirements temporarily disable the scheduling on one worker node before installing the second microservice.
Test the communication from client to server ensuring the inter-cluster communications across Pods is working as expected.

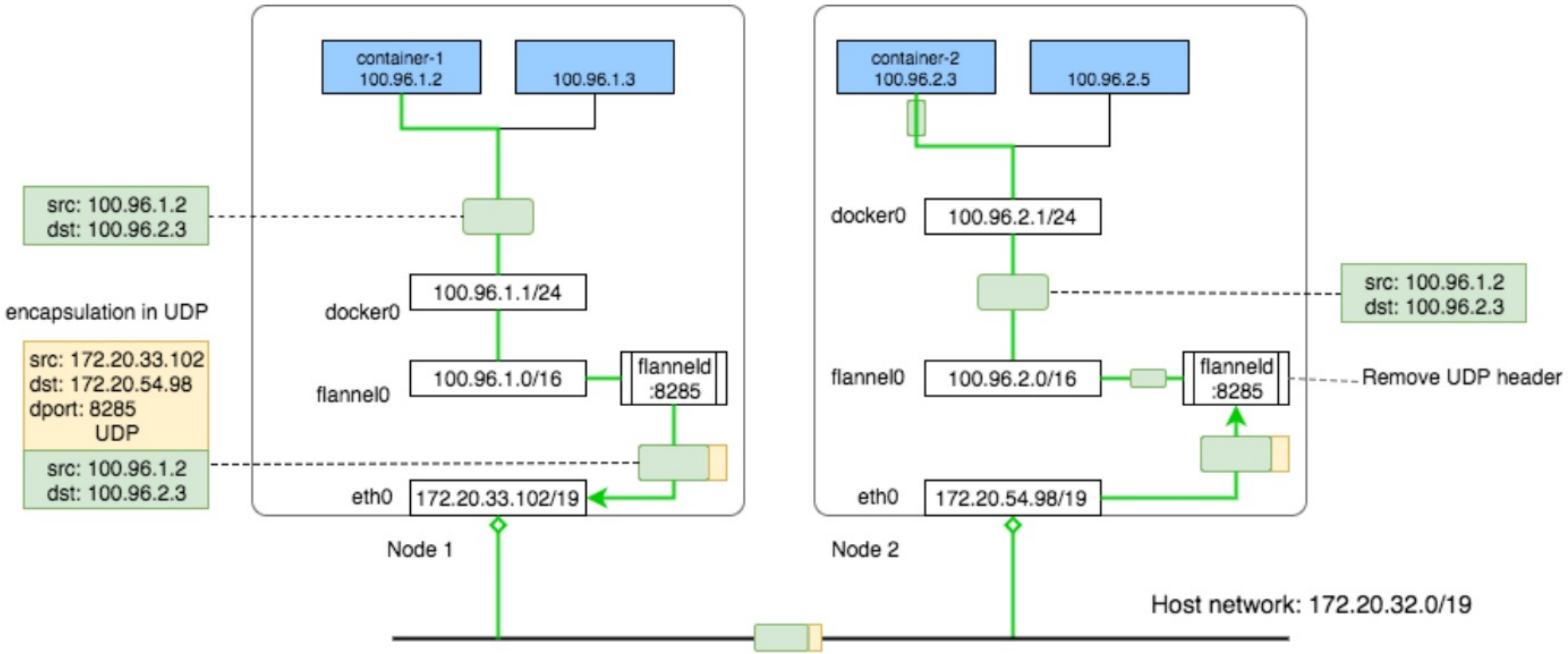
- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e25

# Overlay Network




Flannel overlay network

# Network Plugin - Flannel




Cross host communication

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

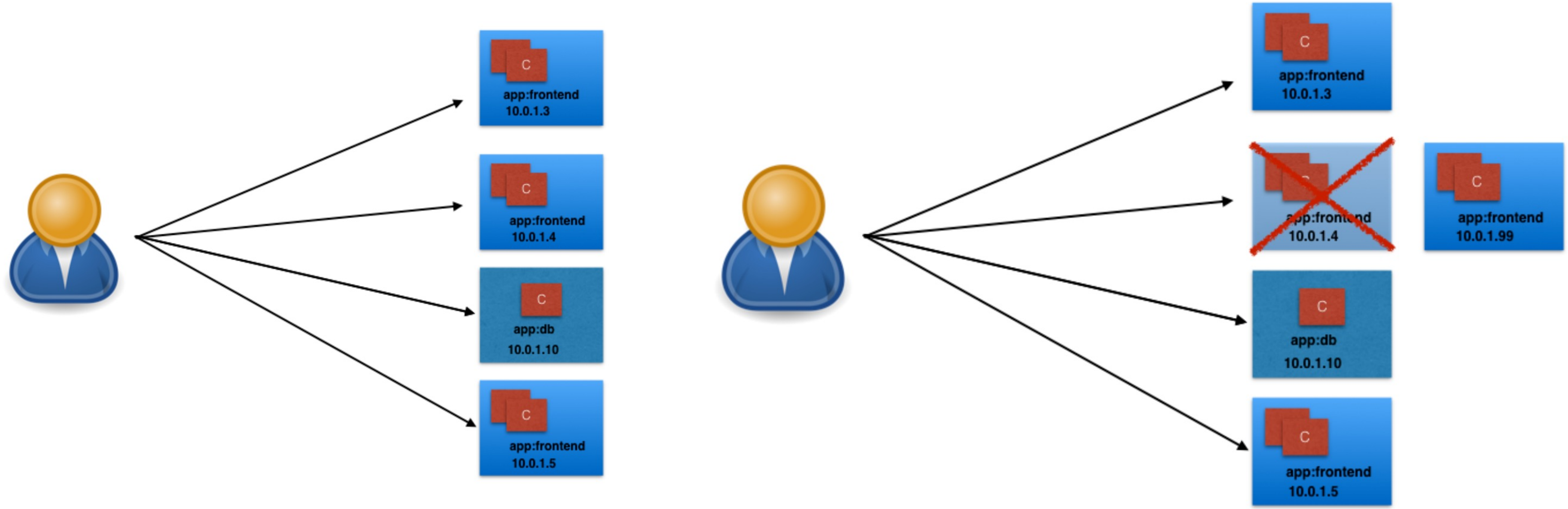
# External World-to-Pod Communication

- By exposing our services to the external world with **kube-proxy**, we can access our applications from outside the cluster.

- 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.

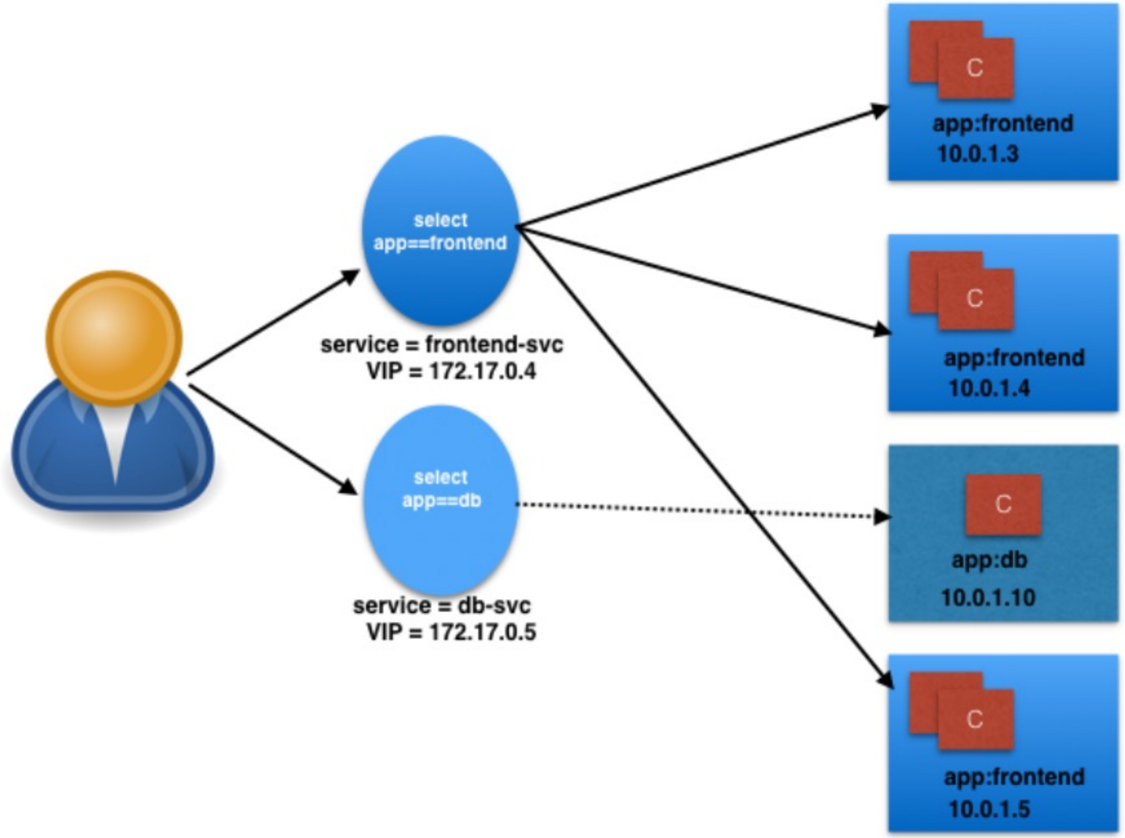


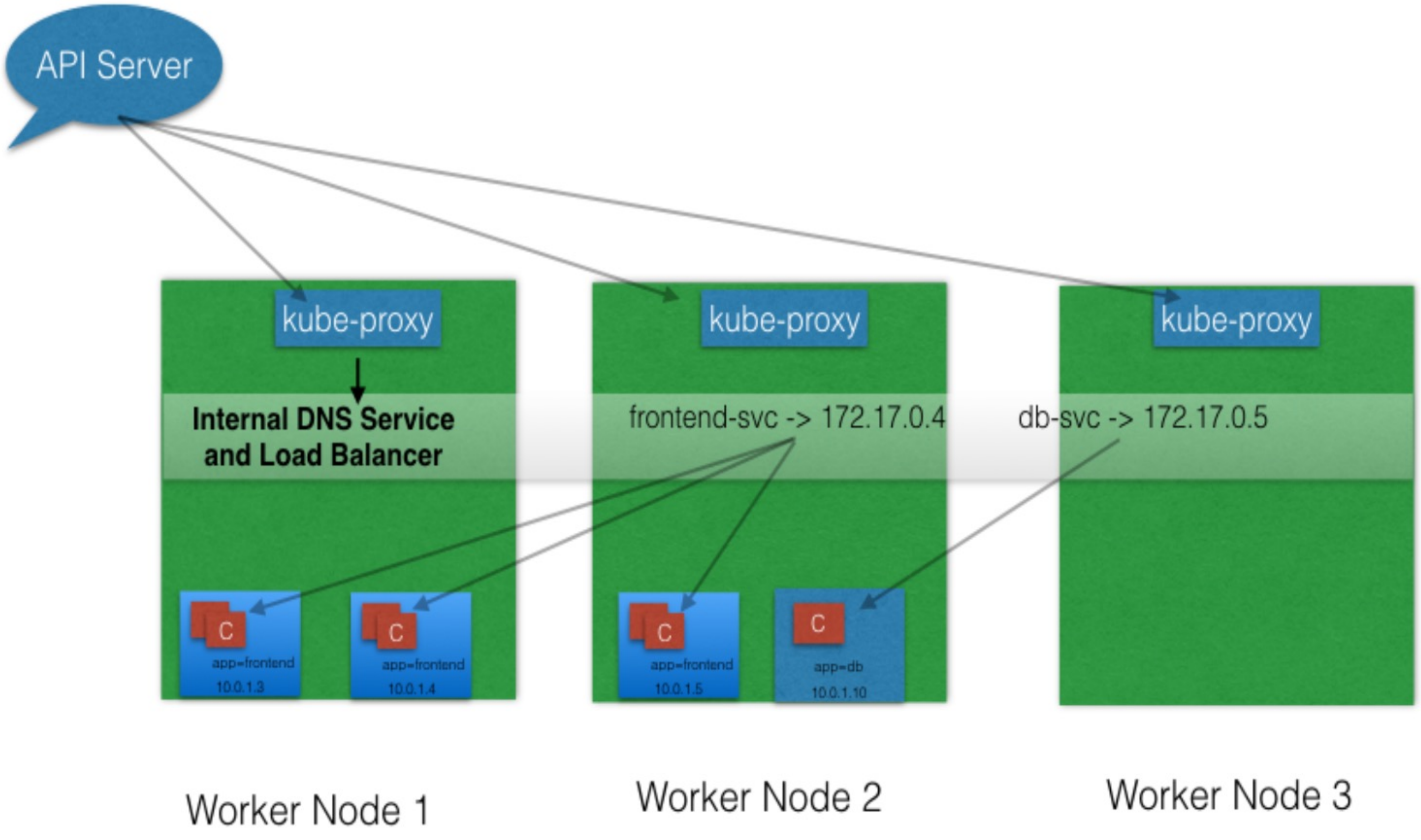
# Services 1/3

- Kubernetes provides a higher-level abstraction called Service, which logically groups Pods and 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

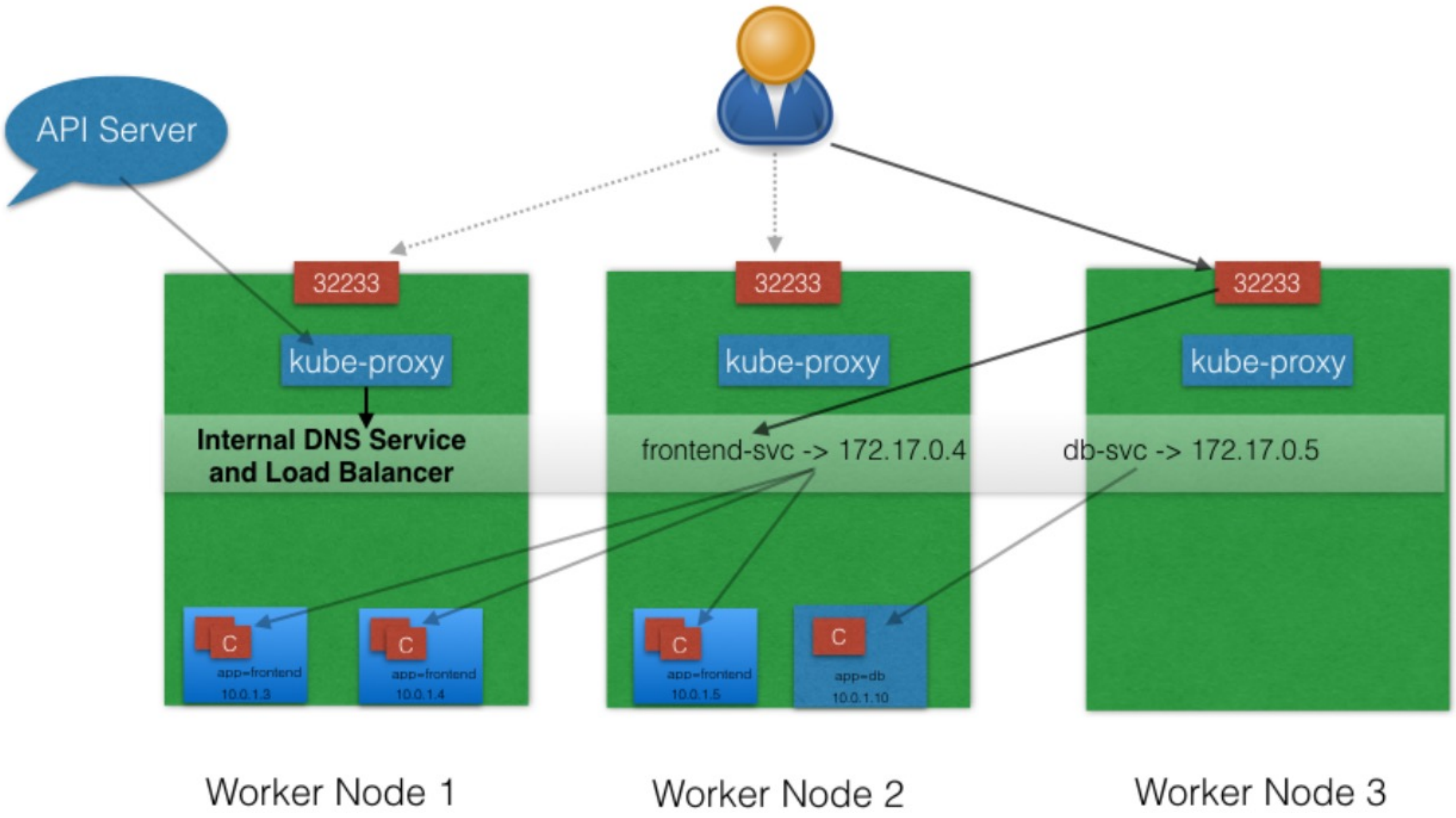
- All of the Worker Nodes run a daemon called kube-proxy, which watches the API Server on the Master Node for the addition and removal of Services and endpoints.

- 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.

# Node Port Service Type



# Exercise 26 – ExternalWorld-to-Pod Communications

FONDAZIONE BRUNO KESSLER

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

**Description:** Deploy a microservices that act as a server. It must expose a service showing the Pod name and the Worker node where it has been scheduled. Moreover this service should be exposed outside the cluster on a specific port, the 30000.
Finally expose the very same microservice using a random external port.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e26

# Exercise 27 - Load Balancing

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

**Description:** Scale the number of microservices of the server Deployment (of course is the ReplicaSet which scale) created during previous exercise 6 replicas. While scaling it look at the workload in the cluster and especially paying attention on how the scheduler spread the workload. Then have a look at how the Service resource has been modified by Kubernetes and finally try to access the server many times.

- **Instructions:**
  [https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e27](https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e27)