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 26th 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
  - Portable and Repeatable Env (CM, IaC, Ansible, Vagrant)
  - Docker Containers
  - IaaS and OpenStack
- Container Orchestration and Kubernetes
  - K8s Architecture
  - Install k8s
  - K8s Object model
    - Pod
    - ReplicaSet
    - Deployment

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]*

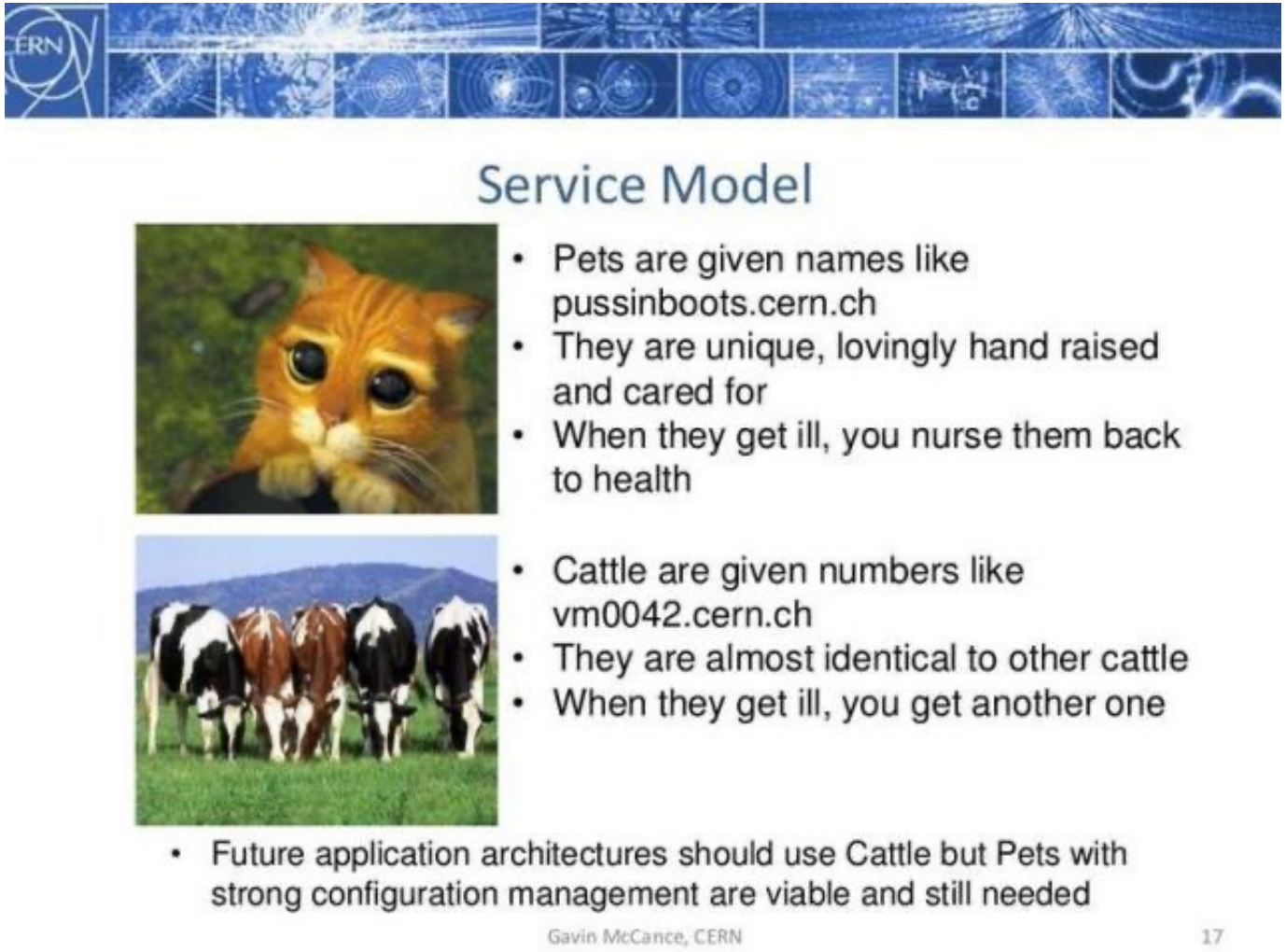
- **Cloud native** is about patterns to build software that scale on elastic infrastructure in fast way. *[ref]*

- Cloud Native Computing Foundation (CNCF)

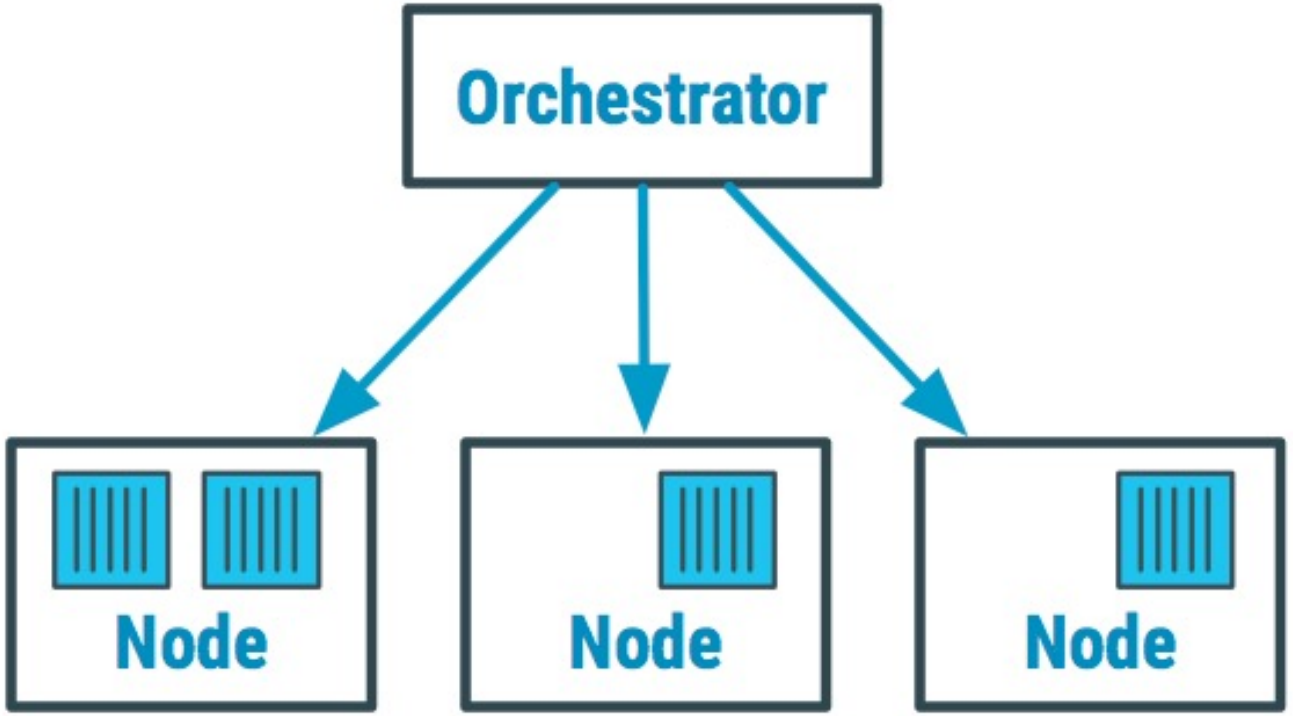
- CN is about (not only) Dev and Ops (DevOps), two main concepts:
  - The Twelve Factors Methodology (Dev)
  - Pets vs Cattle (Ops)

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]



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.

Container Orchestrators "war"

Apache Mesos (2009) (http://mesos.apache.org/)

- Support both containerized, and non-containerized workloads in a distributed manner

Amazon ECS (2014) (https://aws.amazon.com/ecs/)

- Hosted service provided by AWS to run Docker containers at scale

Kubernetes (2015) (https://kubernetes.io/)

- Started by Google, but now, it is a part of the CNCF projects

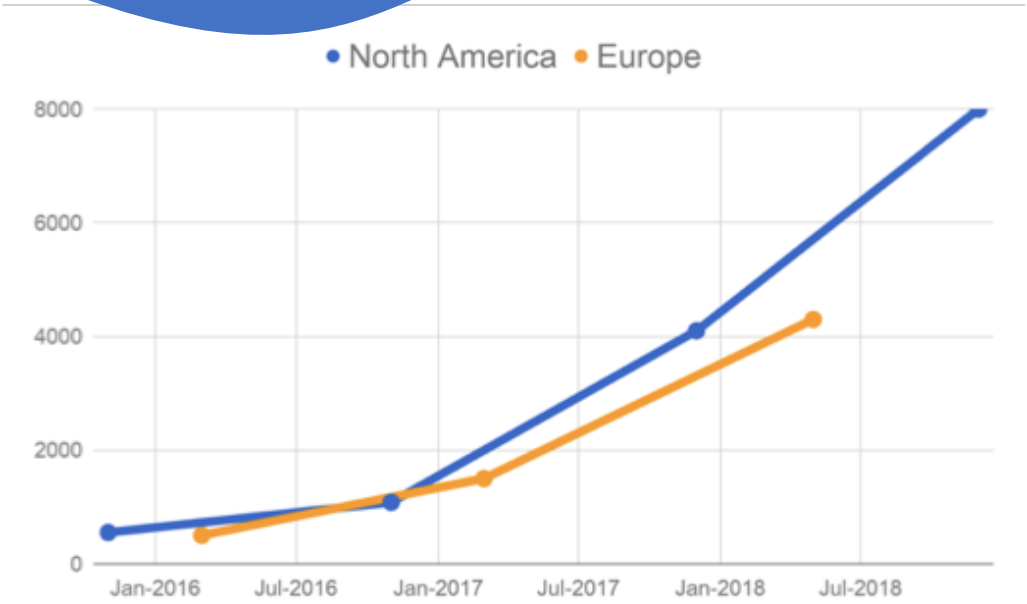
Hashicorp Nomad (2015) (https://www.hashicorp.com/)

- The Container Orchestrator provided by HashiCorp.

Docker Swarm (2016) (https://docs.docker.com/engine/swarm/)

- Special mode of Docker providing native clustering functionality

… and the winner is…



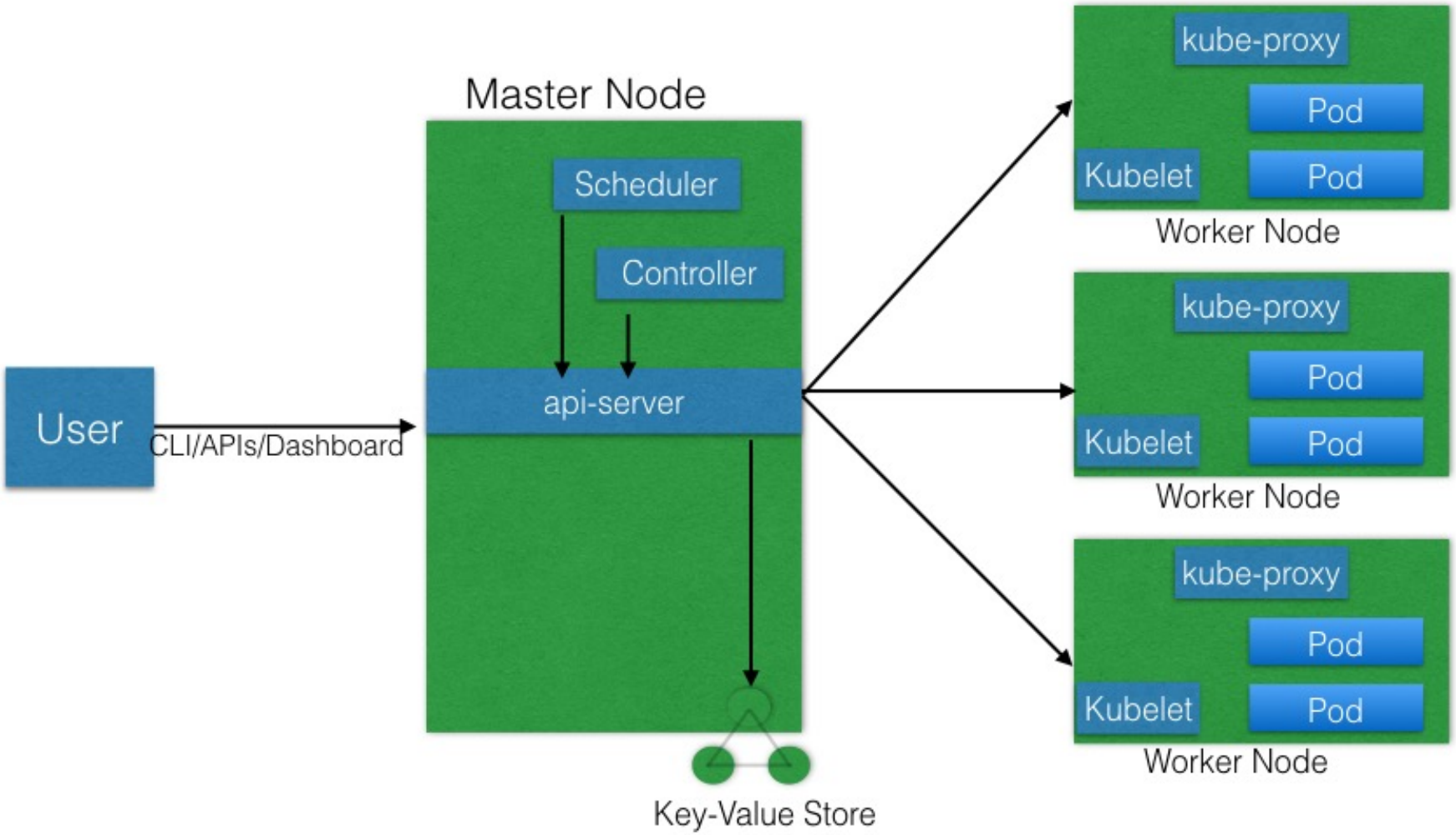
KubeCon participation from 2015 [ref]

- AWS, Azure, Google, Digital Ocean sells manages k8s service
- Docker (Swarm) supports k8s [ref]
- Many projects started (and still starting) on top of k8s ecosystem: https://www.cncf.io/projects/
- All mid/big Cloud providers that do not support k8s yet, are in late
- Some complex and huge projects run on k8s, eg: OpenStack
- More than 3,2k contributors, who, over time, have done almost 108k commits, as of today
- Case studies on k8s website. Some examples are: Wikimedia, eBay and SAP

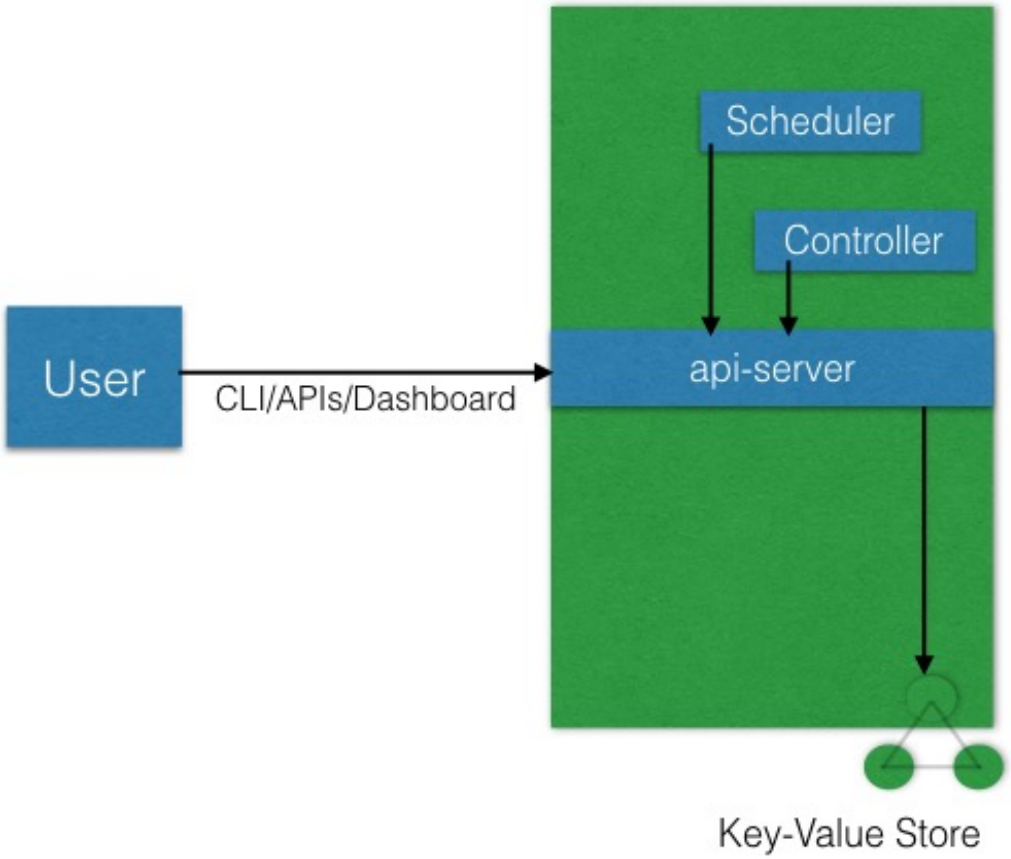
What is Kubernetes, aka k8s ?

- 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 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

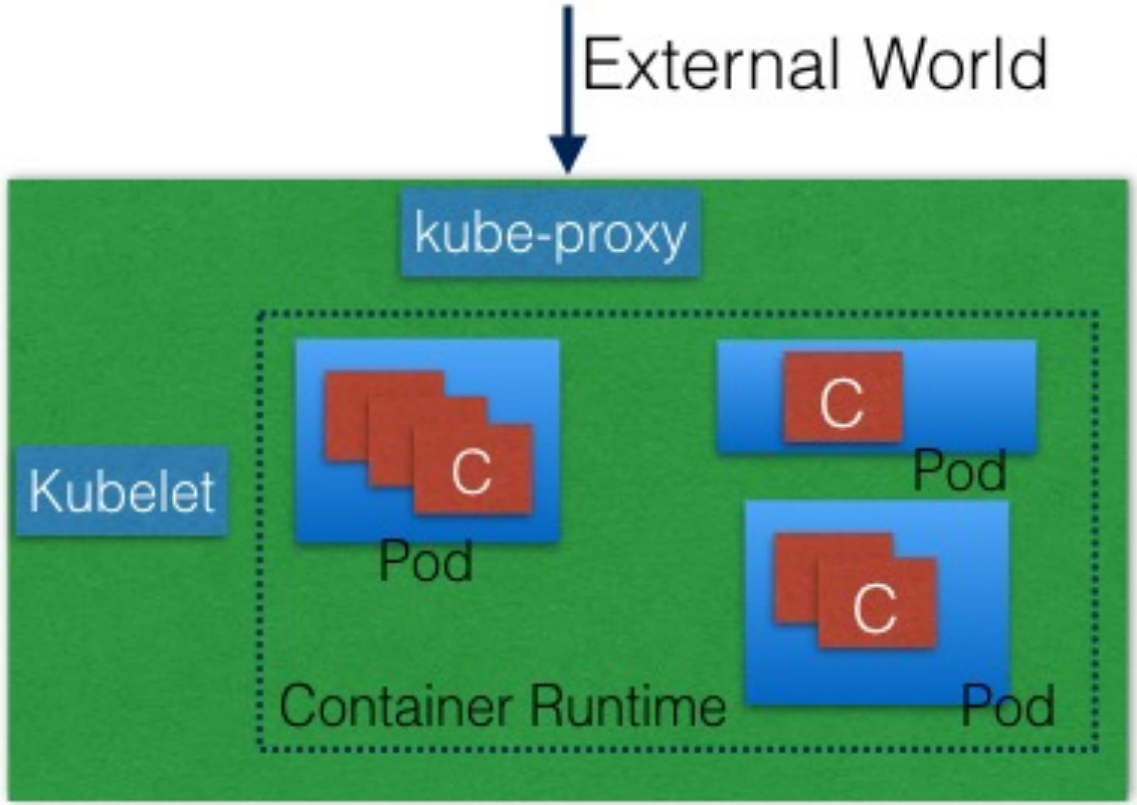
- **API Server:** A user/operator sends REST commands to the API Server, which then **validates** and **processes** the requests. After executing the requests, the resulting **state of the cluster is stored** in the distributed key-value store.

- **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.

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

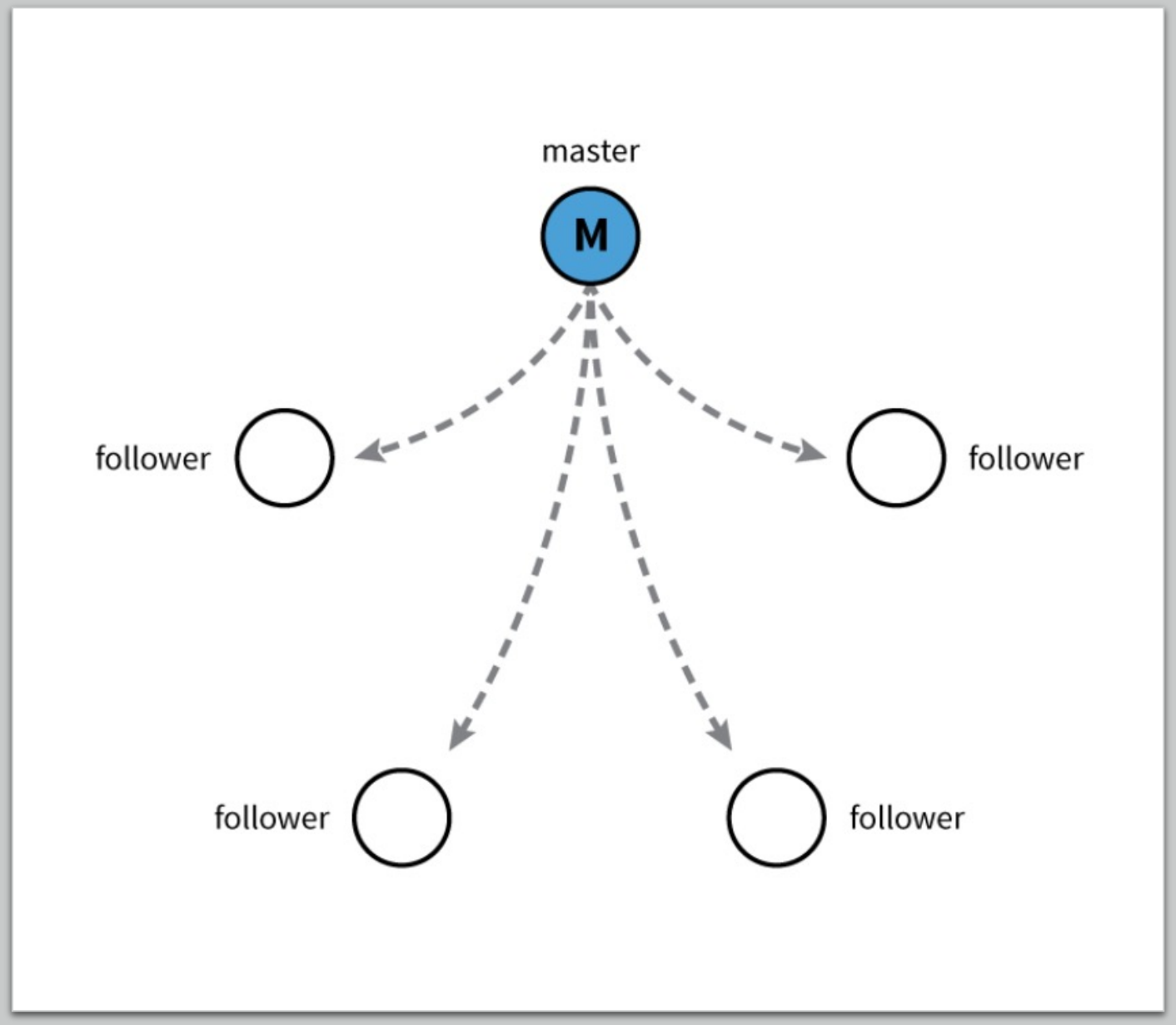
- **Container Runtime:** To run containers, we need a Container Runtime on the Worker Node. By default, Kubernetes is configured to run containers with Docker. It can also run containers using the rkt Container Runtime.

- **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 a 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.

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



Kubernetes Object Model

- Kubernetes has a very rich (and now extensible) object model by means of which it represents different persistent entities in the cluster

- Those entities describes:
  - 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.

- With each object, we declare our intent or desired state using the **`spec`** field. The Kubernetes system manages the **`status`** field for objects, in which it records the actual state of the object.

- At any given point in time, the Kubernetes Control Plane tries to **match the object's actual state to the object's desired state**.

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

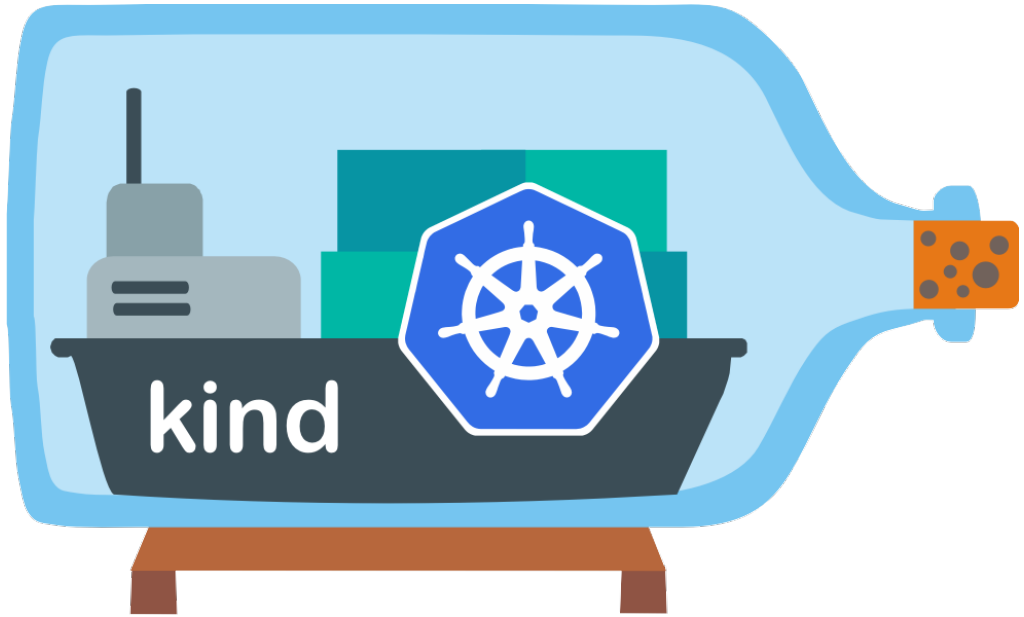
- Most of the time, we provide an object's definition in a **.yaml** file, which is converted by **kubectl** in a JSON payload and sent to the API Server.

Installing Kubernetes

- **In Cloud:** Many providers offer it as managed service, GKE, EKS, AKS…
- **On-Premise VMs:** Kubernetes can be installed on VMs created via Vagrant, VMware vSphere, KVM, etc.
- **On-Premise Bare Metal:** Kubernetes can be installed on on-premise Bare Metal, on top of different Operating Systems, like RHEL, CoreOS, CentOS, Fedora, Ubuntu, etc.
- **Installation tools:** Kubeadm, Kubespray, Kops, Ansible
- **Local installation:** Minikube, Kind (testing purposes)
- **Manual installation:** Kubernetes The Hard Way

Kind (Kubernetes IN Docker)

- We will use a **local single/multi-node installation based** on Kind
  - Pretty new way of installing k8s
    - 1° release 20191129
  - Why put k8s in Docker?
  - Docs: https://kind.sigs.k8s.io/
  - GitHub: https://github.com/kubernetes-sigs/kind
  - Quick Start: https://kind.sigs.k8s.io/docs/user/quick-start/



- Finger crossed and Hope everything works well :D

Exercise 18 - Prepare environment for k8s

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

Description: Prepare your environment to host a Kubernetes cluster based on Kind. Requirements are:

  - Install Kind
  - Download and setup `kubectl`
  - Install `kubectl` completion

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e18

Exercise 19 - Install a single-node Kubernetes cluster

- **Time:** ~15 minutes
  - 5 minutes: *Try by yourself*
  - 10 minutes: *Check, Verify, Ask*

Description: Create a single-node Kubernetes cluster and perform some interaction with the fresh new cluster.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e19

Recap of Kubernetes Object Model

- Kubernetes has a very rich (and now extensible) object model with which it represents different persistent entities in the cluster

- 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
kind: Pod
apiVersion: v1
metadata:
  name: clock
  namespace: default
spec:
  containers:
  - name: clock
    image: jpetazzo/clock
```

Single Pod clock: https://bit.ly/2SgVQpu

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`

Movie Time

The Illustrated Children's Guide to Kubernetes

https://www.youtube.com/watch?v=IcygvgW6sFM

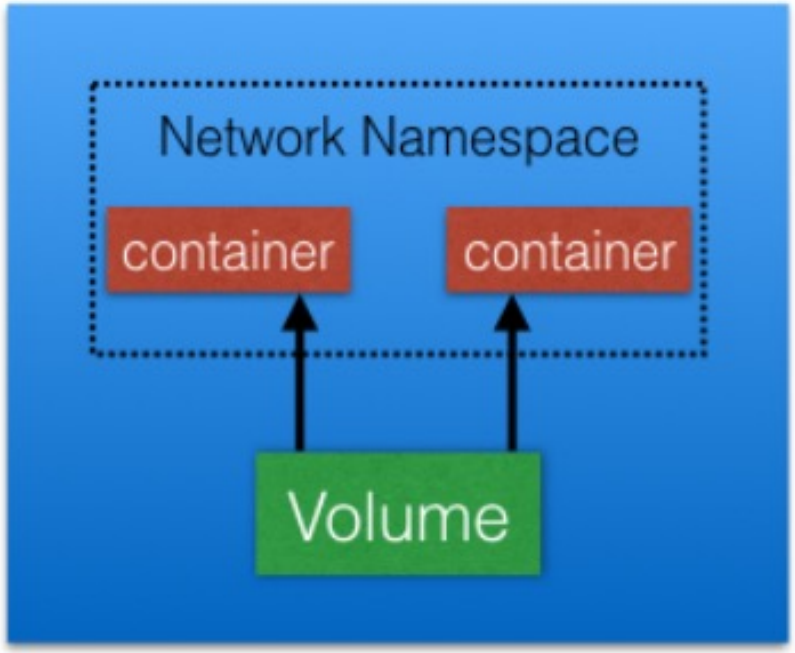
https://www.cncf.io/the-childrens-illustrated-guide-to-kubernetes

Exercise 20 - Play with our fresh new k8s cluster

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

Description: Play with the fresh new Kubernetes cluster and try to answer the questions. We use `kubectl` CLI to operate on the cluster. See the `kubectl` CLI overview here:
https://kubernetes.io/docs/reference/kubectl/overview

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e20



Pod

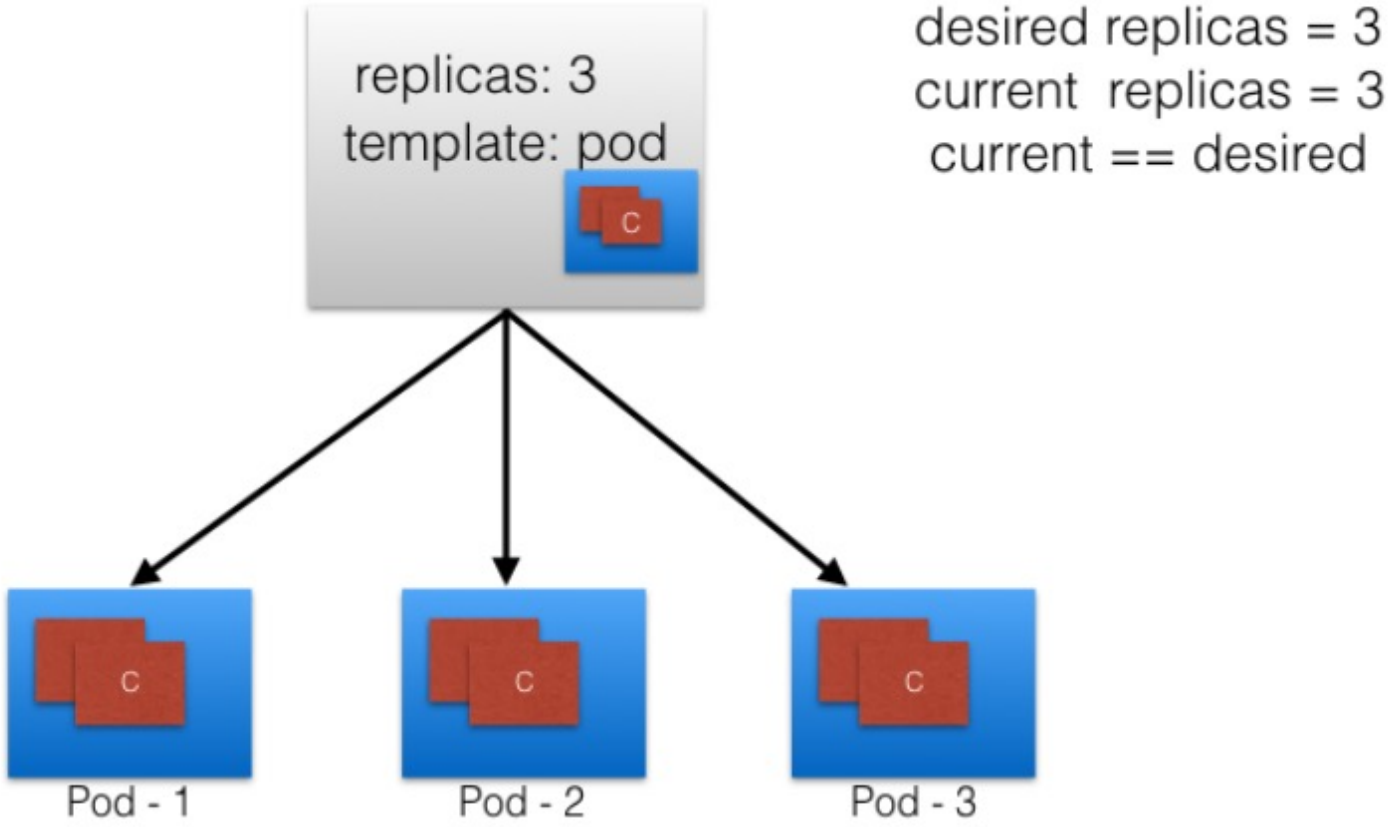
- The **smallest and simplest Kubernetes object**.
- It is the unit of deployment in Kubernetes, which represents a single instance of the application (microservice).
- Is a logical collection of one or more containers, which:
  - **Are scheduled together on the same host**
  - **Share the same network 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

Exercise 21 - Start a single pod using a spec file

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

Description: Analyze the manifest file describing the Pod resource. Use it to create an object on your Kubernetes cluster. Try to scale object just created and manage objects that are running in your cluster.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e21



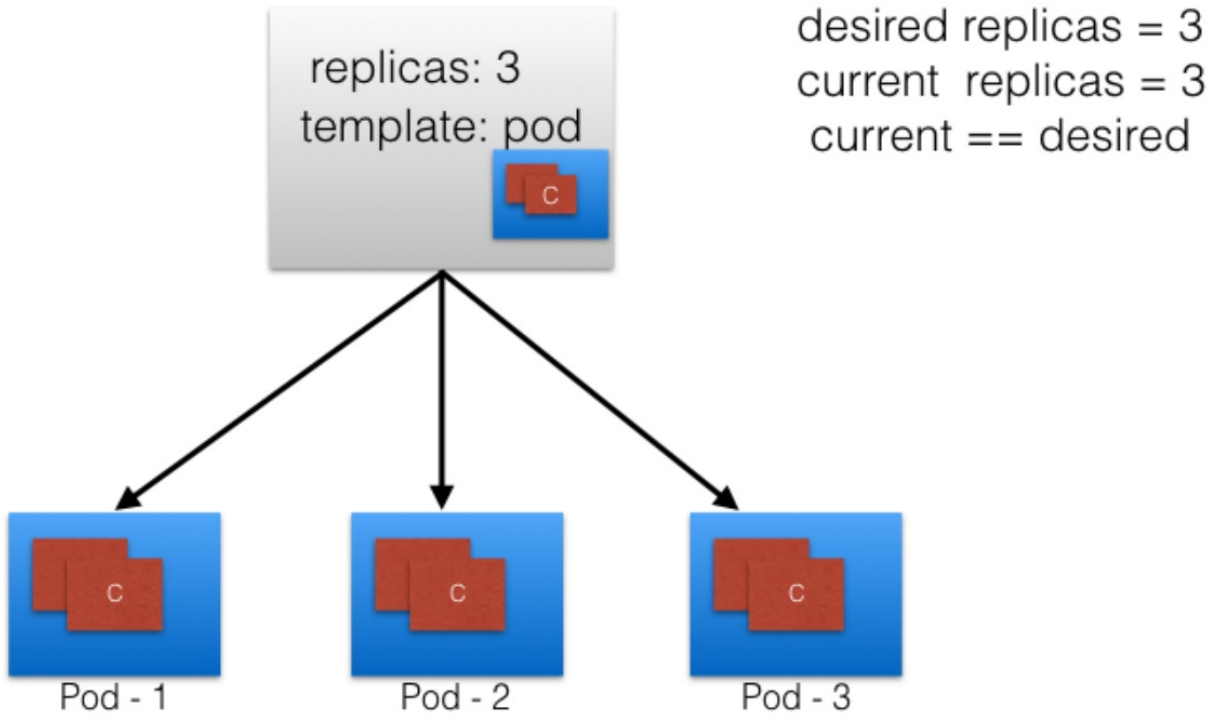
ReplicaSets

- 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.

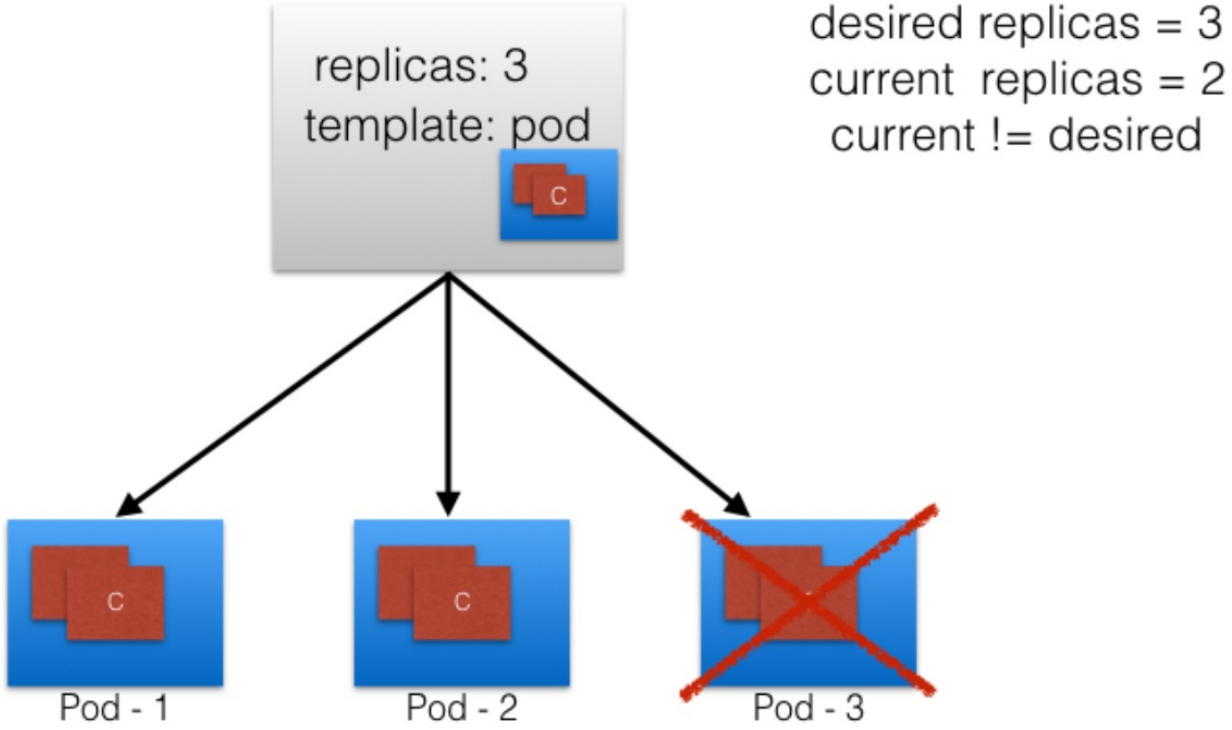
Lifecycle of a ReplicaSet 1/3

- We set the replica count to 3 Pods in the ReplicaSet **spec** section under the attribute `replicas`



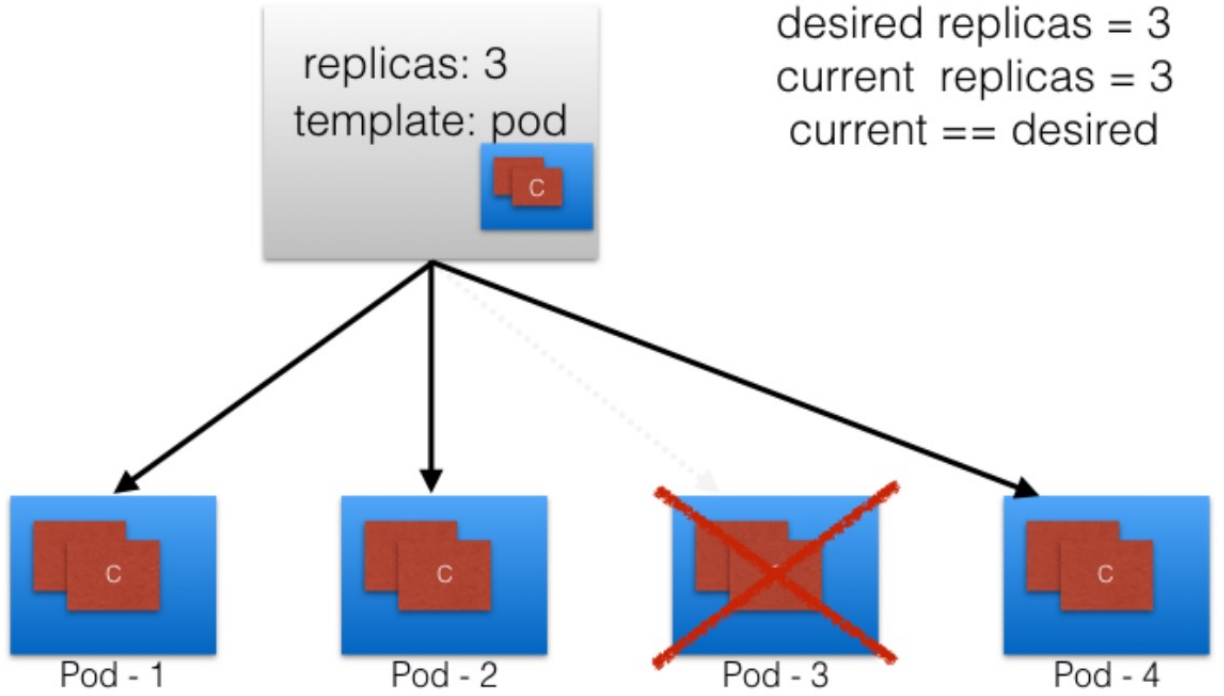
Lifecycle of a ReplicaSet 2/3

- If one Pod dies, our **current state** is not matching the **desired state** anymore.



Lifecycle of a ReplicaSet 3/3

- The ReplicaSet detects that the current state is no longer matching the desired state.
- So, it will create one more Pod, thus ensuring that the current state matches the desired state.
- We can use ReplicaSet independently but most of the time we use Deployments to manage the Pods: creation, deletion and updates.



Exercise 22 – Inspect the ReplicaSet

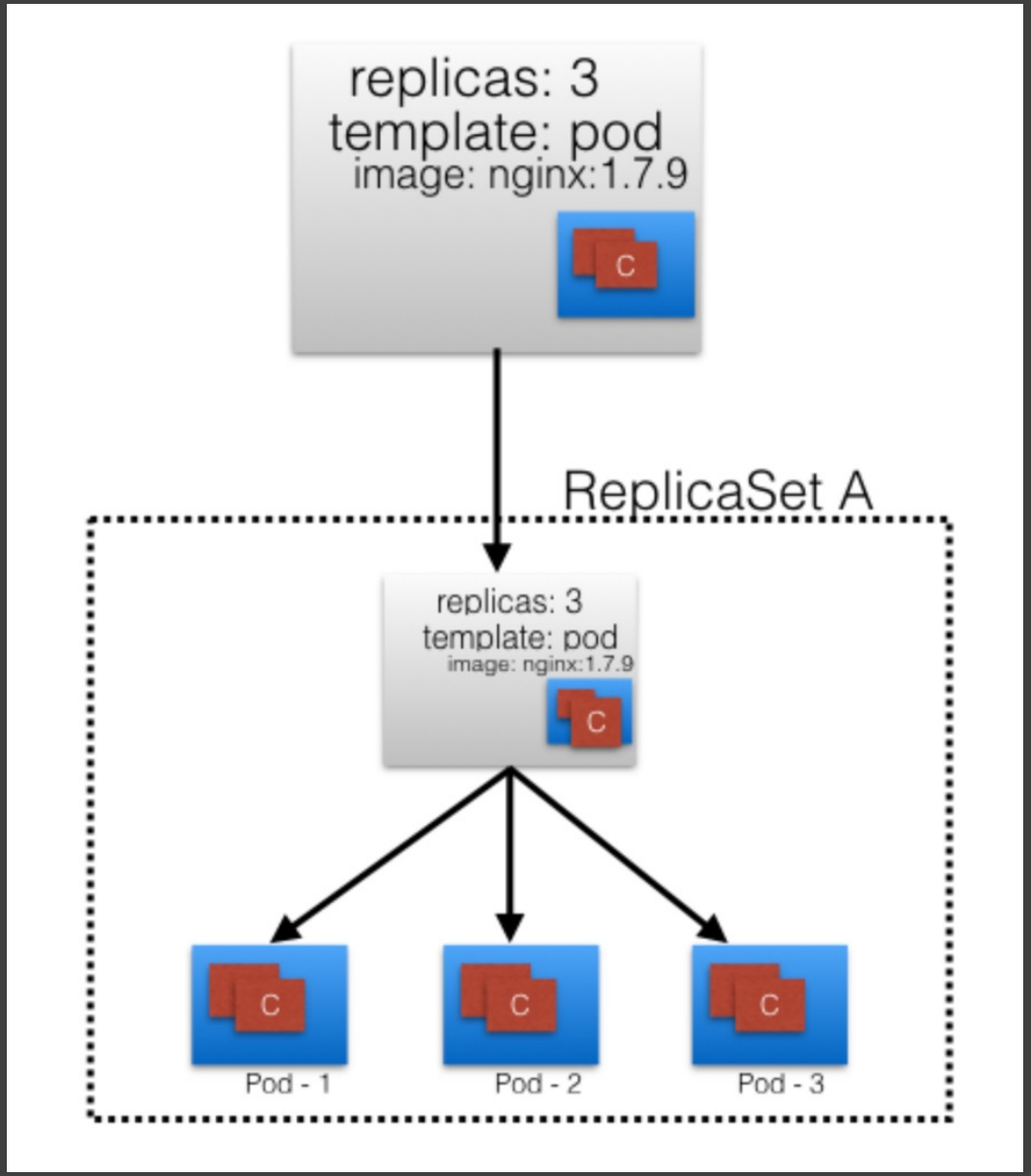
- **Time:** ~10 minutes
  - 5 minutes: *Try by yourself*
  - 5 minutes: *Check, Verify, Ask*

Description: Inspect resources of your cluster, try to find relations. Learn how to inspect `spec` and `status` of your manifests and to filter information using `JSON` or `YAML` code processors.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e22

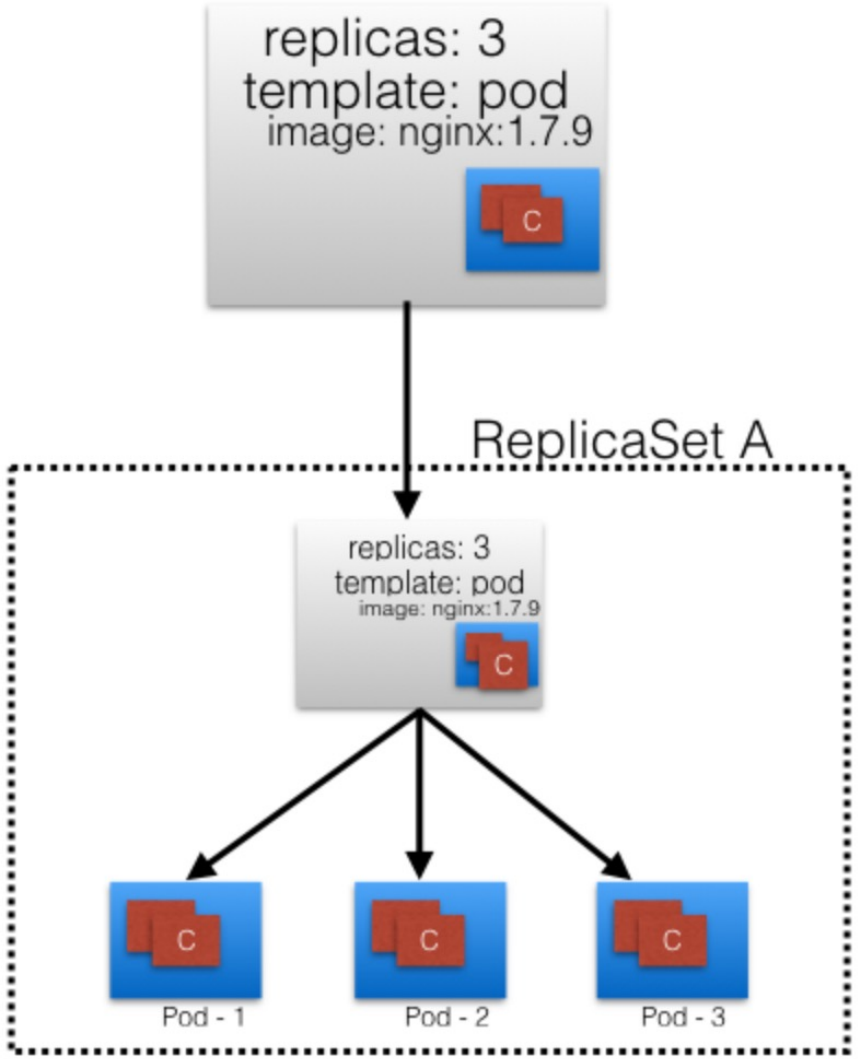
Deployments

- 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.



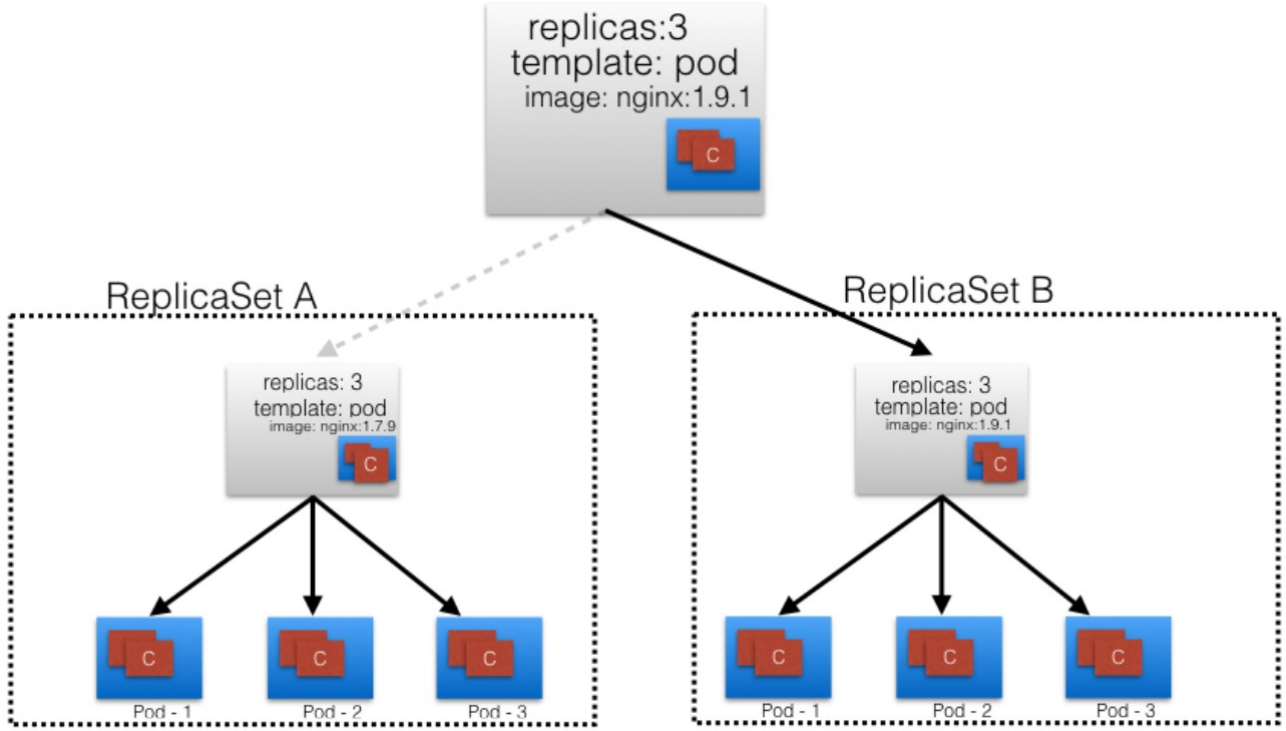
Lifecycle of a Deploymen 1/3

- One Deployment creates a ReplicaSet A
- ReplicaSet A creates 3 Pods
- Each Pod uses nginx:1.7.9 as container image



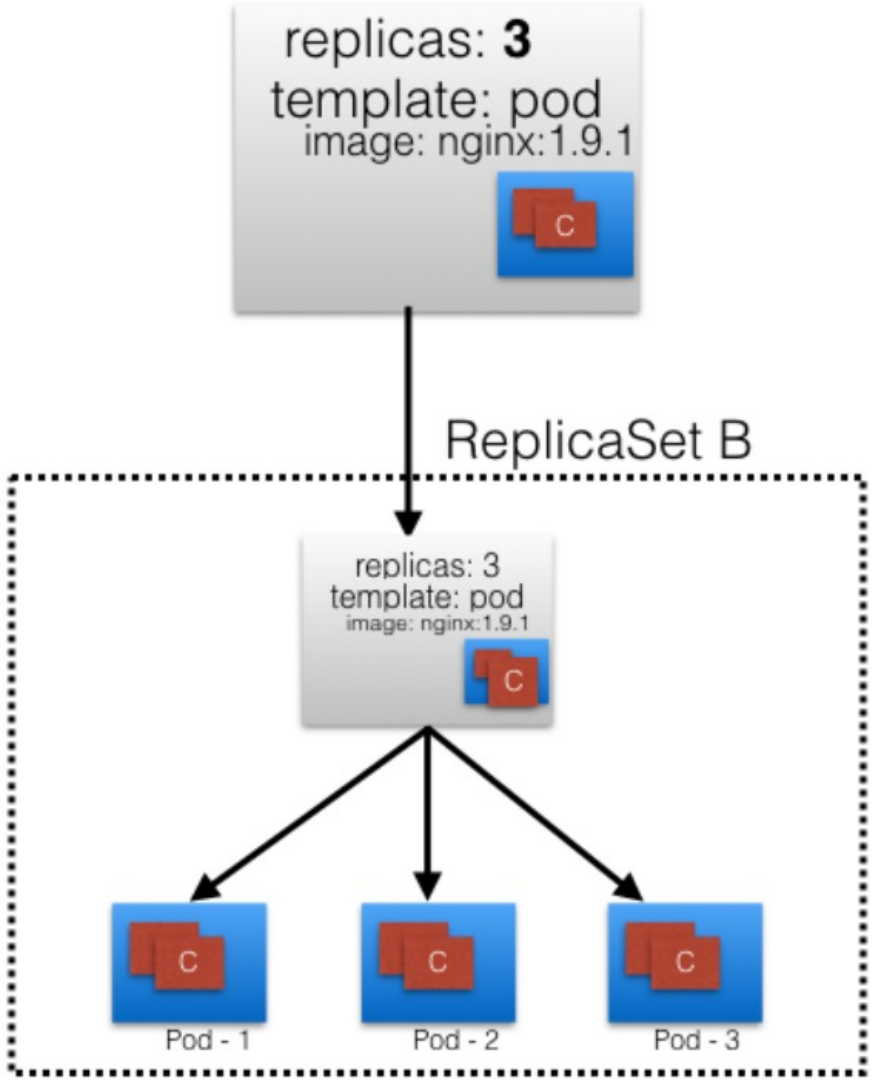
Lifecycle of a Deploymen 2/3

- The Pod template section of the Deployment is changed
- The nginx container image is renamed from nginx:1.7.9 to nginx:1.9.1
- As Deployment is modified a new ReplicaSet B is created



Lifecycle of a Deploymen 3/3

- Deployment now point to ReplicaSet B
- By means of ReplicaSets, Deployments provide features like Deployment recording
- If something goes wrong, we can rollback to a previously known state.



Exercise 23 – Deployment rollout

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

Description: Change the image of an existing Deployment and see how the Rolling Update feature comes into play. Inspect the log of the new application version and ensure that the update has been done. finally restore the original version of your application (Deployment).

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e23

LinuxApp Summit – April 29,30 Rovereto