Kubernetes in pillole

Alessandro Costantini

Installazione e amministrazione Kubernetes

9, 10, 11 Dicembre 2024

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Ministero dell'Università e della Ricerca

Italiadomani

PNC
Piano nazionale per gli investimenti complementari al PNRR
Ministero dell'Università e della Ricerca

Overview

- Container Orchestration
- Kubernetes components
- Kubernetes fundamentals
- Kubernetes deployment steps

Container orcehstration

Docker containers, microservices…

- **Let's recap things.**
- **Docker containers** help to easily create and share applications that are – as the name says – self-contained.
- On the other hand, we just saw that **microservice architectures** are based on the composition of many independent (but communicating) services. And that through some processes and tools such as DevOps, we can **write microservice-based applications** that are scalable, reliable and maintainable.
- **Combining these two points**, *containers* can greatly help with the creation of a *microservice architecture*.

… and orchestration

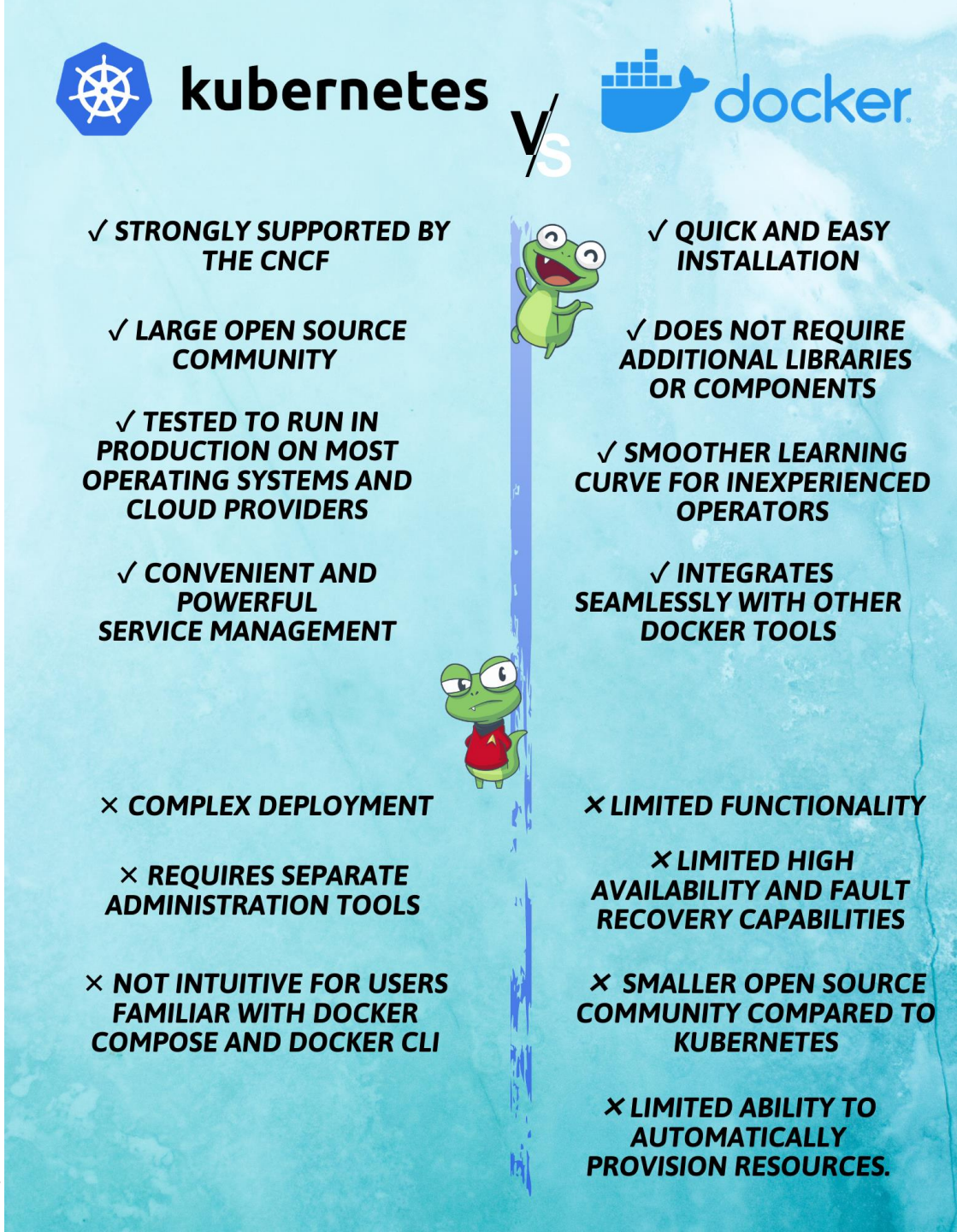
- We therefore need to understand how to effectively *orchestrate* many containers across **multiple, distributed hosts**. This is called **container orchestration**.

Docker Swarm and Kubernetes

- **Docker Swarm** is a simple way of orchestrating containers with Docker. Some of its main features:
  - **It is integrated with the Docker Engine**: no other software than Docker is needed.
  - **It has a decentralized design**: this means that any node in a Docker Swarm can assume any role (master, slave) at runtime.
  - It uses Docker's **overlay networks**.
- Another very popular container orchestration engine is **Kubernetes**. This is the tool we will now turn our attention to, first with theory and then with a hands-on exercise. Later, we will do the same hands-on to compare Kubernetes and Swarm.

Swarm vs. Kubernetes

- You can find online plenty of comparisons between the two. See for example this picture, taken from https://sensu.io/blog/kubernetes-vs-docker-swarm.
- CNCF: Cloud Native Computing Foundation https://www.cncf.io/



Kubernetes

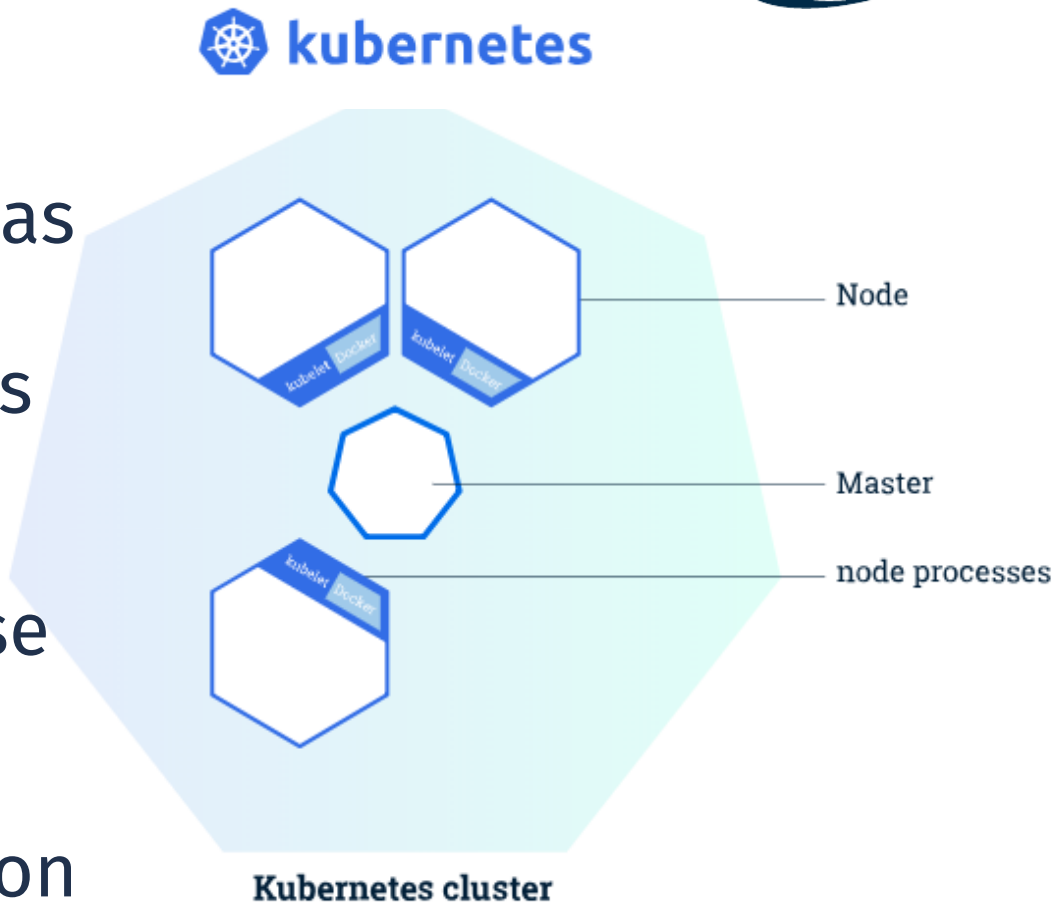
- **Kubernetes**, or k8s, is probably the most popular container orchestration toolset in use today (https://kubernetes.io/).
- Kubernetes [*] was initially developed at Google to scale container applications over a Google-scale infrastructure.
- Before coming to Kubernetes hands-on exercises, we need to shortly describe its main concepts.

[*] Kubernetes: κυβερνήτης, Greek for “helmsman” or “pilot” or “governor” (https://en.wikipedia.org/wiki/Kubernetes)

Kubernetes components

Kubernetes

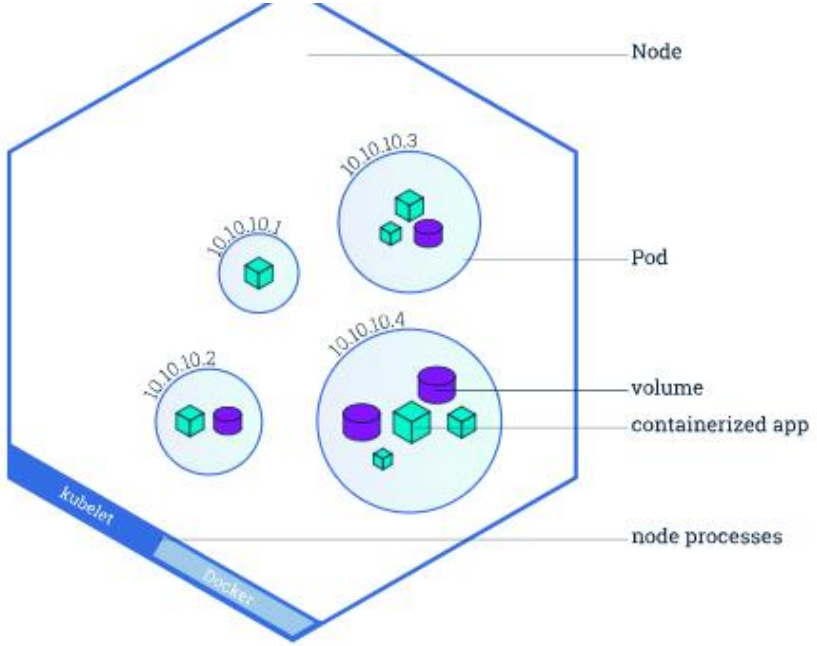
- Kubernetes coordinates a *cluster of computers* that are connected to work as a single unit.
- Applications that can run in Kubernetes cluster have to be **containerized**.
- Kubernetes then efficiently automates the distribution and scheduling of these containerized applications across the cluster.
- A Kubernetes cluster can be deployed on either physical or virtual machines.

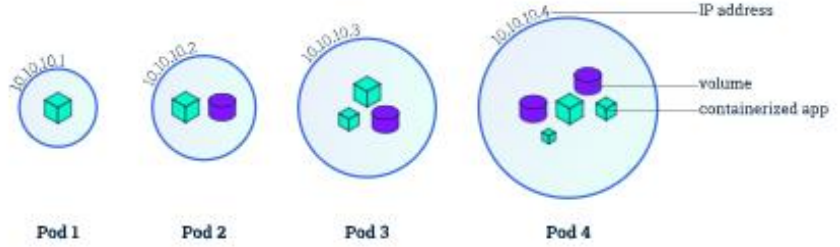


https://kubernetes.io

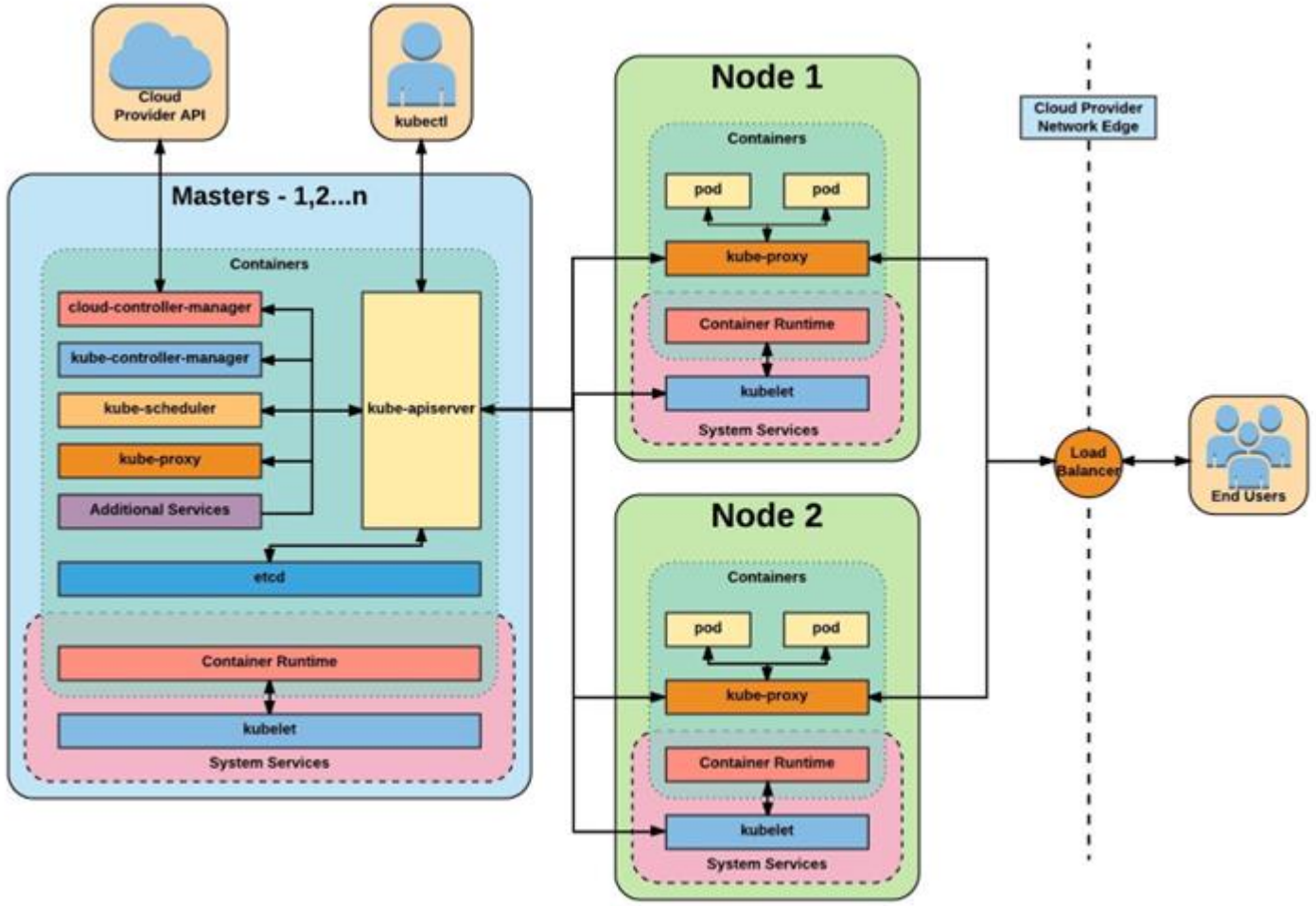
Kubernetes clusters

- A Kubernetes *cluster* consists of two types of resources:
  - One or more **Masters** coordinate the cluster.
  - **Nodes** are the workers that run containerized applications.
- The **Master** is responsible for managing the cluster.
  - It coordinates all activities in the 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 runs containerized applications by special processes called **pods**.

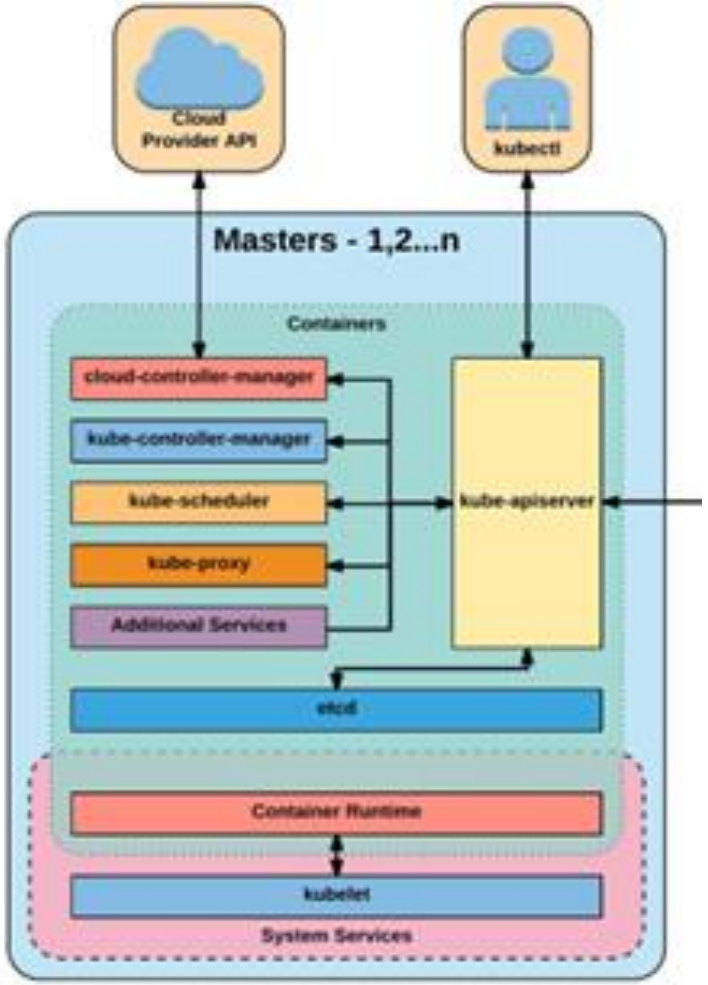




Kubernetes clusters

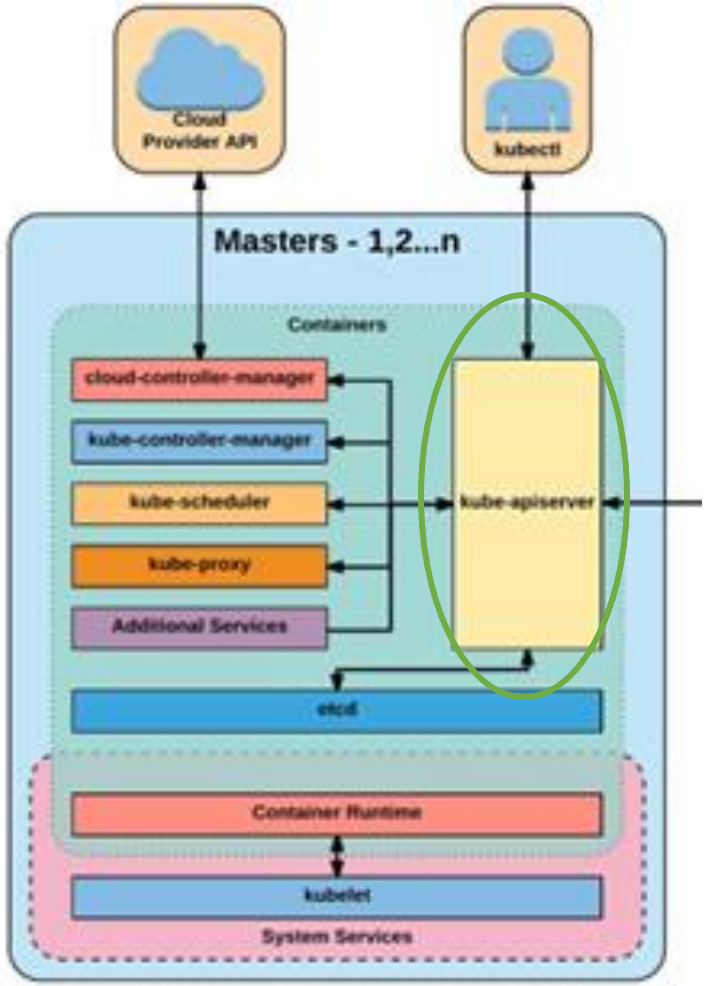


Kubernetes clusters



- Kube-apiserver
- Etcd
- Kube-controller-manager
- Cloud-controller-manager
- Kube-scheduler

Kubernetes clusters

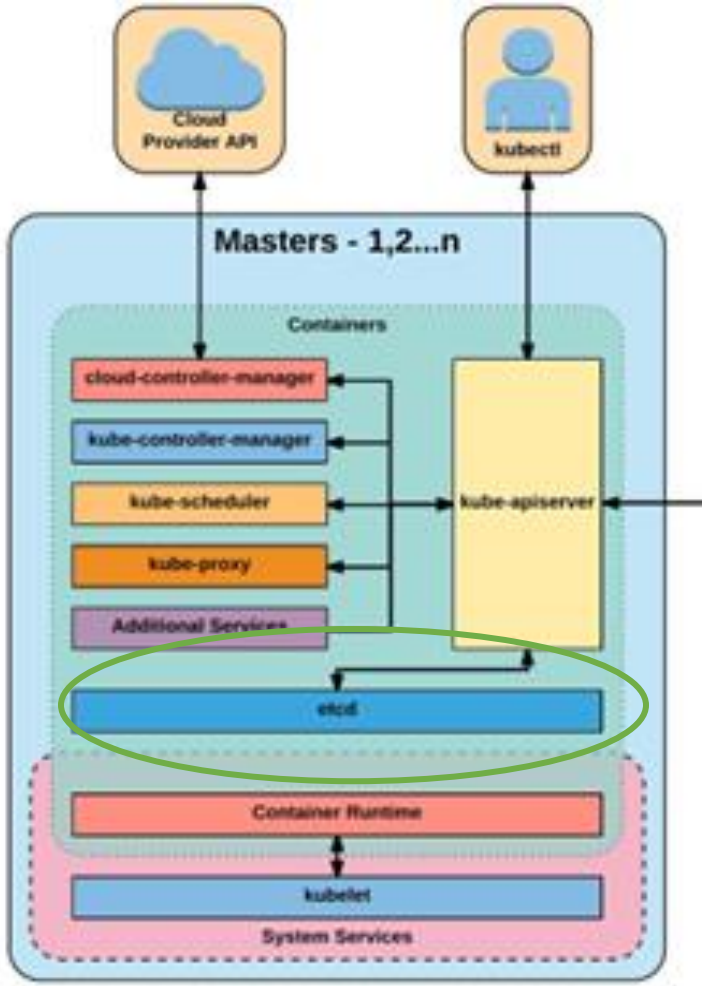


Kube-apiserver

The apiserver provides a forward facing REST interface into the kubernetes control plane and datastore.
All clients, including nodes, users and other applications interact with kubernetes **strictly** through the API Server.

It is the true core of Kubernetes acting as the gatekeeper to the cluster by **handling authentication and authorization, request validation, mutation**, and admission control in addition to **being the front-end to the datastore.**

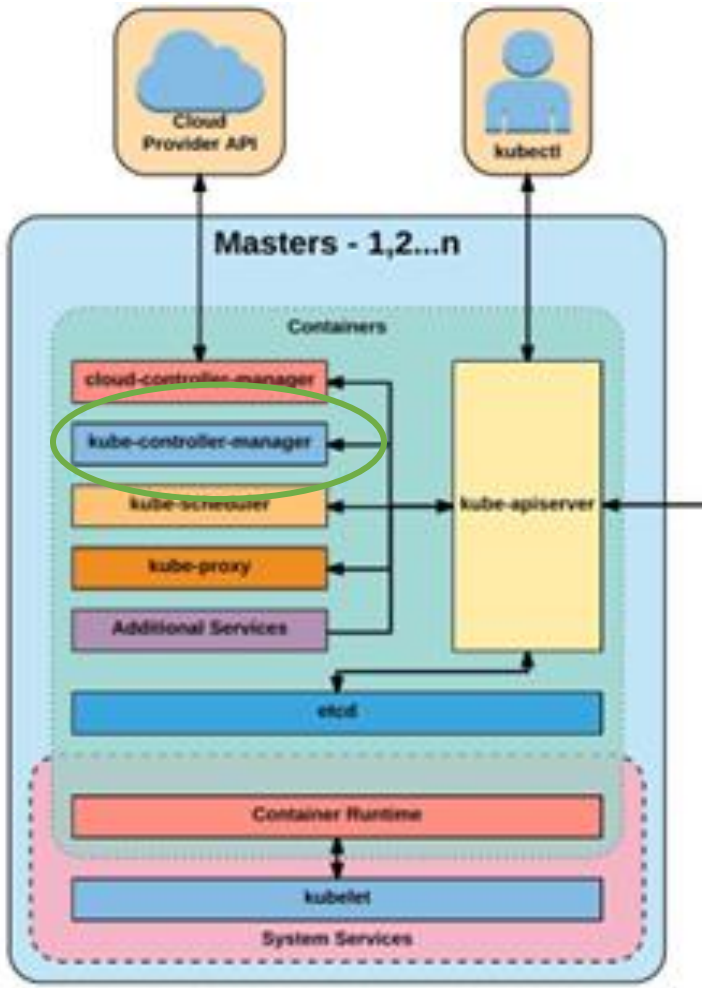
Kubernetes clusters



ETCD

Etcd acts as the cluster datastore; providing a strong, consistent and highly available key-value store used for persisting cluster state.

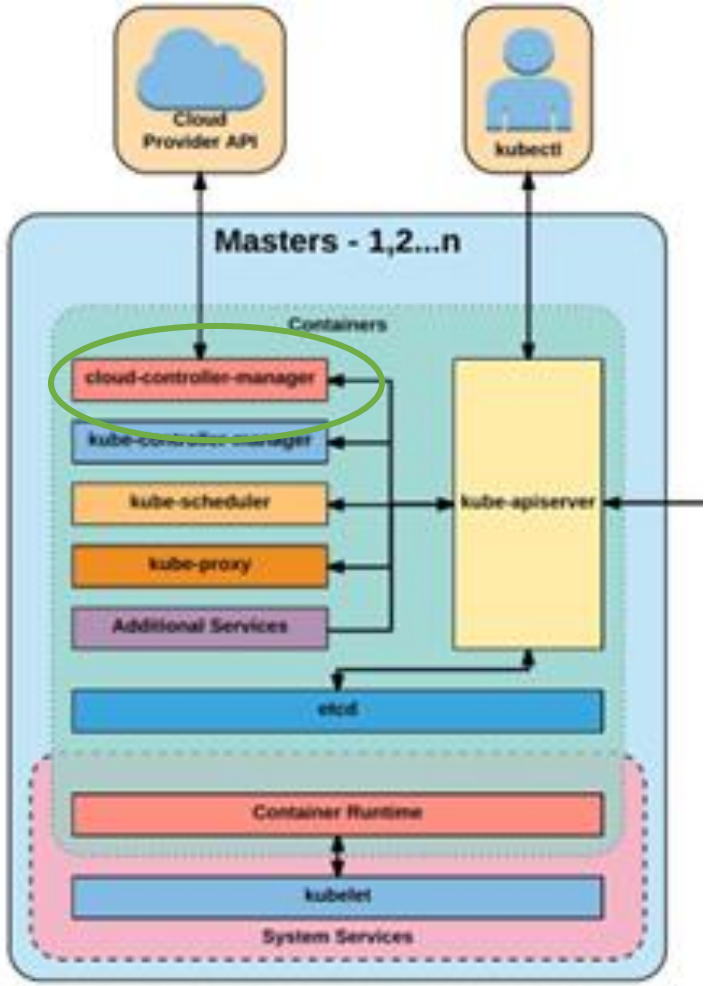
Kubernetes clusters



Kube-controller-manager

The **controller-manager** is the primary daemon that manages all core component control loops. It **monitors the cluster state** via the apiserver and steers the cluster towards the desired state.

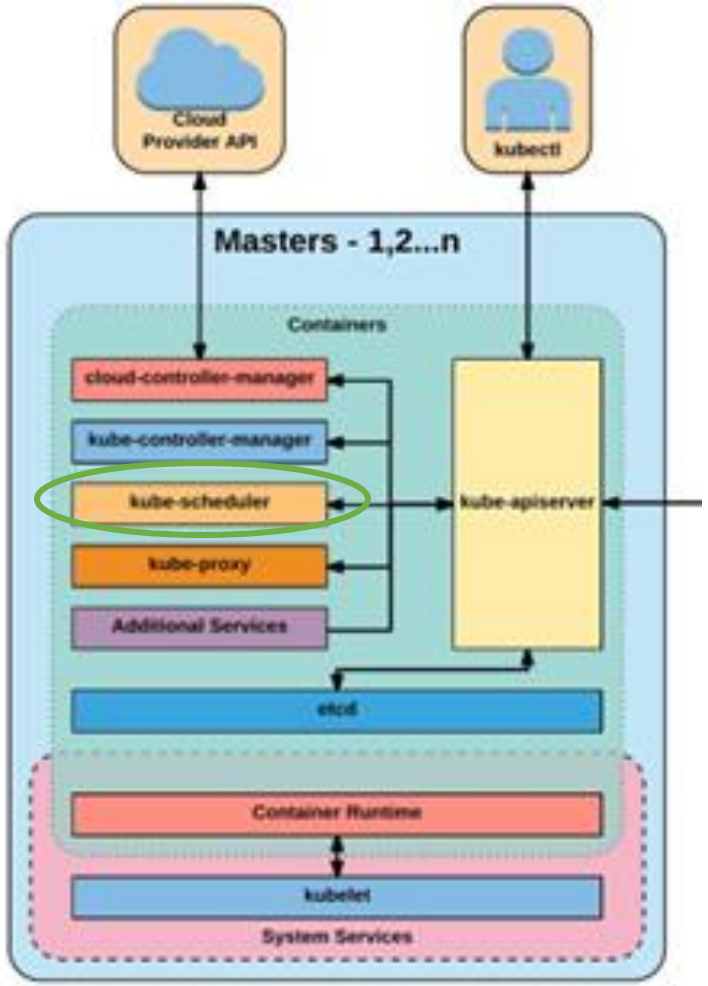
Kubernetes clusters



Cloud-controller-manager

The **cloud-controller-manager** is a daemon that provides **cloud-provider specific knowledge and integration capability** into the core control loop of Kubernetes. The controllers include Node, Route, Service, and add an additional controller to handle PersistentVolumeLabels

Kubernetes clusters

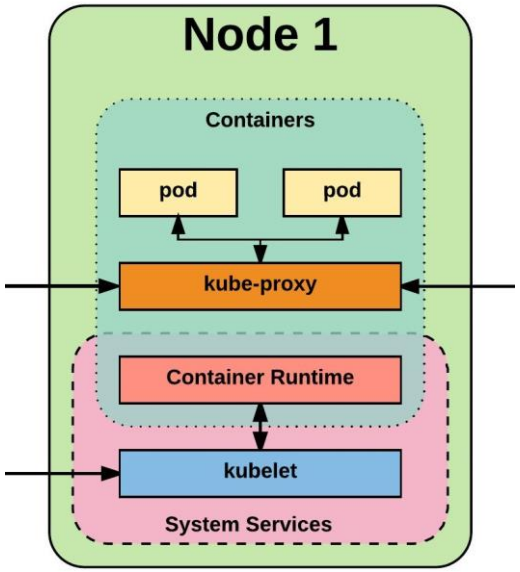


Kube-scheduler

Kube-scheduler is a verbose policy-rich engine that **evaluates workload requirements** and attempts to place it on a matching resource. These requirements can include such things as general hardware reqs, affinity, anti-affinity, and other custom resource requirements

Kubernetes clusters

- Kubelet
- Kube-proxy
- Container runtime engine

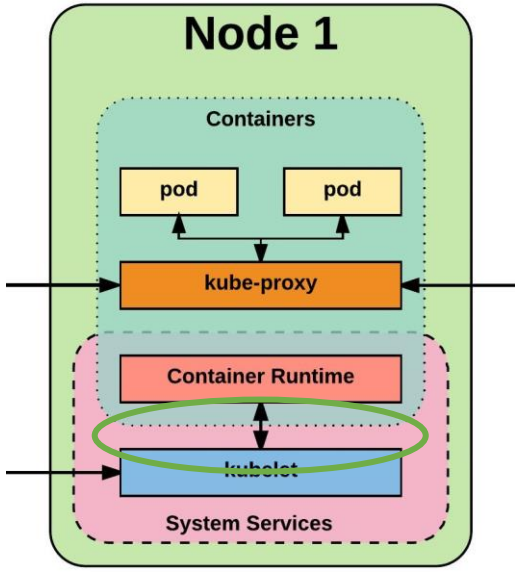


Kubernetes clusters

Kubelet

Acts as the **node agent** responsible for **managing pod lifecycle** on its host. Kubelet understands **YAML** container manifests that it can read from several sources:

- File path
- HTTP Endpoint
- Etcd watch acting on any changes
- HTTP Server mode accepting container manifests over a simple API.



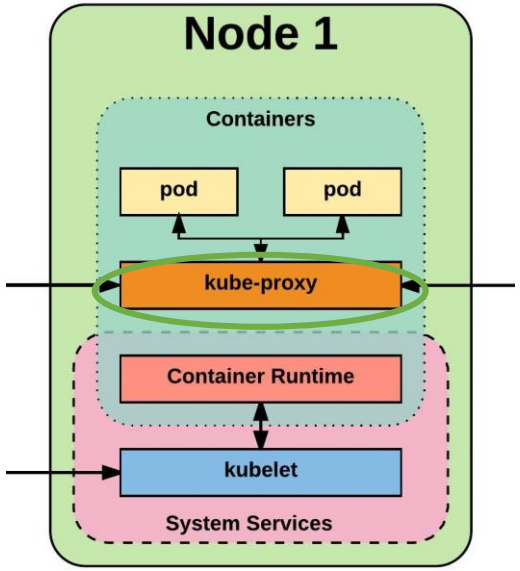
Kubernetes clusters

Kube-proxy

Manages the network rules on each node **and performs connection forwarding or load balancing** for Kubernetes cluster services.

Available Proxy Modes:

- Userspace
- iptables
- ipvs (alpha in 1.8)

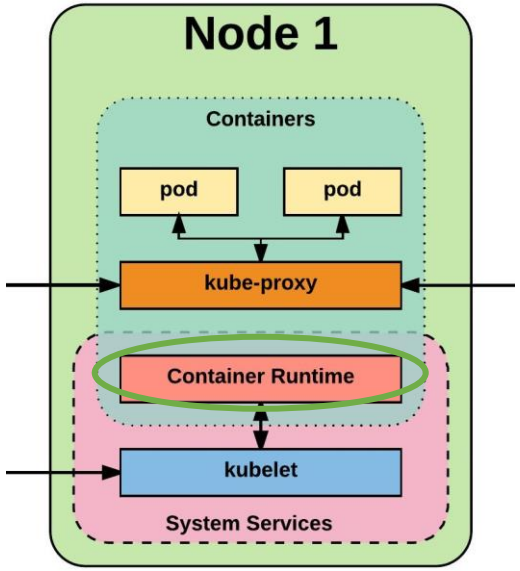


Kubernetes clusters

Container Runtime

With respect to Kubernetes, A container runtime is a **CRI** (Container Runtime Interface) compatible application that **executes and manages containers**.

- Containerd (docker)
- Cri-o
- Rkt
- Kata (formerly clear and hyper)
- Virtlet (VM CRI compatible runtime)



Kubernetes clusters

Additional services

Kube-dns - Provides cluster wide DNS Services. Services are resolvable to *<service>.<namespace>.svc.cluster.local*.

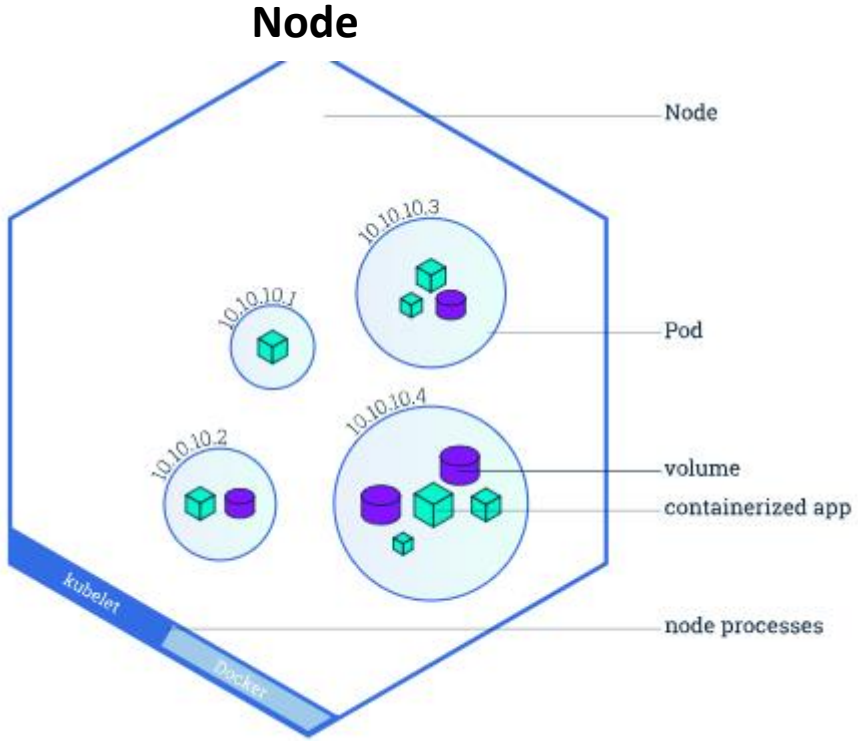
Heapster - Metrics Collector for kubernetes cluster, used by some resources such as the Horizontal Pod Autoscaler. (required for kubedashboard metrics)

Kube-dashboard - A general purpose web based UI for kubernetes.

Kubernetes fundamentals

Kubernetes Pods

- A **Pod** is the basic building block of Kubernetes. It represents a running process on your cluster.
- A Pod encapsulates:
  - application containers;
  - storage resources;
  - a unique IP address;
  - options that govern how the container(s) sh



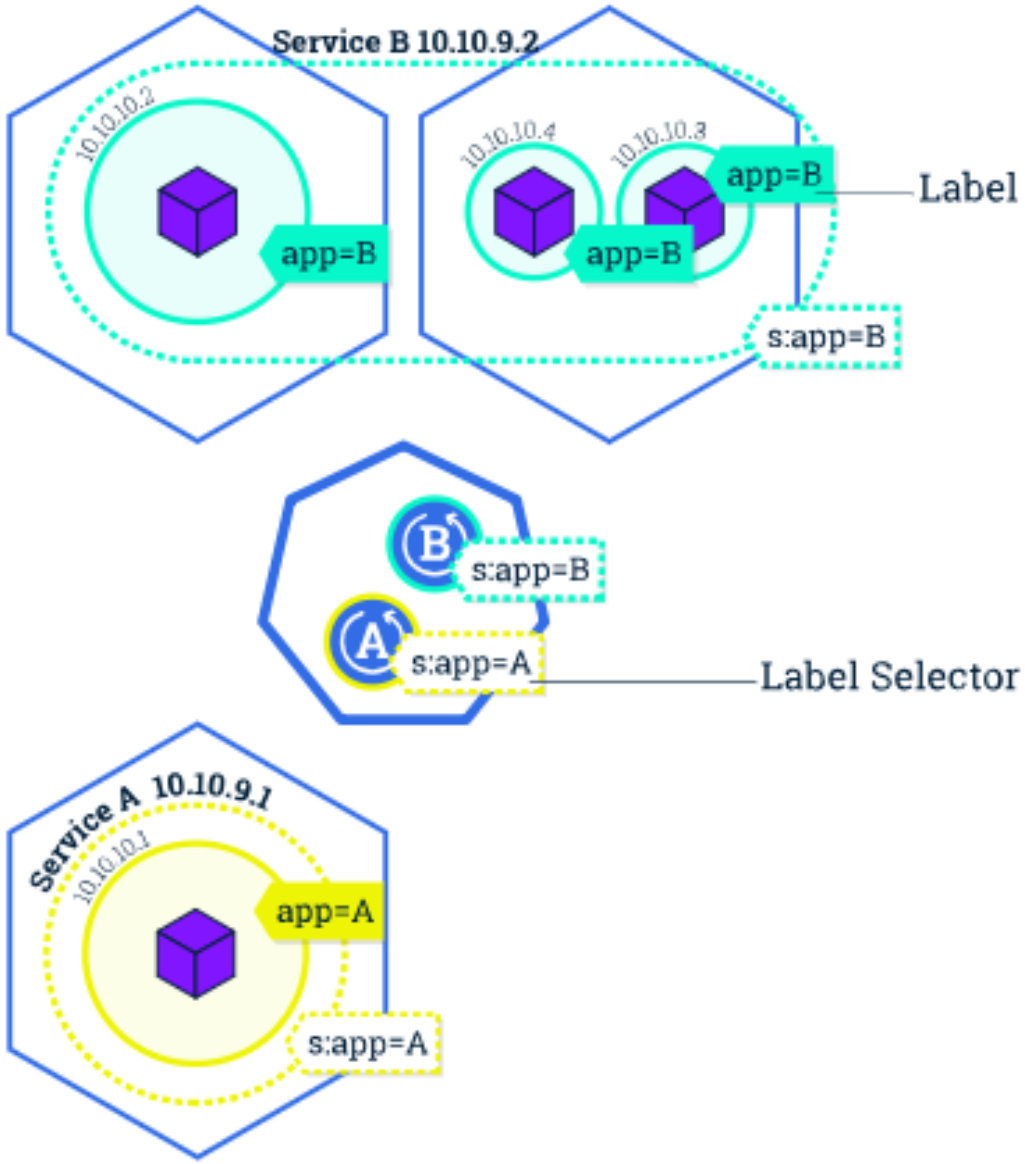
Kinds of Pods

- We may have *two kinds of Pods*:
  - **Pods running 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. Kubernetes manages the Pods rather than the containers directly.
  - **Pods running 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.

Pod examples

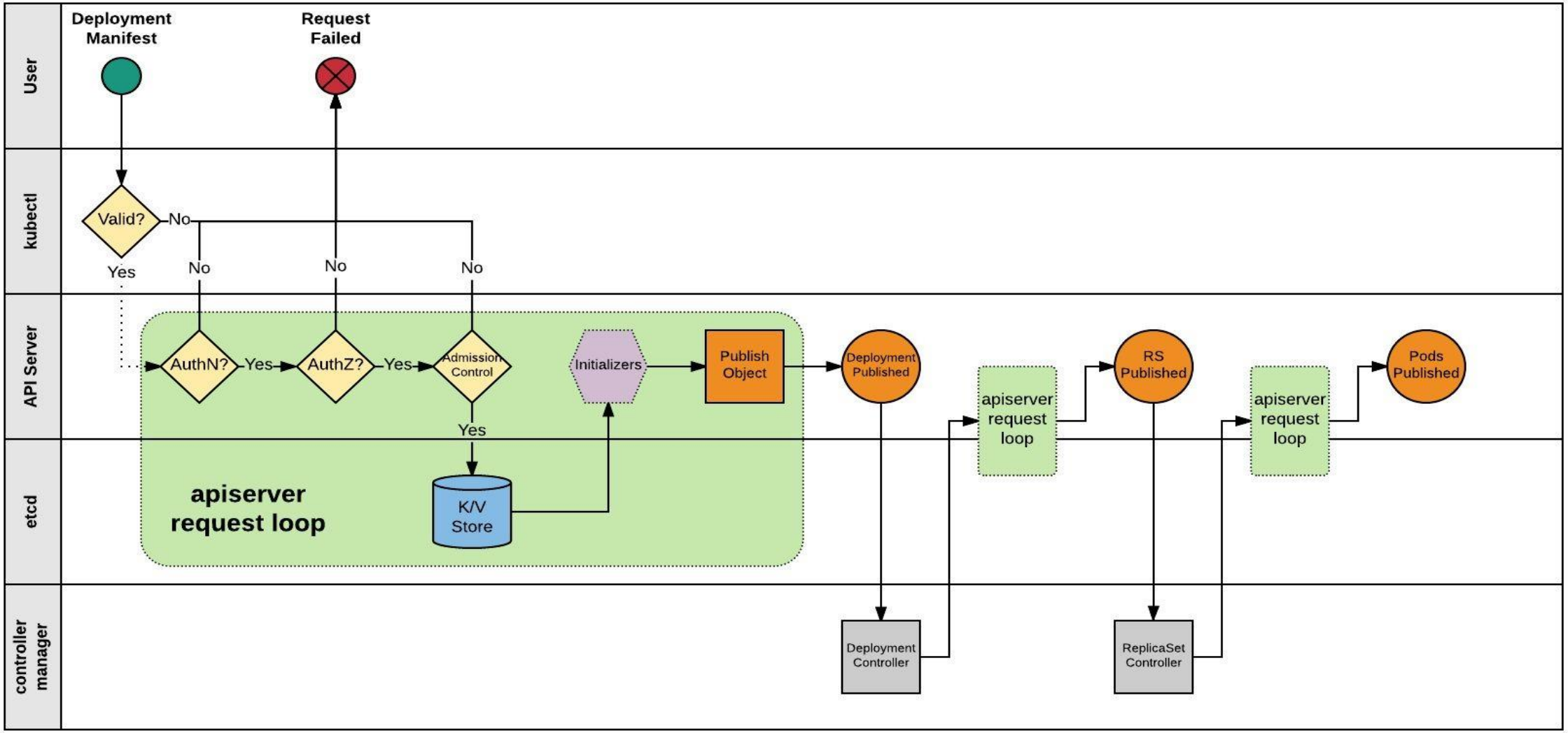
Kubernetes Services

- 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, these IPs are not exposed outside the cluster without a Service. Therefore, **you need Services to allow your applications to receive traffic**.
- Services match a set of Pods using [labels and selectors], allowing to group and operate on objects in Kubernetes. Labels are key/value pairs attached to objects and can be used in multiple ways. For instance:
  - To designate objects for development, test, and production
  - To embed version tags
  - To classify objects using tags



Kubernetes deployment steps

Kubernetes deployment



Kubernetes deployment