Kubernetes in pillole

Alessandro Costantini

Installazione e amministrazione Kubernetes

1, 2, 3 Aprile 2025

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

Ministero dell'Università e della Ricerca | Italiadomani Piano Nazionale di Ripresa e Resilienza | PNC Piano nazionale per gli investimenti complementari al PNRR Ministero dell'Università e della Ricerca

Overview

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

Container orcehstration

Microservices…

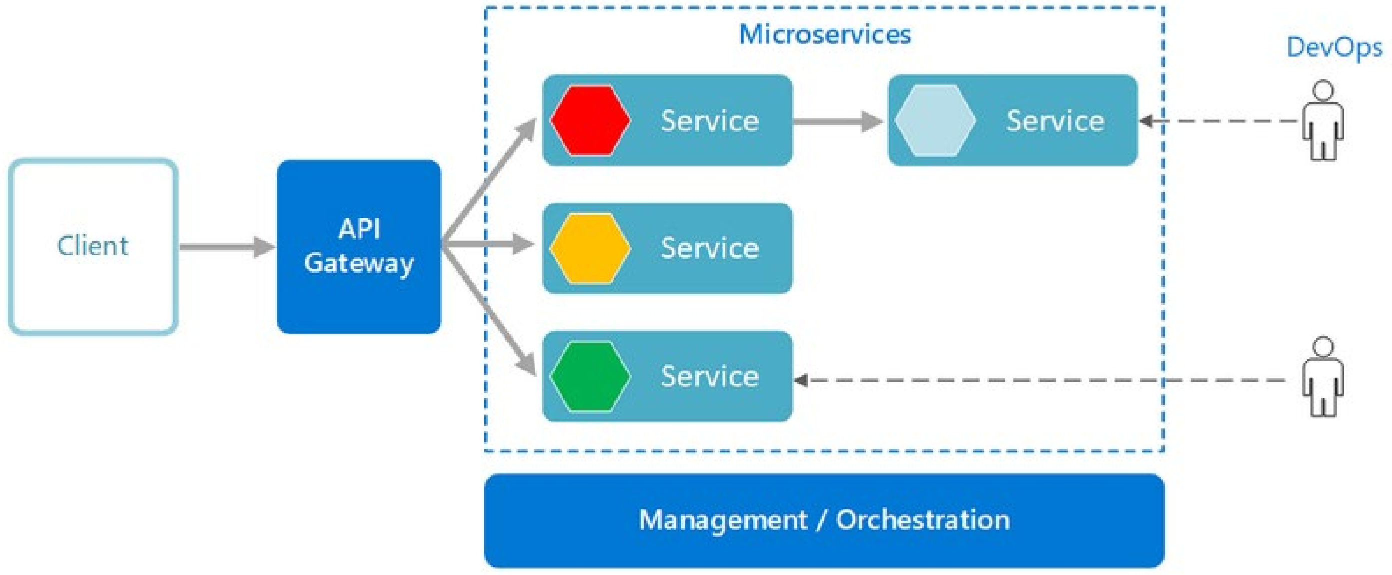
- Robert C. Martin coined the term single responsibility principle which states “gather together those things that change for the same reason, and separate those things that change for different reasons.” https://it.wikipedia.org/wiki/Robert_Cecil_Martin

Loosely coupled services which can be developed, deployed, and maintained independently. Each of these services is responsible for discrete task and can communicate with other services through simple APIs to solve a larger complex business problem.

- **Suggested reading - The What, Why, and How of a Microservices Architecture**
  - https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9

Microservices: architecture

- A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability within a bounded context. A bounded context is a natural division within a business and provides an explicit boundary within which a domain model exists.



Microservices: advantages

- Microservices are small, independent, and loosely coupled. A single small team of developers can write and maintain a service.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Services don't need to share the same technology stack, libraries, or frameworks.

Docker containers, microservices…

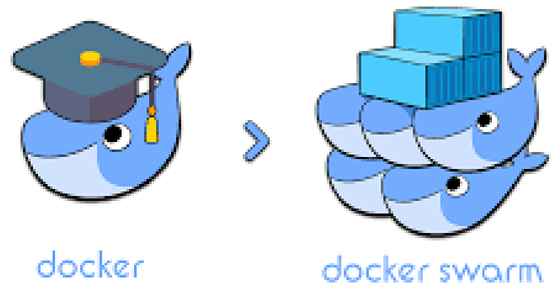
- **Let’s recap things.**
- **Docker containers** help to easily create and share applications that are – as the name says – self-contained.
- On the other hand, we just saw that **microservice architectures** are based on the composition of many independent (but communicating) services. And that through 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

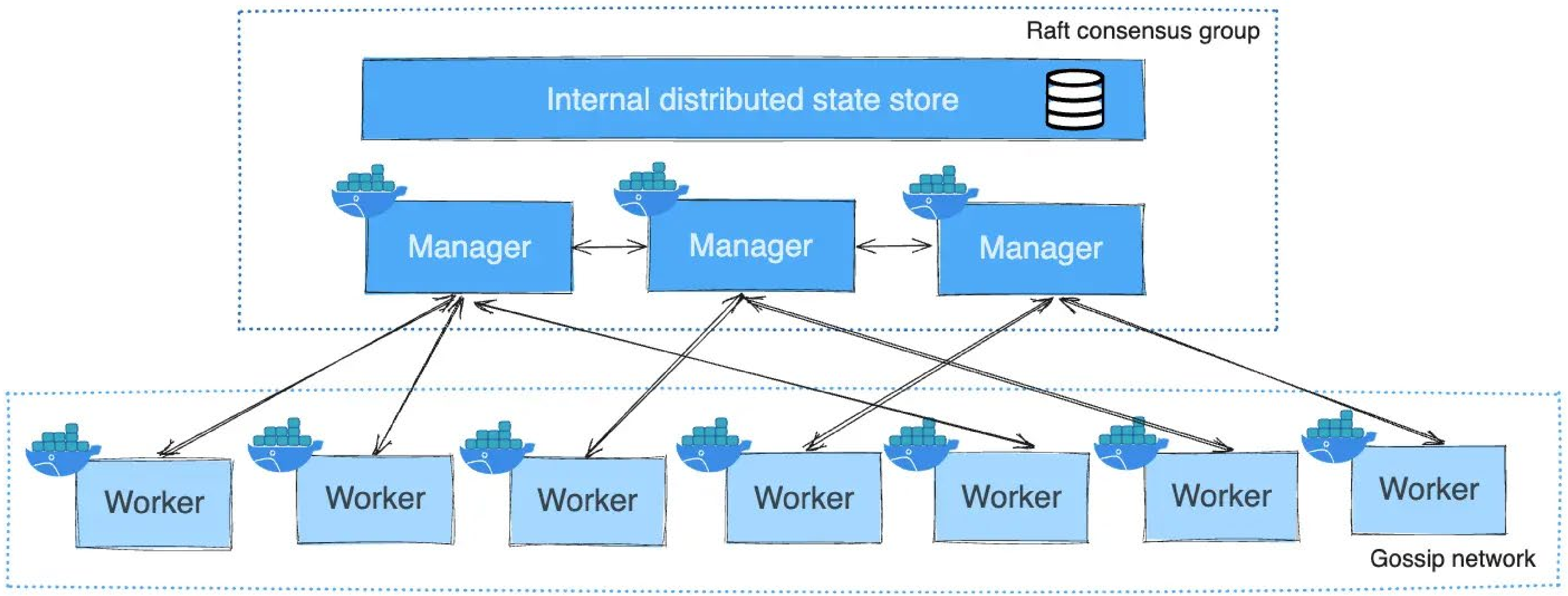
- We therefore need to understand how to effectively *orchestrate* many containers across **multiple, distributed hosts**. This is called **container orchestration**.

Docker Swarm

- **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**.



Docker Swarm architecture



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 vs Docker Swarm

Installation and Setup

Installation and setup are complex, requiring a deep understanding of its components and configuration options to deploy a cluster securely and efficiently

Installation process is straightforward. It seamlessly integrates into existing Docker environments with minimal configuration, making it ideal for teams seeking quick deployment.

Kubernetes vs Docker Swarm

Deployment

Provides a highly configurable environment supporting various workloads, including stateless, stateful, and batch processes. It offers detailed control over how applications are deployed and scaled, enabling precise management of containerized applications across clusters.

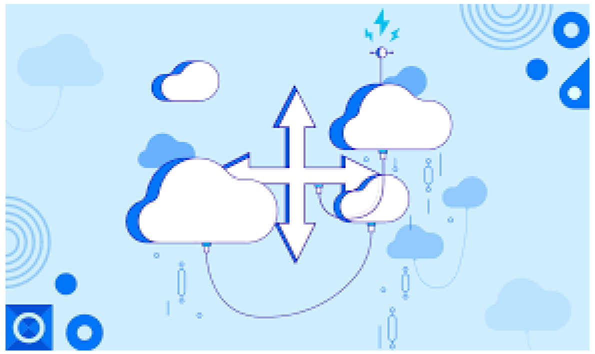
Offers a more streamlined deployment process with fewer configuration options. Its approach is particularly suited to straightforward applications, providing sufficient control for basic scaling and management without the overhead and complexity of Kubernetes.

Kubernetes vs Docker Swarm

Autoscaling

Supports autoscaling, allowing applications to dynamically adjust their size based on performance metrics and predefined policies. This feature ensures efficient utilization of resources and optimal application performance under varying loads.

Offers basic scaling capabilities, Docker Swarm lacks the same sophisticated autoscaling mechanisms in Kubernetes. Its simpler model often requires manual scaling decisions, potentially leading to over-provisioning or under-utilizing resources.



Kubernetes vs Docker Swarm

Storage

Offers advanced storage capabilities, supporting a range of storage backends and configurations. This allows for persistent storage, which is essential for stateful applications, and has high flexibility and control.

Storage options are simpler, emphasizing ease of use but offering fewer configurations and integrations. While sufficient for many use cases, Swarm may not cater to complex, stateful applications requiring intricate storage setups.

Kubernetes vs Docker Swarm

Security

Provides comprehensive security features, including role-based access control (RBAC), secrets management, and network policies, enabling fine-grained security configurations tailored to specific application and organization requirements.

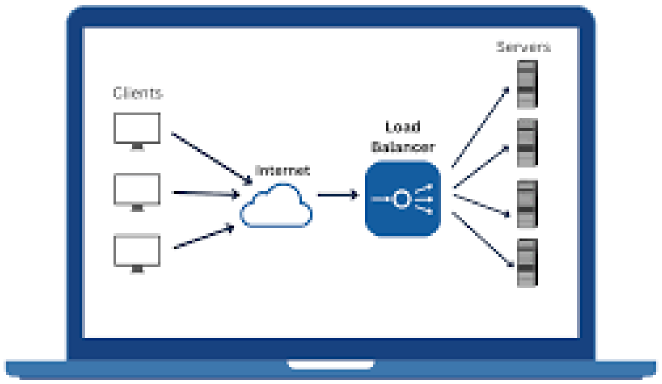
Offers basic security features, Docker Swarm lacks the depth and flexibility of Kubernetes' security model. Its more straightforward approach may suffice for less complex environments but might not meet the stringent security requirements of larger, more complex deployments.

Kubernetes vs Docker Swarm

Load Balancing

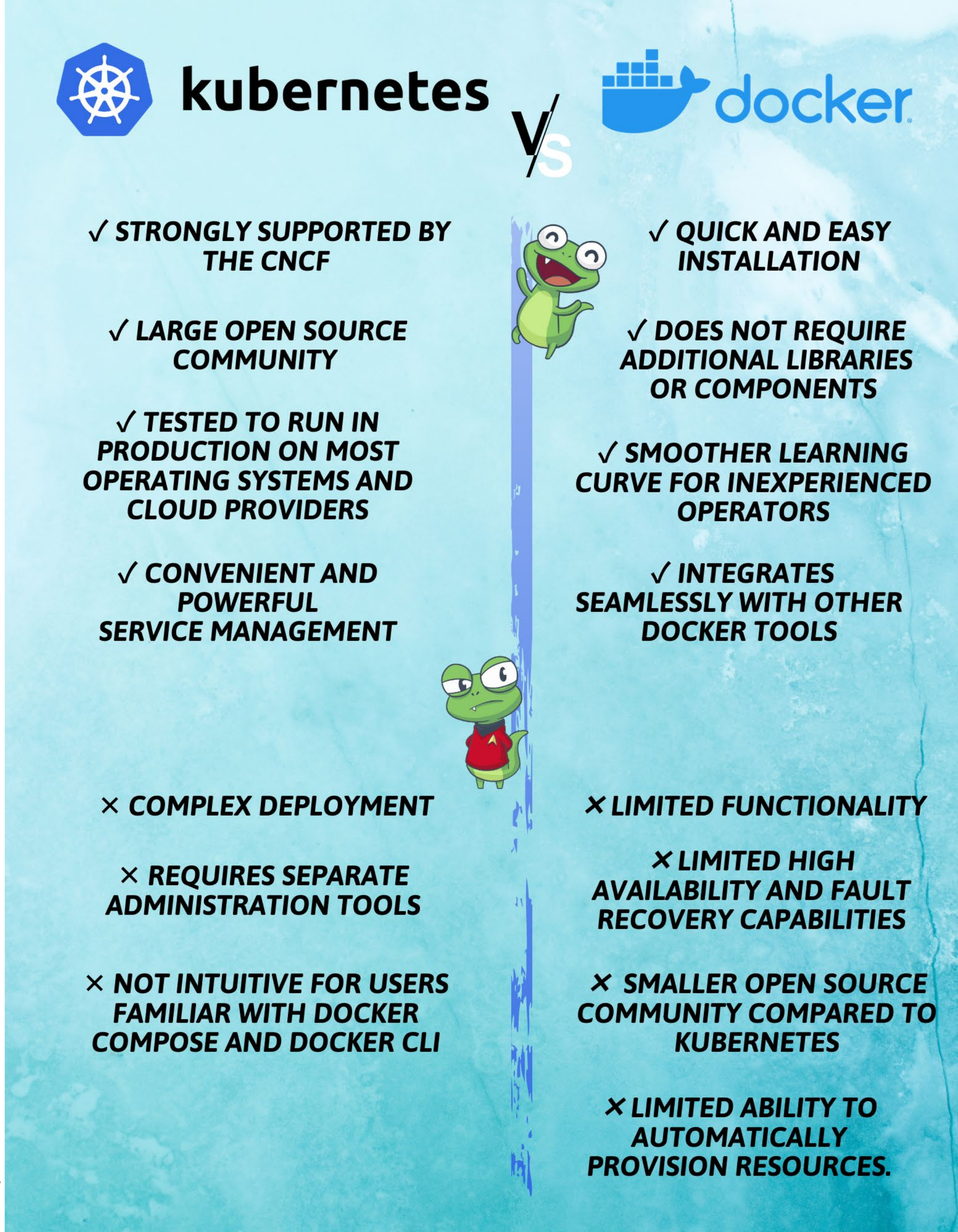
Is highly configurable. It supports both internal and external traffic with advanced routing capabilities. This allows for efficient traffic distribution across services, enhancing application performance and reliability.

Provides simpler, effective load-balancing mechanisms tightly integrated with Docker services. While it covers the basic needs of containerized applications, it may not offer the same level of control and options as Kubernetes.



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/



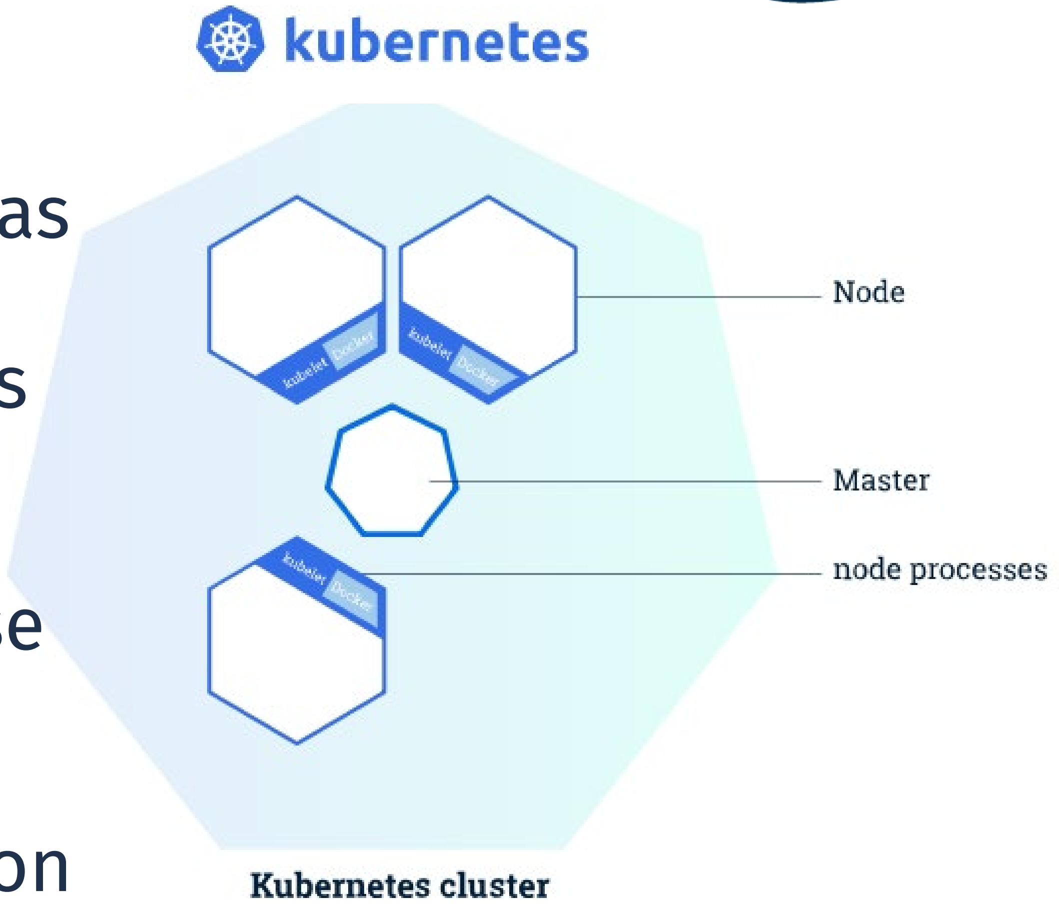
kubernetes Vs docker

kubernetes	docker
✓ STRONGLY SUPPORTED BY THE CNCF	✓ QUICK AND EASY INSTALLATION
✓ LARGE OPEN SOURCE COMMUNITY	✓ DOES NOT REQUIRE ADDITIONAL LIBRARIES OR COMPONENTS
✓ TESTED TO RUN IN PRODUCTION ON MOST OPERATING SYSTEMS AND CLOUD PROVIDERS	✓ SMOOTHER LEARNING CURVE FOR INEXPERIENCED OPERATORS
✓ CONVENIENT AND POWERFUL SERVICE MANAGEMENT	✓ INTEGRATES SEAMLESSLY WITH OTHER DOCKER TOOLS
✕ COMPLEX DEPLOYMENT	✕ LIMITED FUNCTIONALITY
✕ REQUIRES SEPARATE ADMINISTRATION TOOLS	✕ LIMITED HIGH AVAILABILITY AND FAULT RECOVERY CAPABILITIES
✕ NOT INTUITIVE FOR USERS FAMILIAR WITH DOCKER COMPOSE AND DOCKER CLI	✕ SMALLER OPEN SOURCE COMMUNITY COMPARED TO KUBERNETES
	✕ LIMITED ABILITY TO AUTOMATICALLY PROVISION RESOURCES.

Kubernetes components

Kubernetes

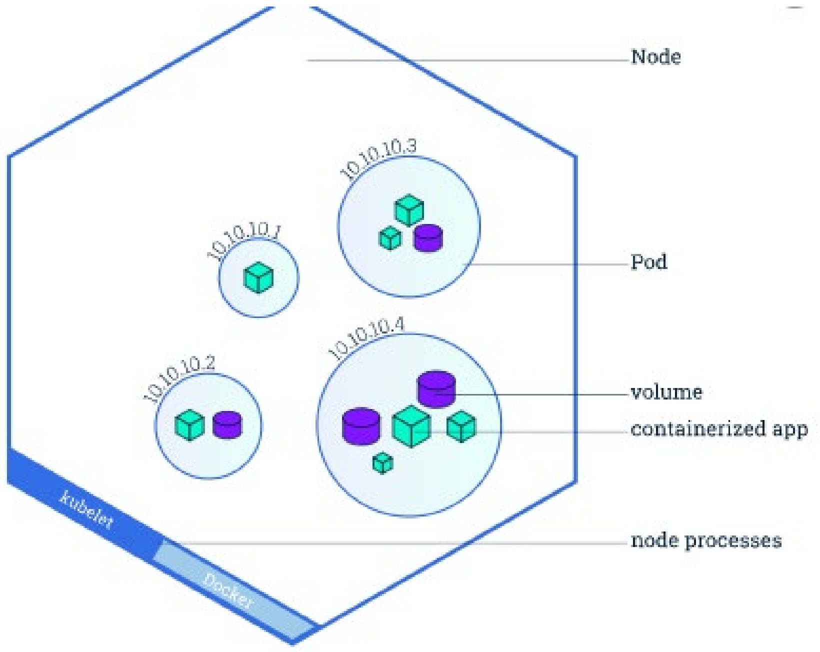
- Kubernetes coordinates a *cluster of computers* that are connected to work as a single unit.
- Applications that can run in Kubernetes cluster have to be **containerized**.
- Kubernetes then efficiently automates the distribution and scheduling of these containerized applications across the cluster.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

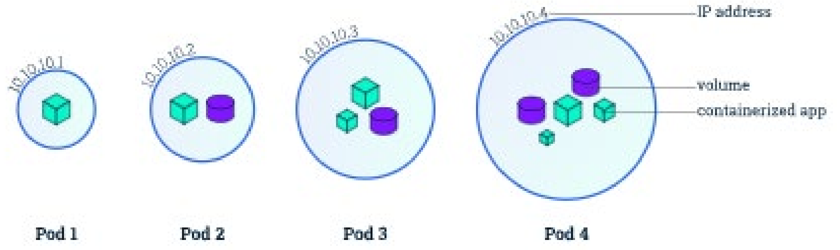


https://kubernetes.io

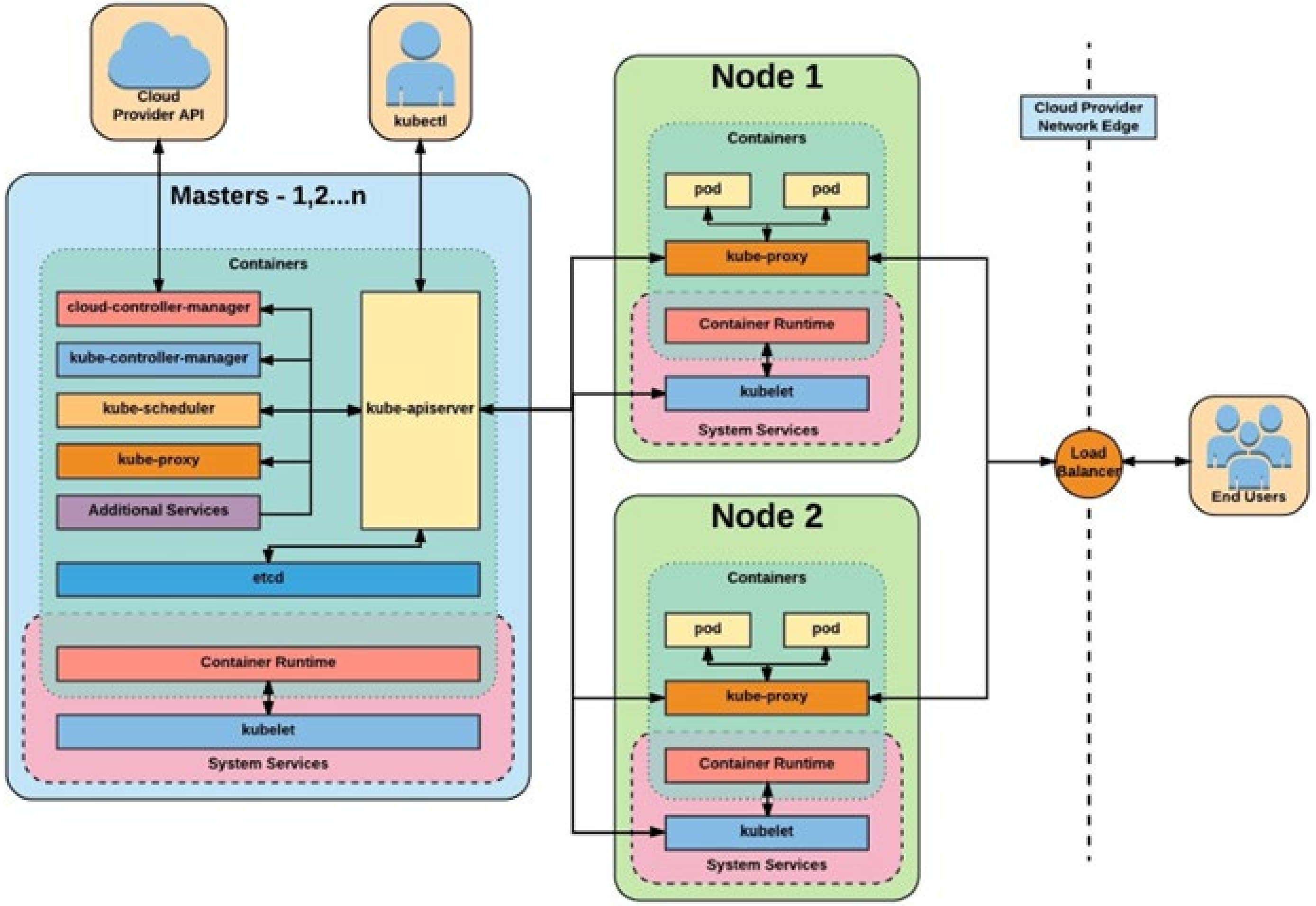
Kubernetes clusters

- 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**.

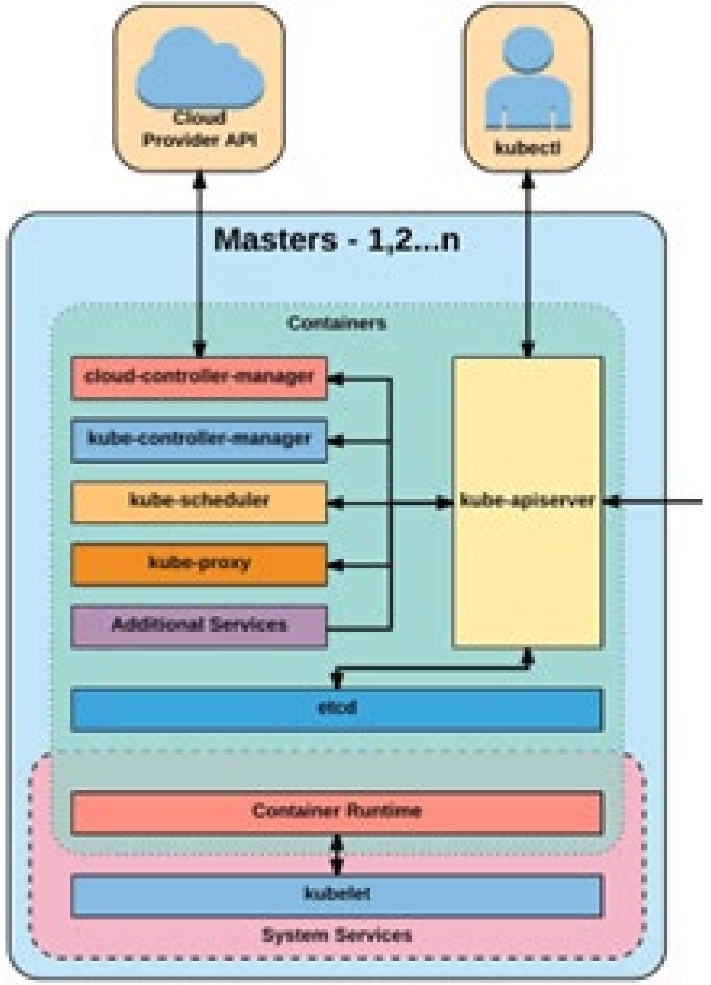




Kubernetes clusters

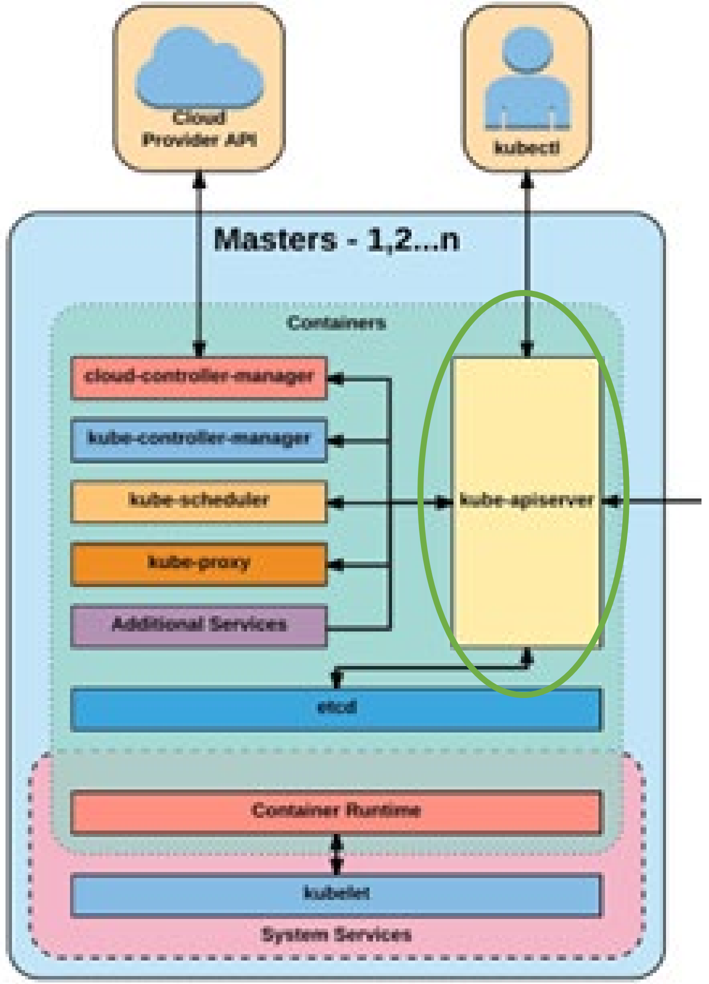


Kubernetes clusters: Master



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

Kubernetes clusters: apiserver

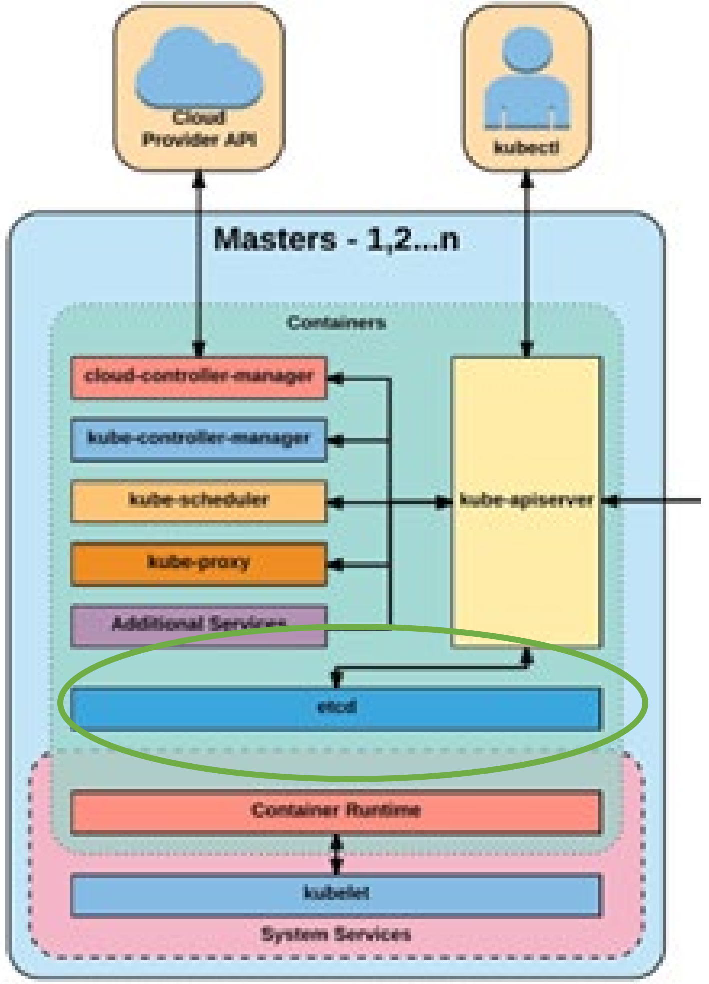


Kube-apiserver

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**, and admission control in addition to **being the front-end to the datastore (ETCD).**

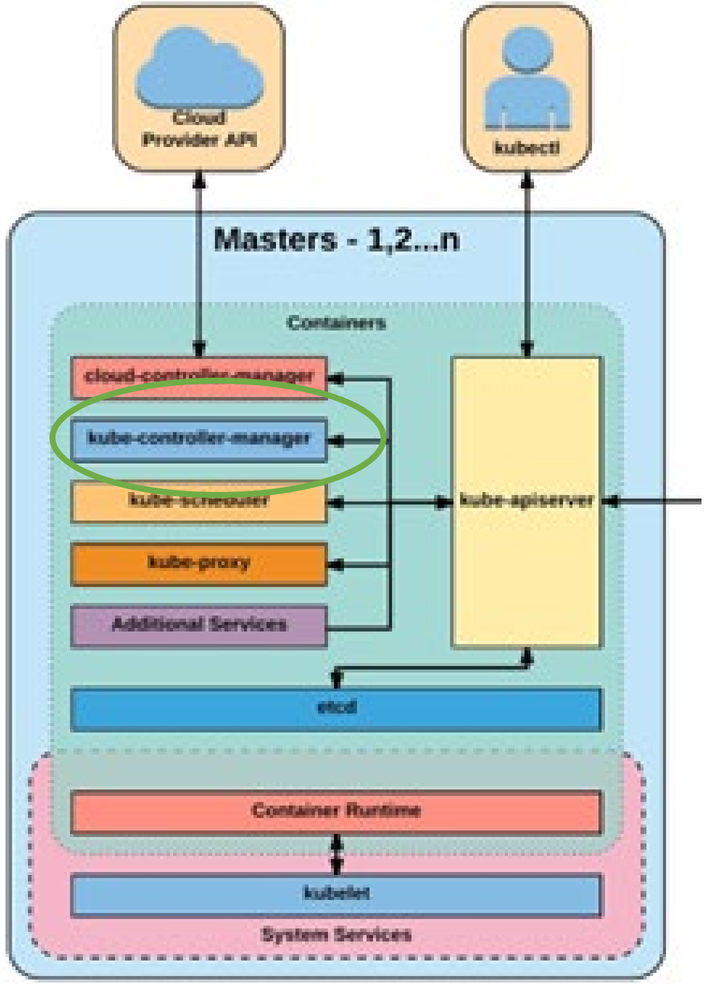
Kubernetes clusters: ETCD



ETCD

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

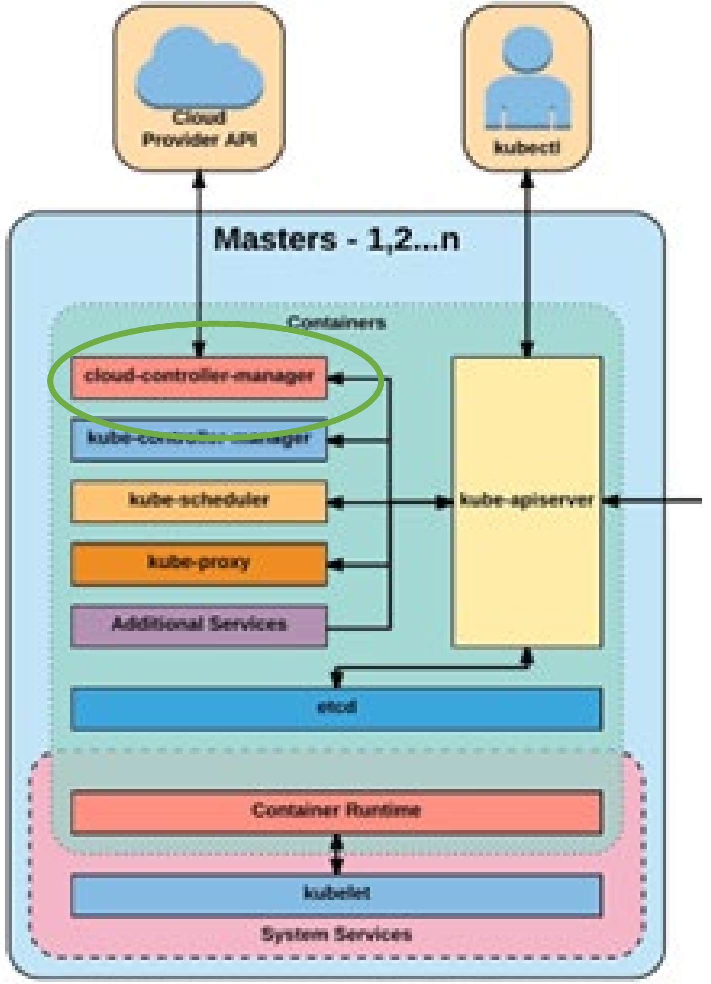
Kubernetes clusters: controller-managers



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.

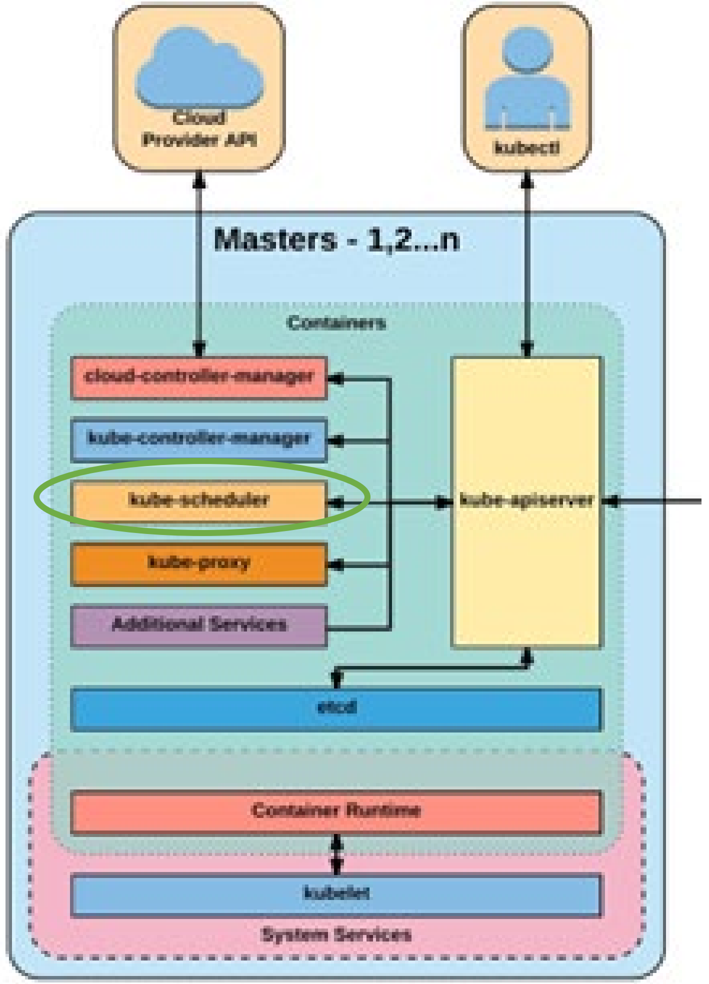
Kubernetes clusters: controller-managers



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: scheduler

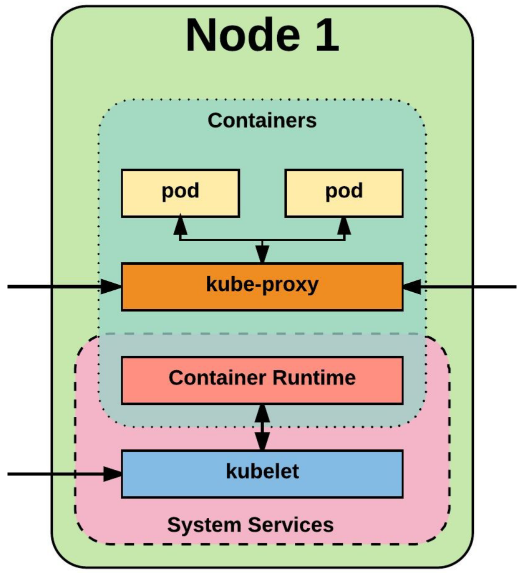


Kube-scheduler

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: node

- Kubelet
- Kube-proxy
- Container runtime engine



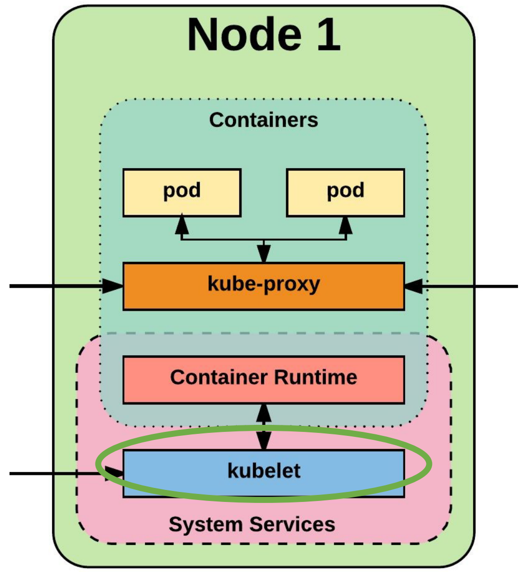
Kubernetes clusters: kubelet

Kubelet

Acts as the **node agent** responsible for **managing pod lifecycle** on its host.

The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy.

The kubelet doesn't manage containers which were not created by Kubernetes.



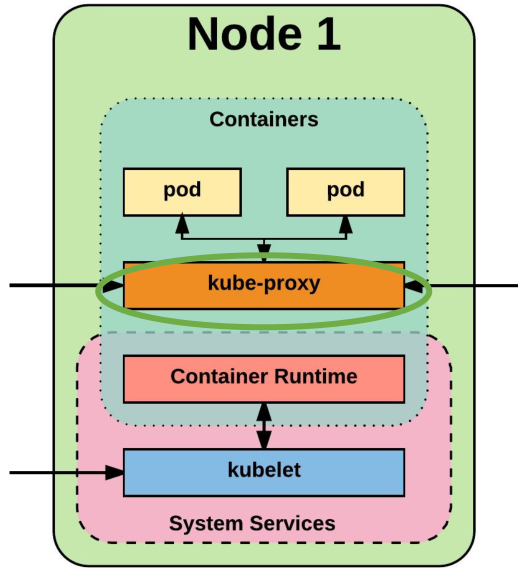
Kubernetes clusters: kube-proxy

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)



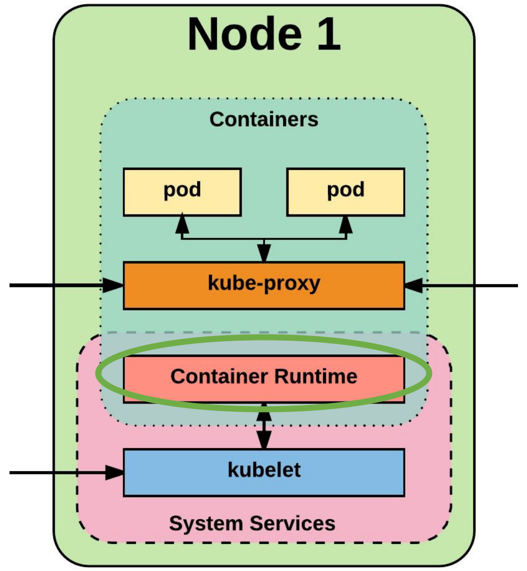
Kubernetes clusters: container runtime

Container Runtime

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

Container runtime supported by Kubernetes:

- 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)

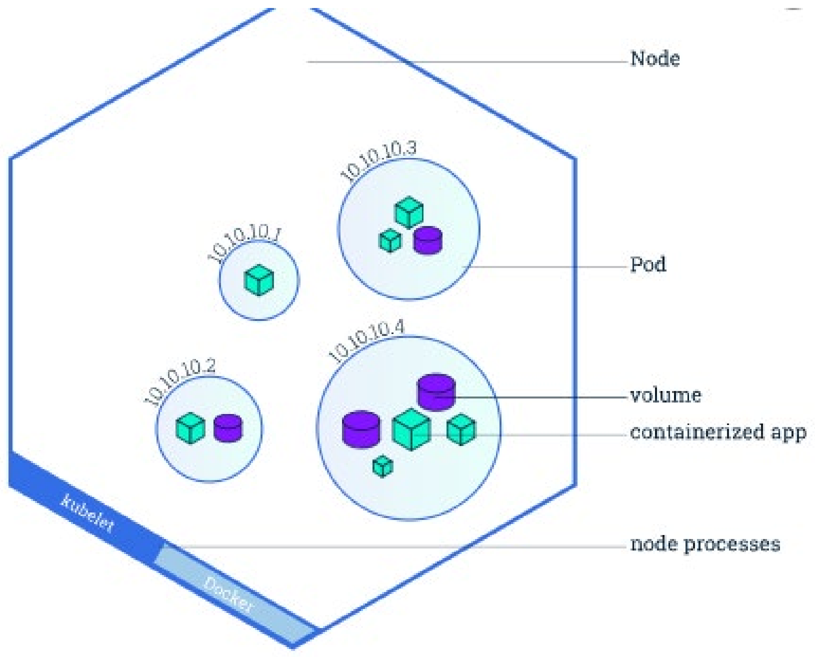
Kube-dashboard - A general purpose web based UI for kubernetes.

Kubernetes fundamentals

Kubernetes Pods

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

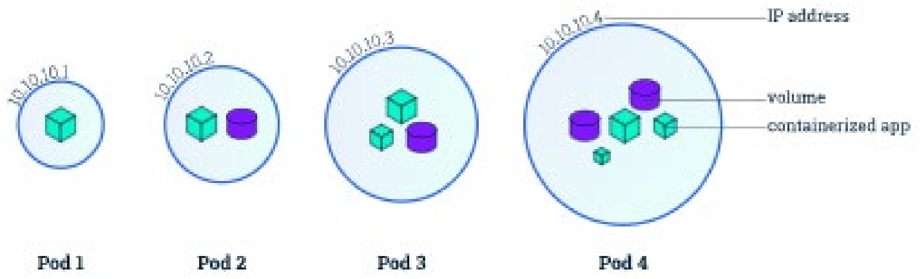
Node



Kinds of Pods

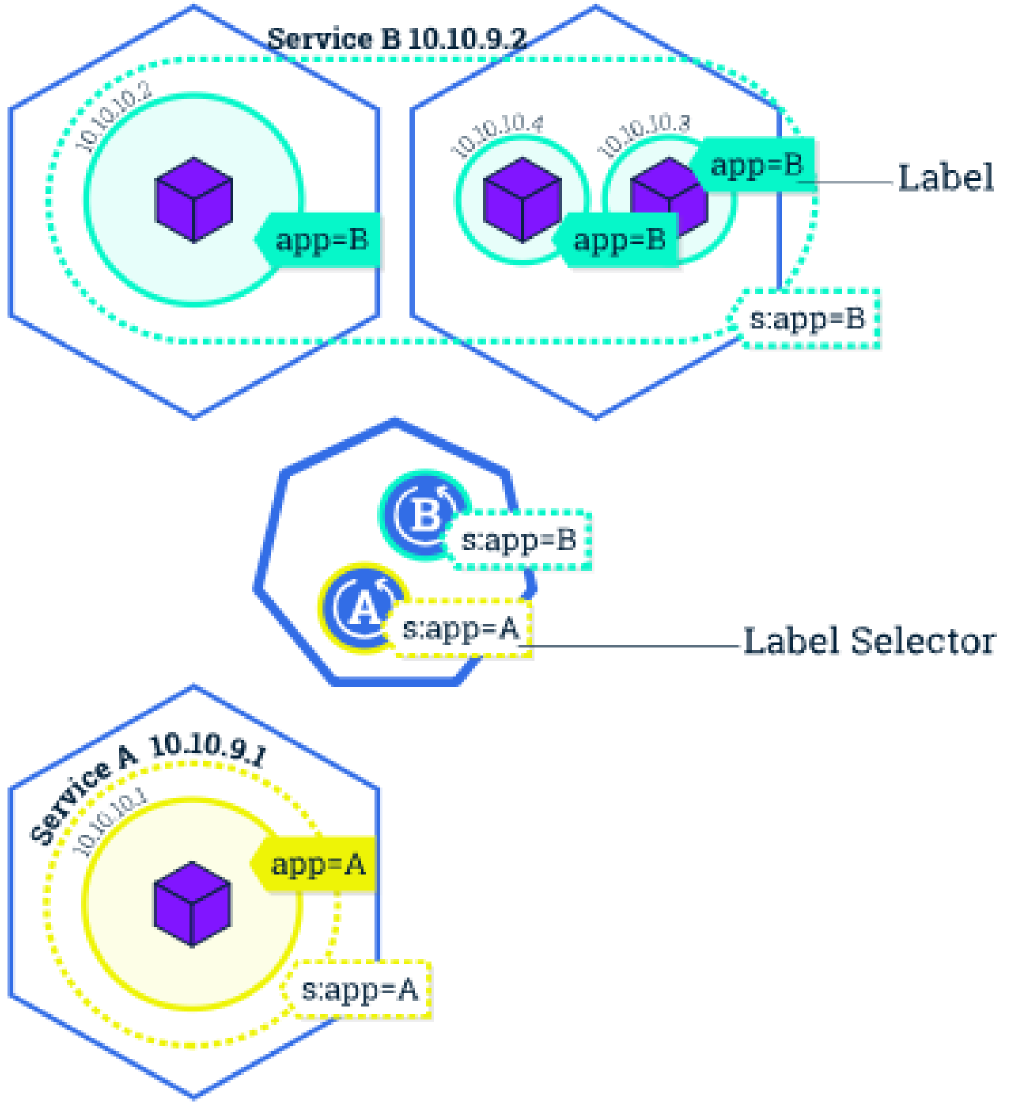
- We may have *two kinds of Pods*:
  - **Pods running a single container**. The “one-container-per-Pod” model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container. Kubernetes manages the Pods rather than the containers directly.
  - **Pods 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

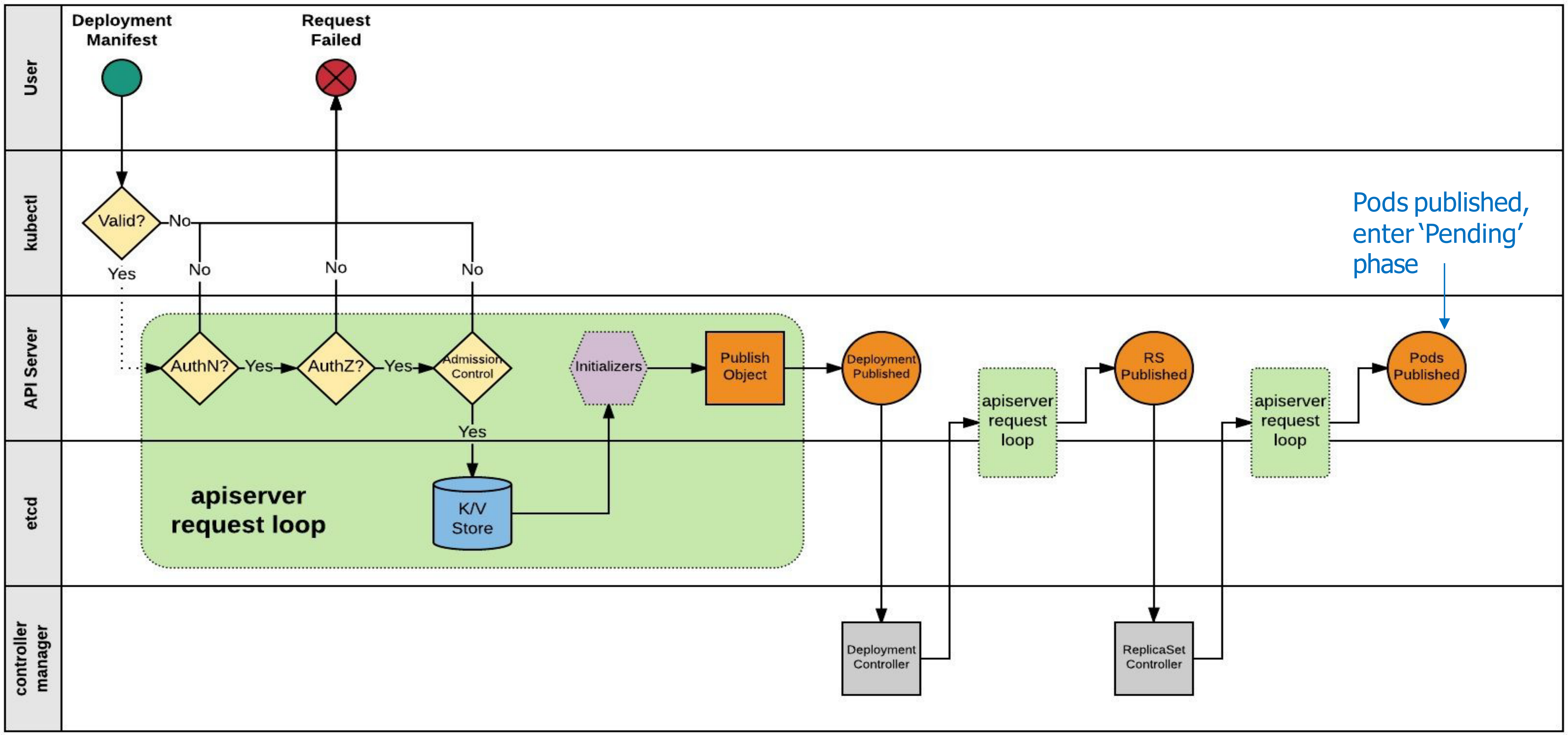
- A Kubernetes **Service** is an abstraction which defines *a logical set of Pods* and a policy by which to access it.
- Although each Pod has a unique IP address, these IPs are not exposed outside the cluster without a Service. Therefore, **you need Services to allow your applications to receive traffic**.
- 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



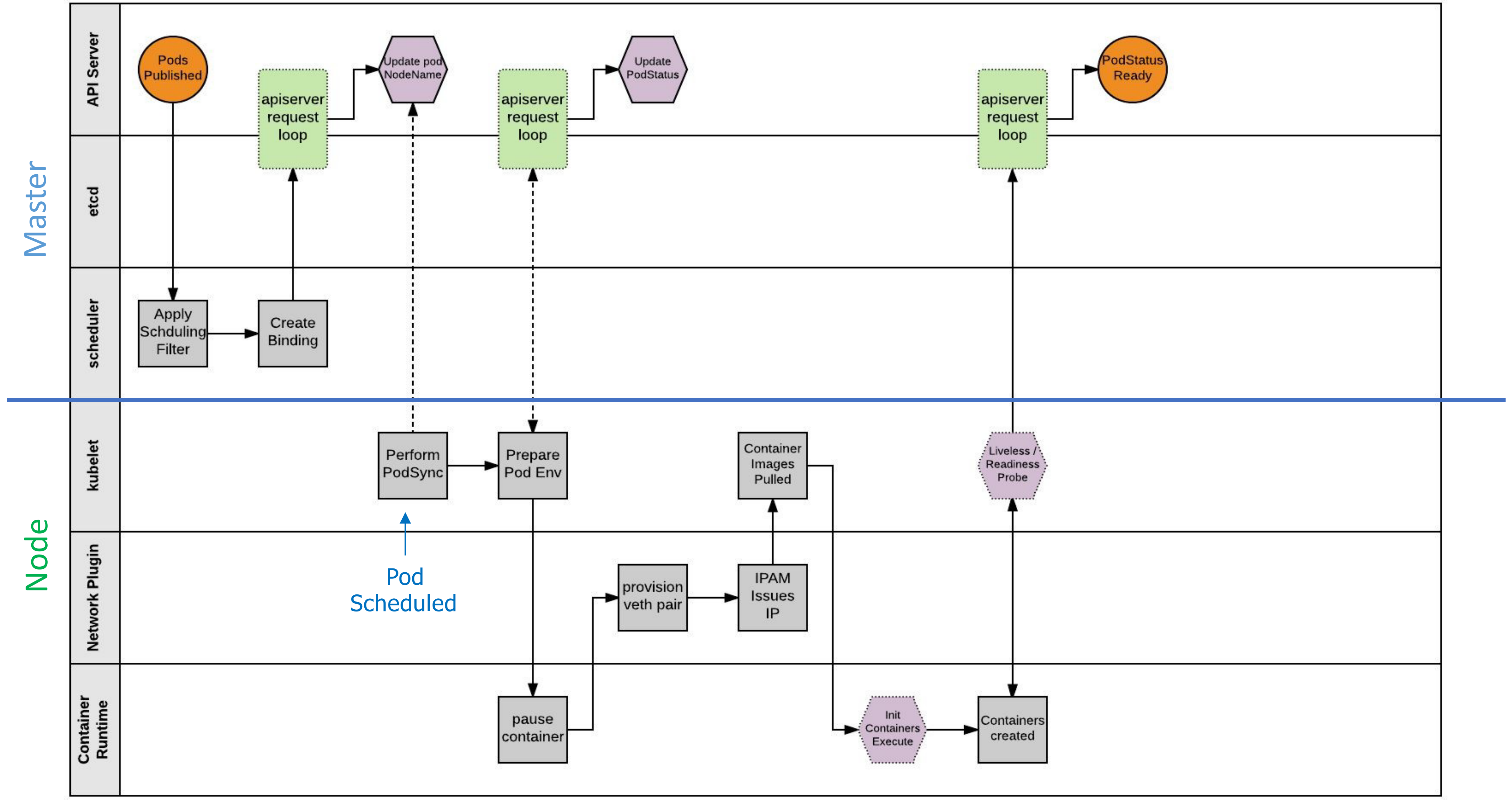
Kubernetes deployment steps

Kubernetes deployment

Master



Kubernetes deployment


Master
Node
API Server
etcd
scheduler
kubelet
Network Plugin
Container Runtime
Pods Published
apiserver request loop
Update pod NodeName
apiserver request loop
Update PodStatus
apiserver request loop
PodStatus Ready
Apply Schduling Filter
Create Binding
Perform PodSync
Prepare Pod Env
Container Images Pulled
Liveless / Readiness Probe
Pod Scheduled
provision veth pair
IPAM Issues IP
pause container
Init Containers Execute
Containers created