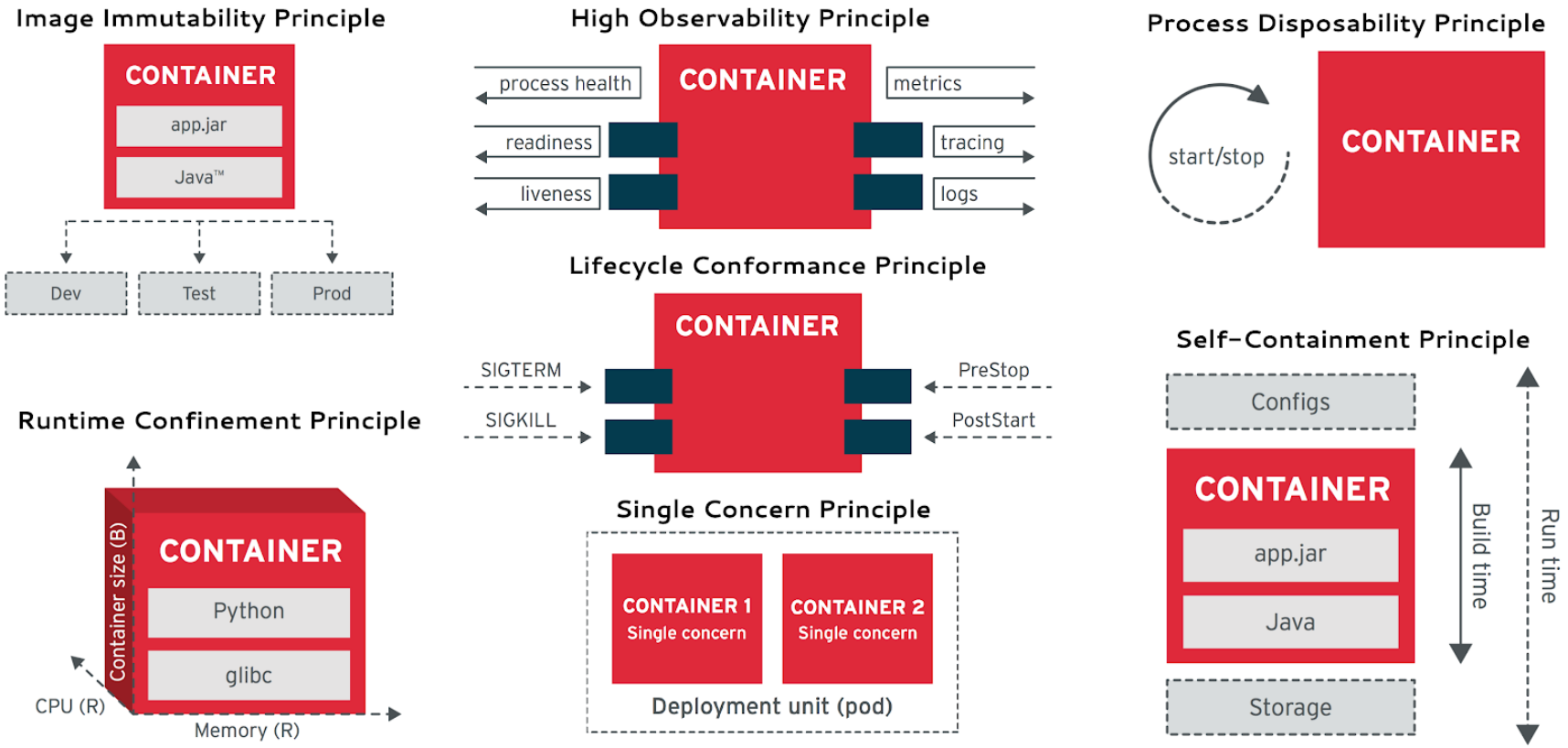
Kubernetes

Agenda

- Introduction
- Kubernetes Architecture
- Cluster
- Resources / Objects
- HPA
- Namespace
- Helm

Introduction

Principles of Container-based



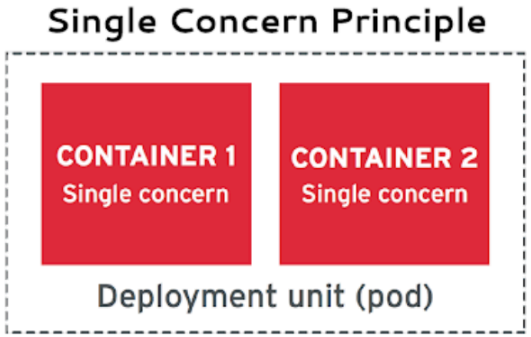
Principles of Container-based

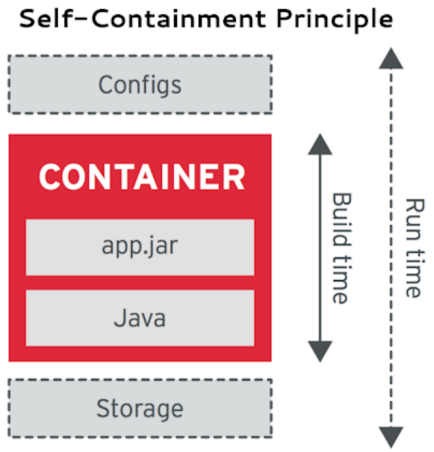
Build Time

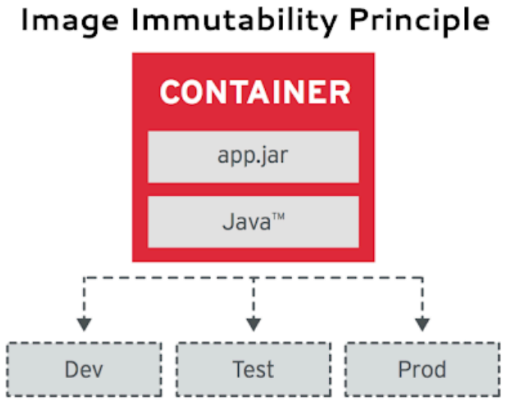
Single Concern: Each container addresses a single concern and does it well.

Self-Containment: A container relies only on the presence of the Linux kernel. Additional libraries are added when the container is built.

Image Immutability: Containerized applications are meant to be immutable, and once built are not expected to change between different environments.







Principles of Container-based

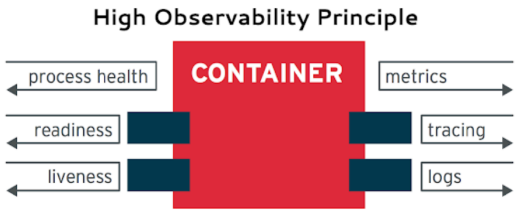
Run Time

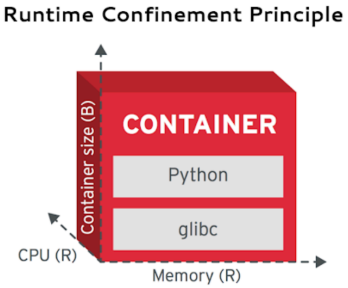
High Observability: Every container must implement all necessary APIs to help the platform observe and manage the application in the best way possible.

Lifecycle Conformance: A container must have a way to read events coming from the platform and conform by reacting to those events.

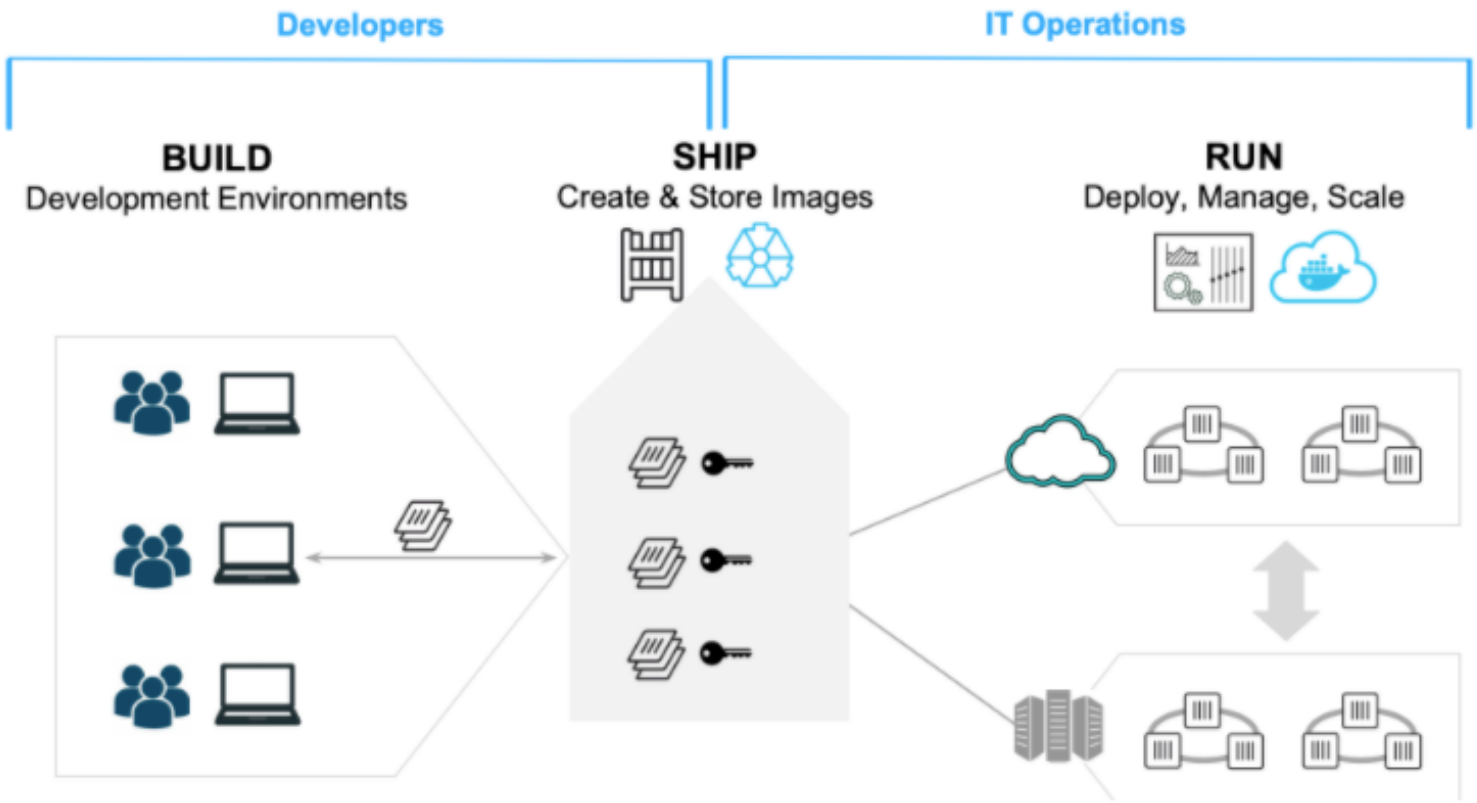
Process Disposability: Containerized applications must be as ephemeral as possible and ready to be replaced by another container instance at any point in time.

Runtime Confinement: Every container must declare its resource requirements and restrict resource use to the requirements indicated.

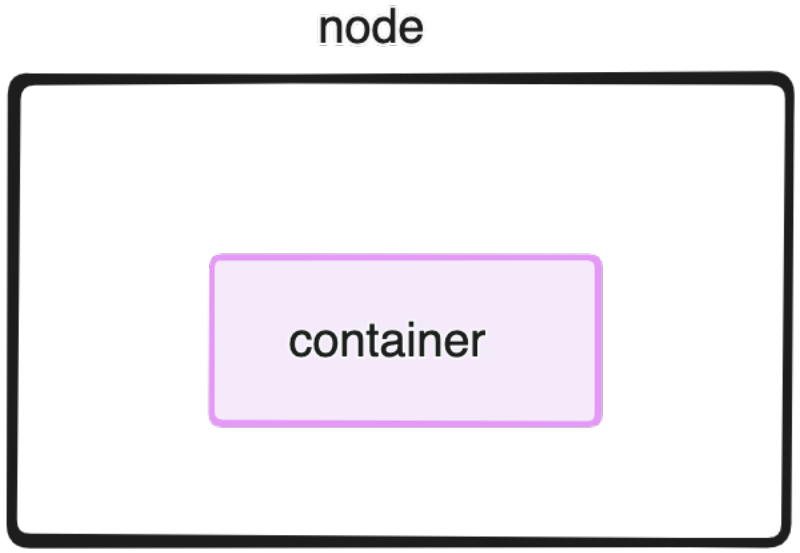




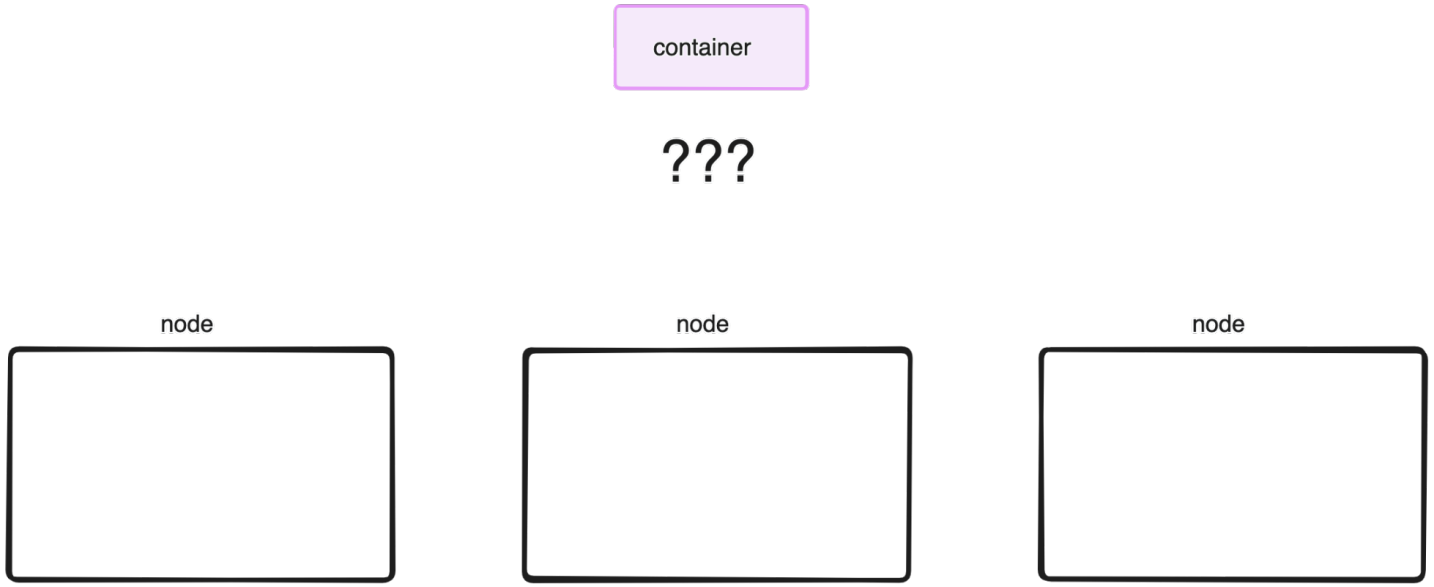
Using Docker: Build, Ship, Run



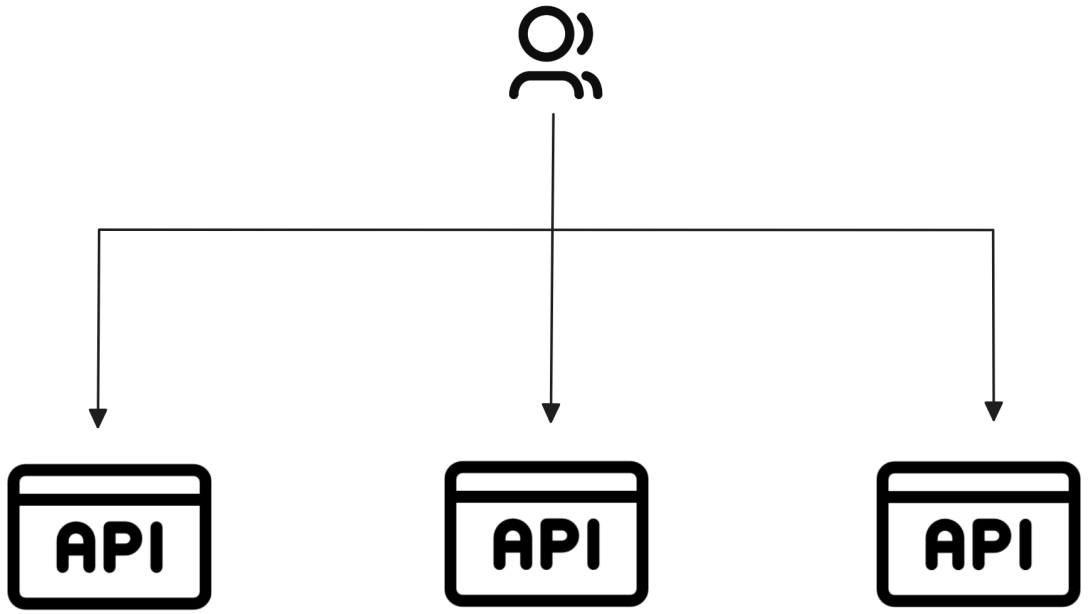
Docker standalone server



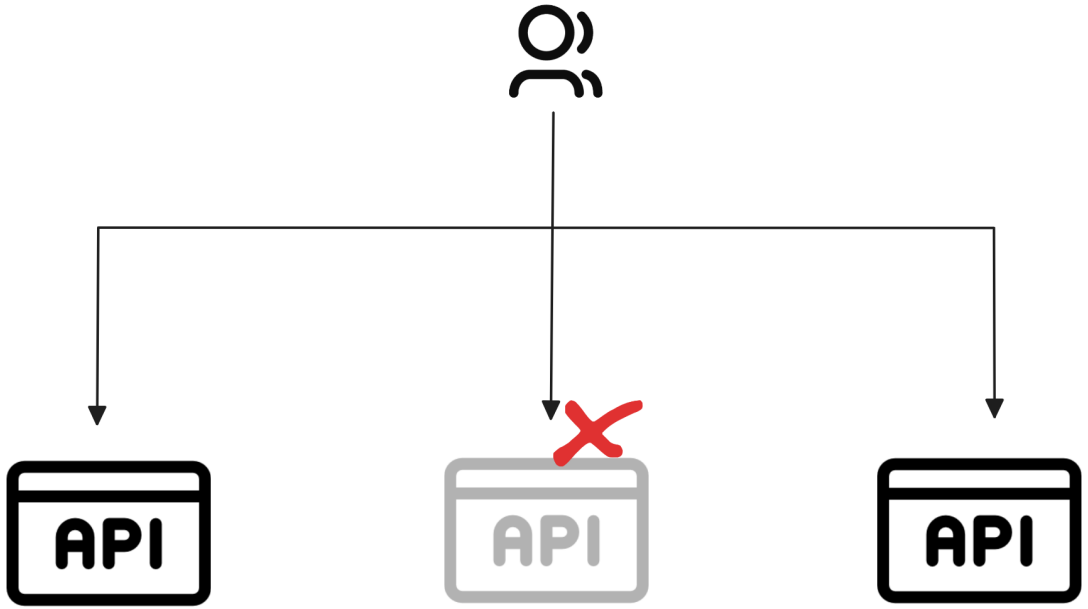
Docker multi server



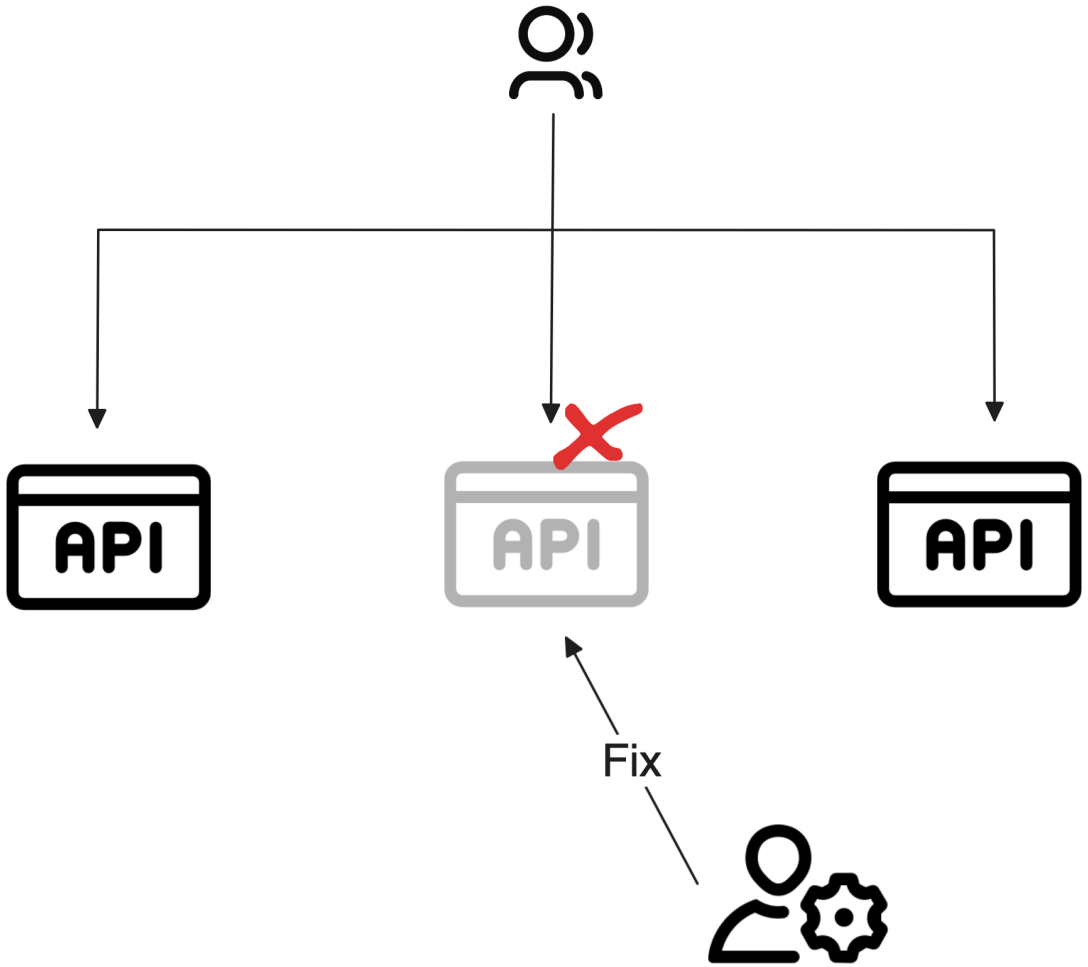
Applications problem



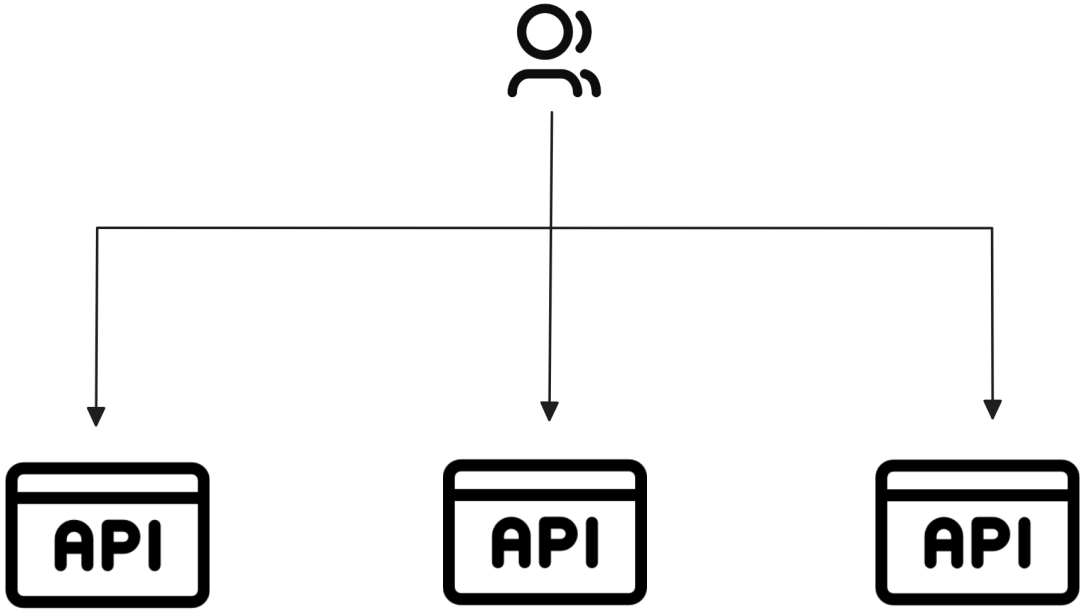
Applications problem



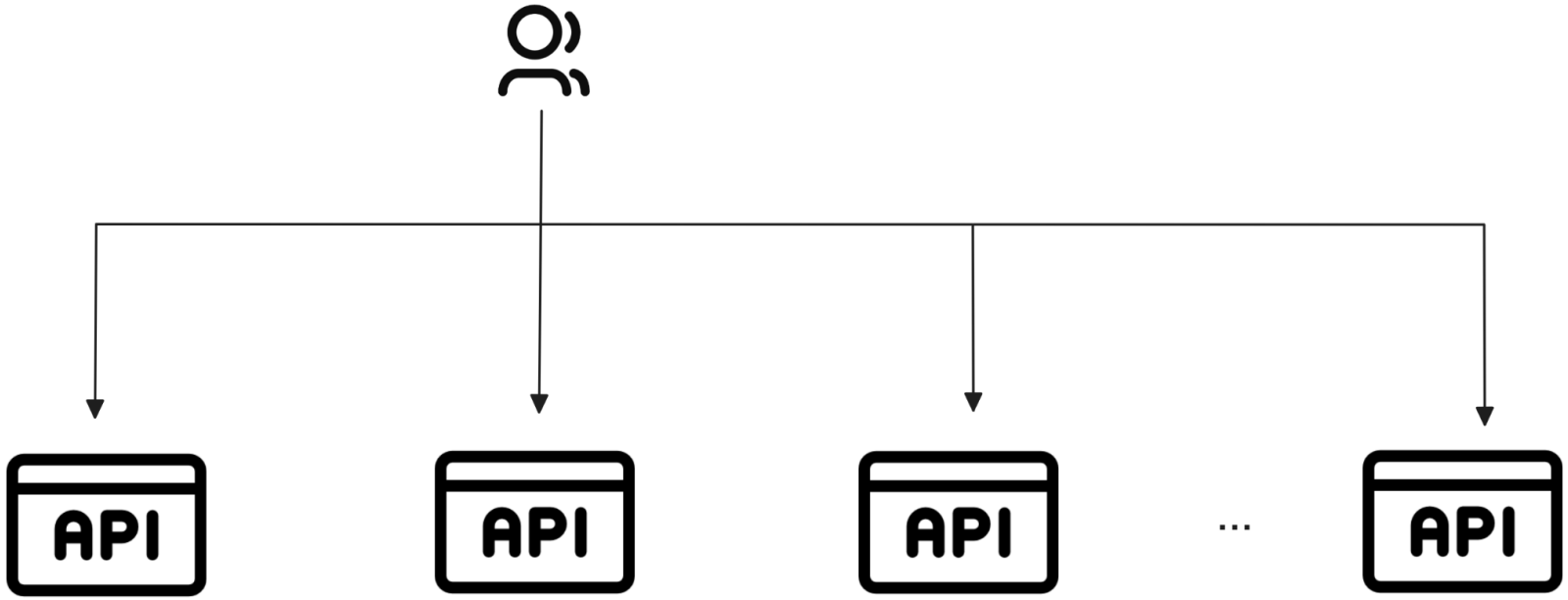
Applications problem



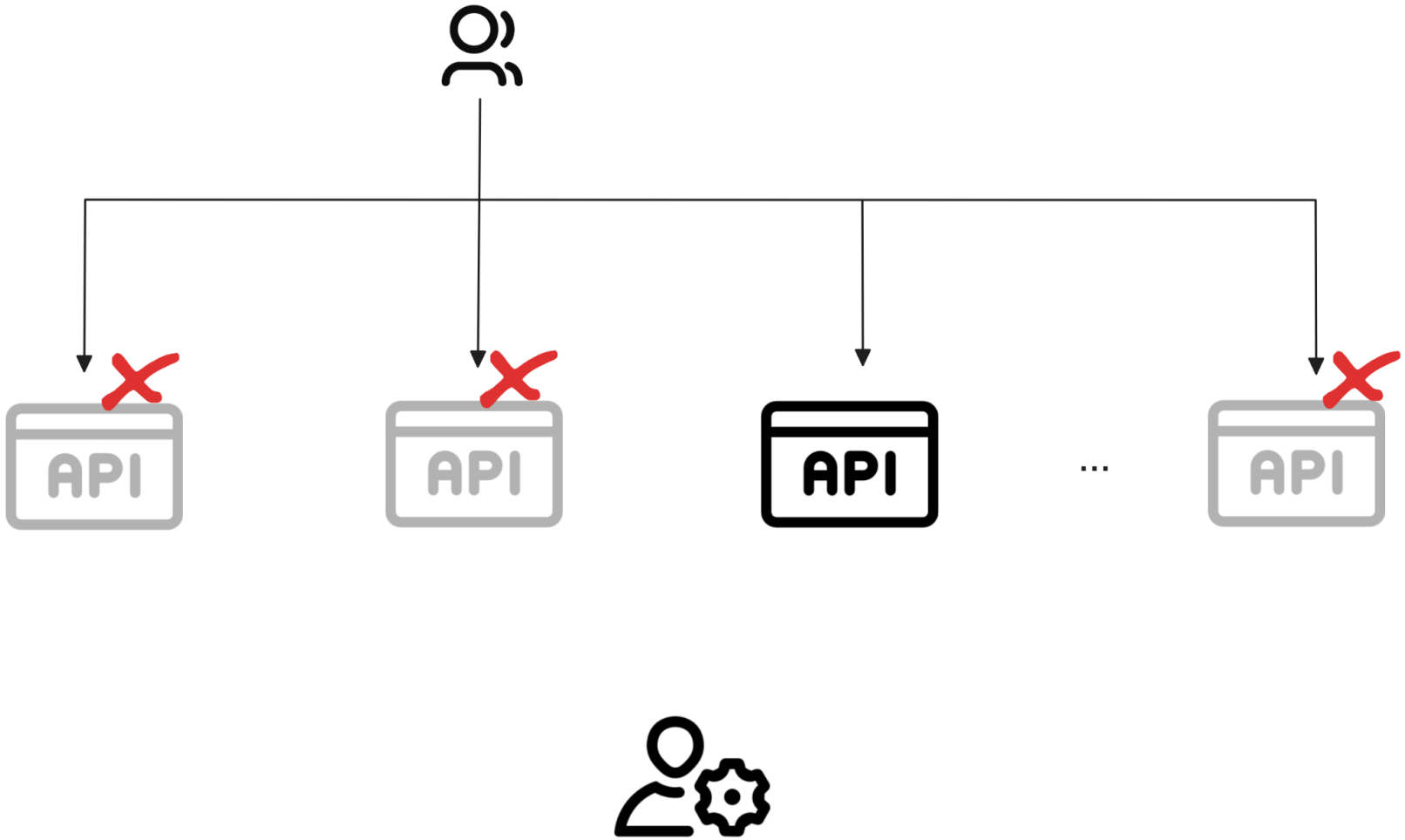
Applications problem



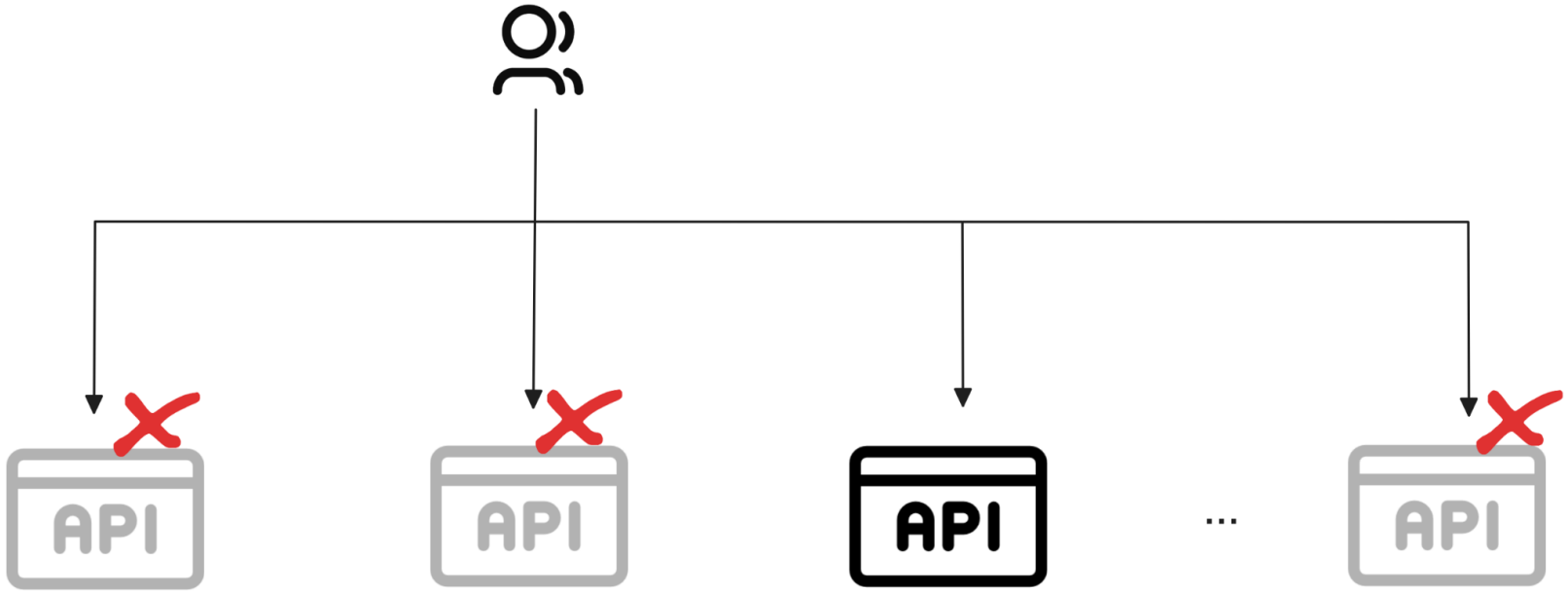
Applications problem



Applications problem



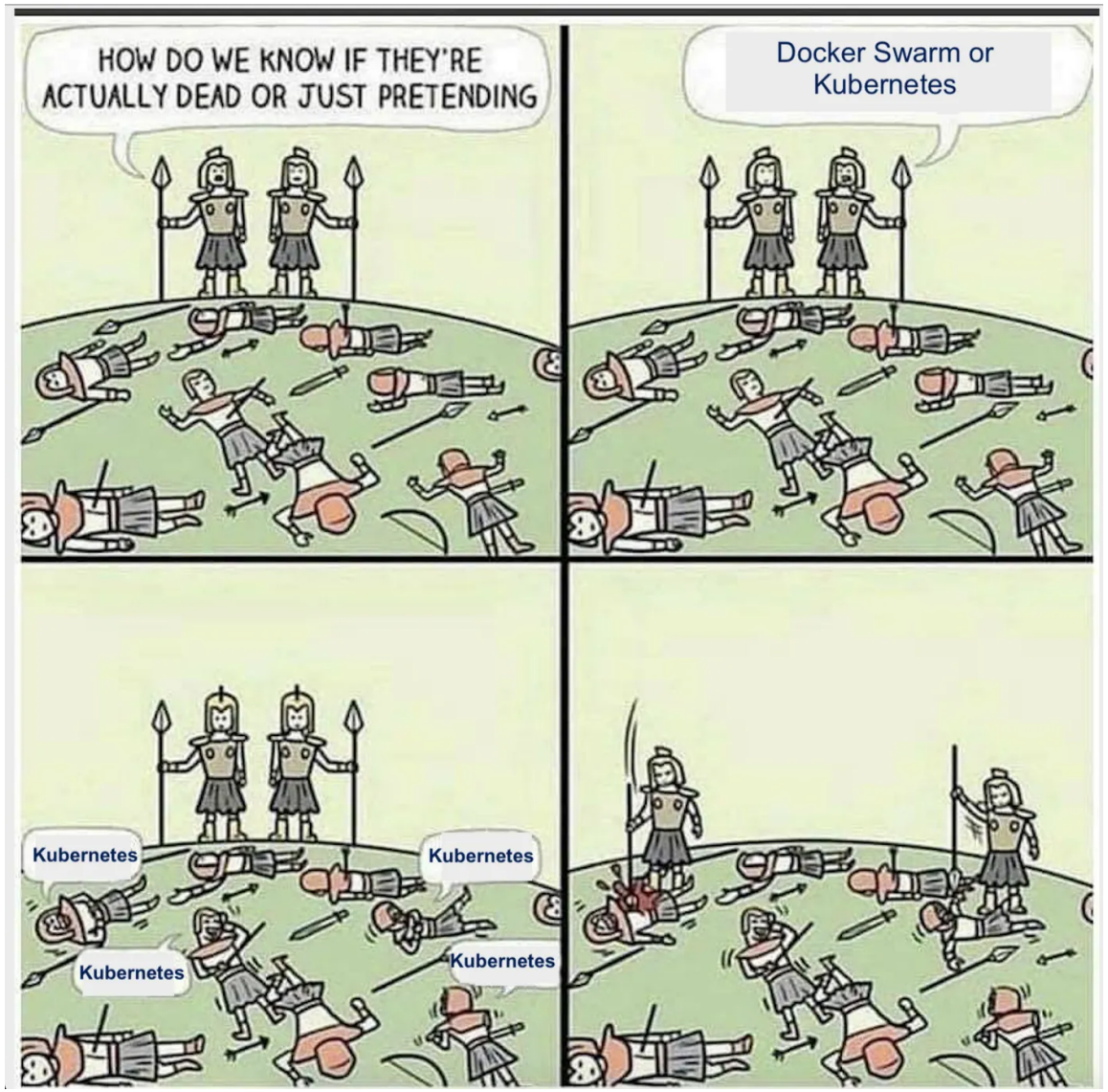
Applications problem



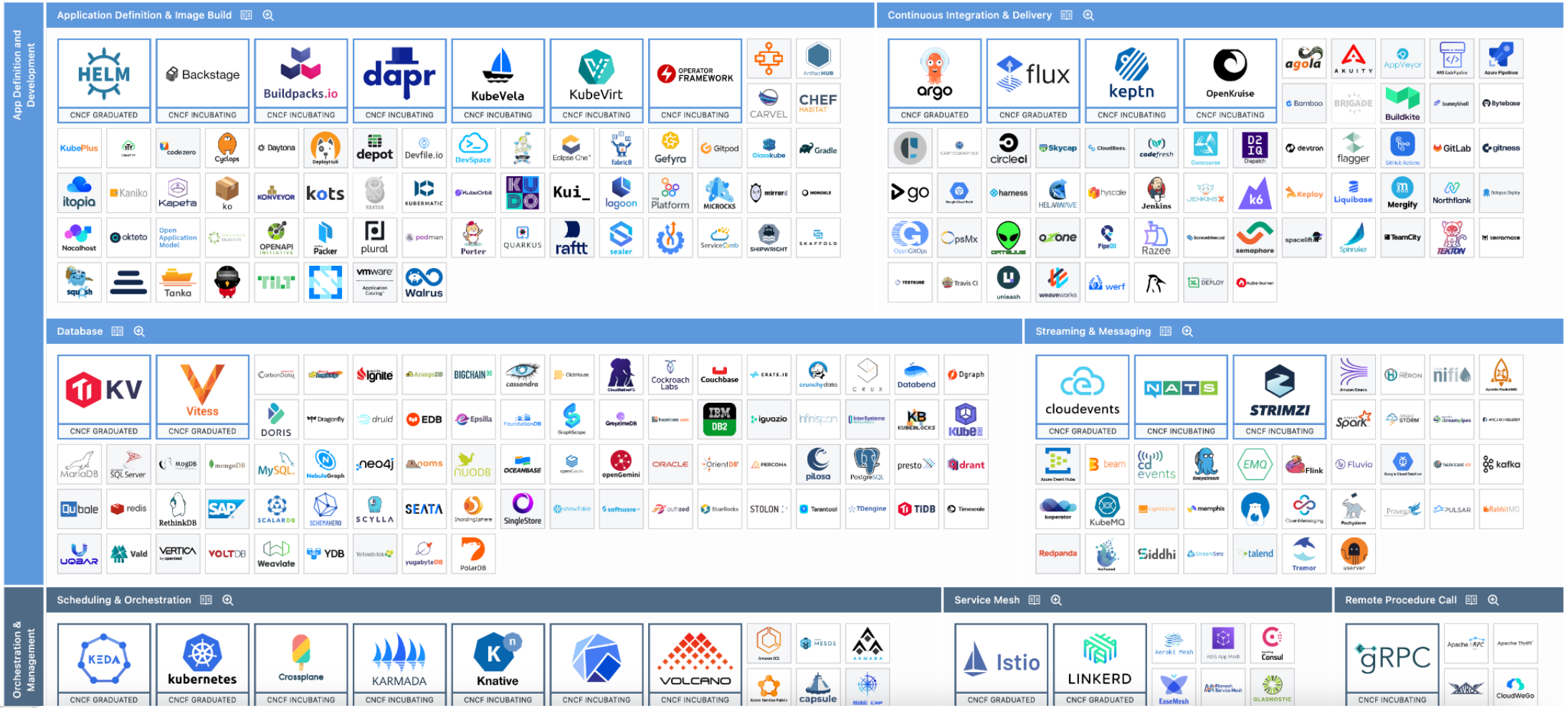
Fix !!!!

What is Kubernetes ?

- Kubernetes in Greek means the Helmsman
- Also known as k8s
- Inspired and informed by Google’s experiences and internal systems
- 100% opensource, written in GO
- Container orchestrator, runs and manages containers
- Kubernetes v1.0 released on July 2015, join CNCF on March 2016
- Container choose Docker, Container Orchestrator choose Kubernetes



CNCF Landscape

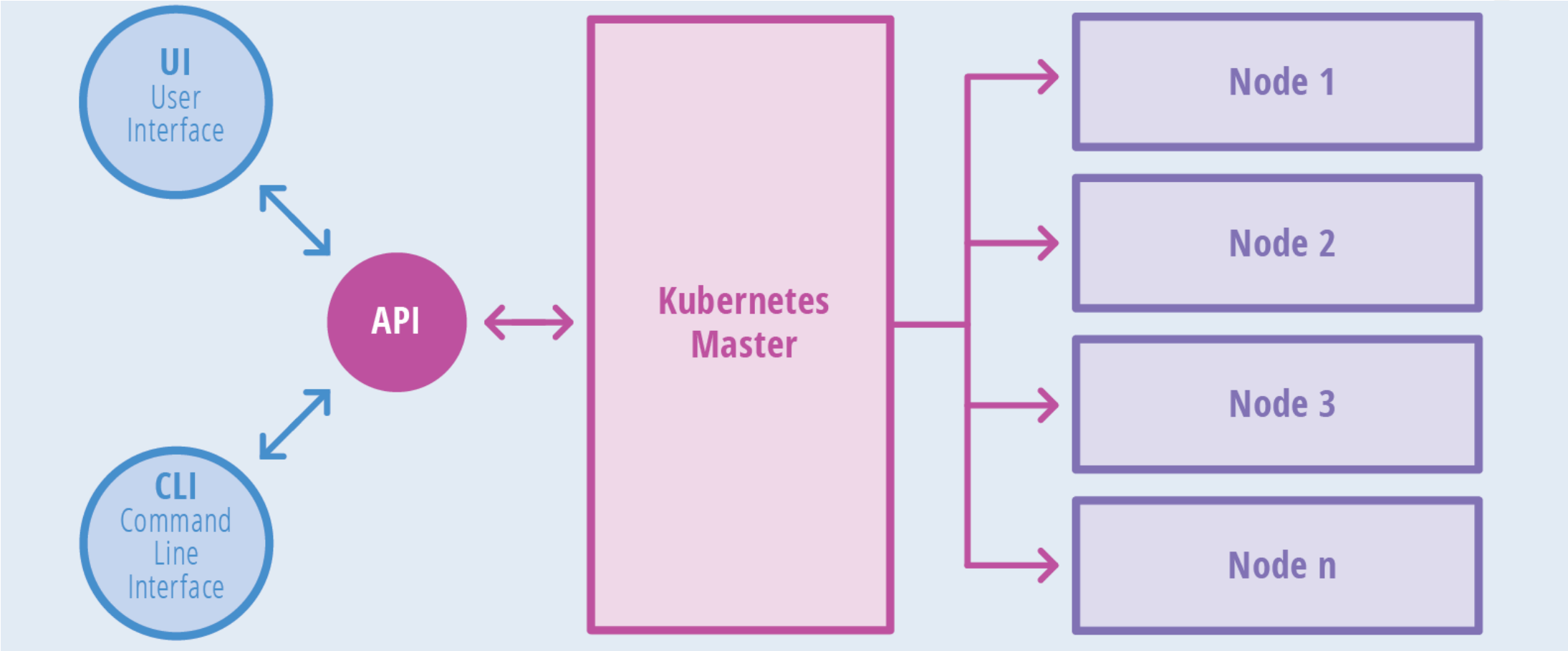


The features of Kubernetes

- Automatic bin packing
- Self healing
- Automated rollout & rollback
- Service Discovery & load Balancing
- Horizontal scaling
- Declarative Configuration

Kubernetes Architecture

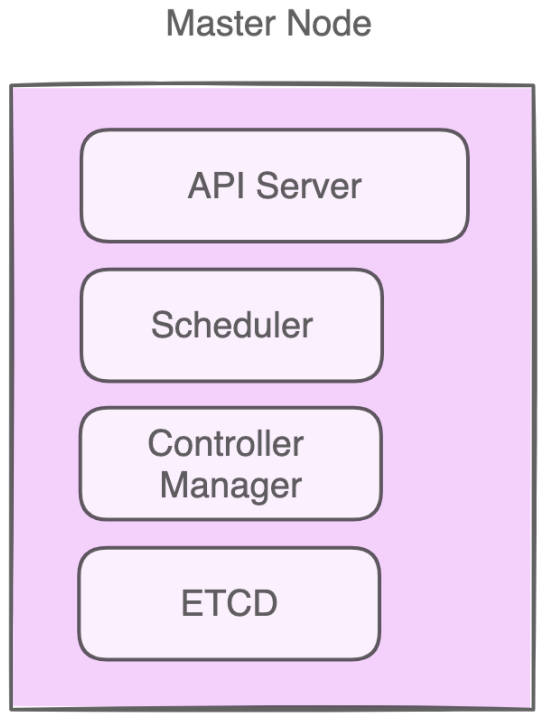
Kubernetes Architecture



Kubernetes Architecture

Master node

- Responsible for managing cluster
- Access master node via CLI, UI or API
- For fault tolerance, can be more than 1 master in cluster
- Include 4 components: API Server, Scheduler, Controller manager and ETCD



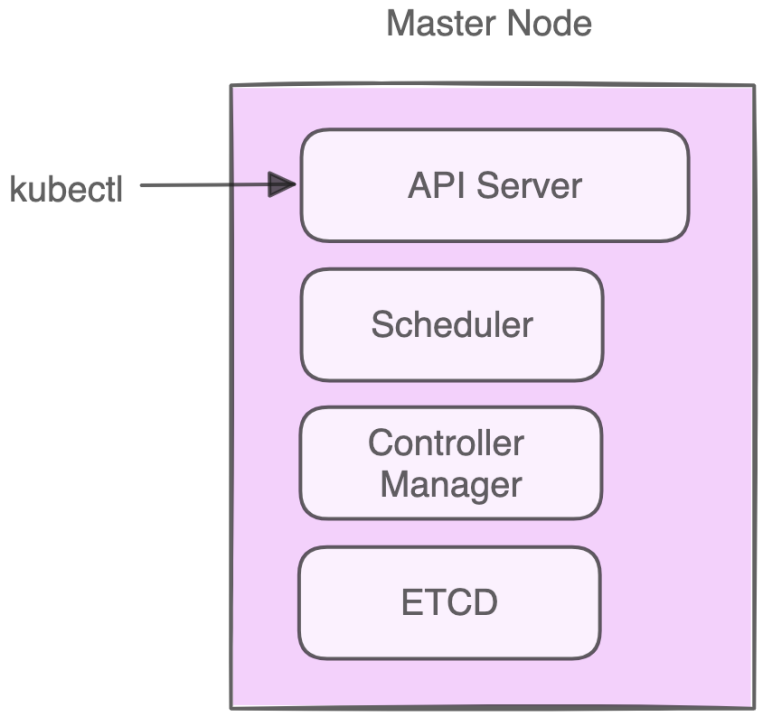
Kubernetes Architecture

Master node

API Server

- Centralized component where all the cluster components ar communicated

- Execute and validate from user command

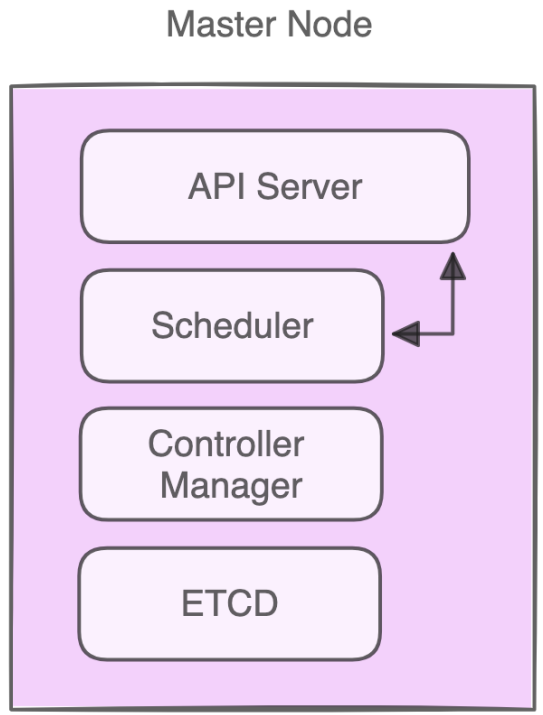


Kubernetes Architecture

Master node

Scheduler

- Assigning the application to worker node
- Automatically detect which pod to place on which node based on all factors

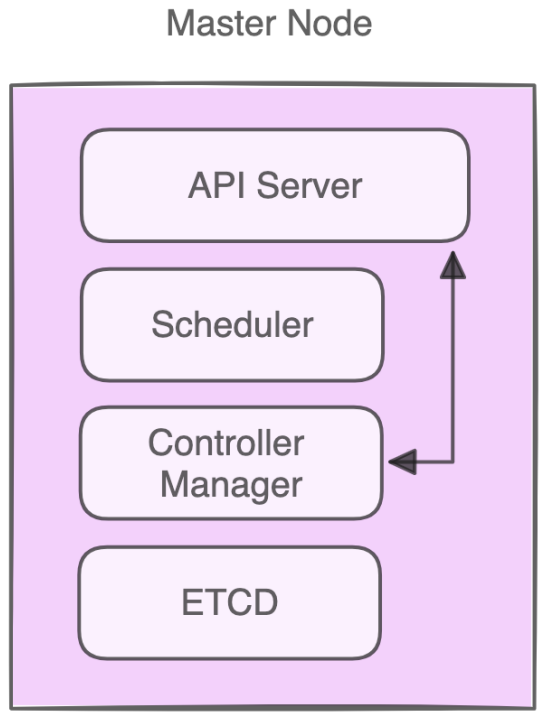


Kubernetes Architecture

Master node

Controller Manager

- The Controller Manager maintains the cluster
- Handles node failures, replicating components, maintaining the correct number of pods, etc.

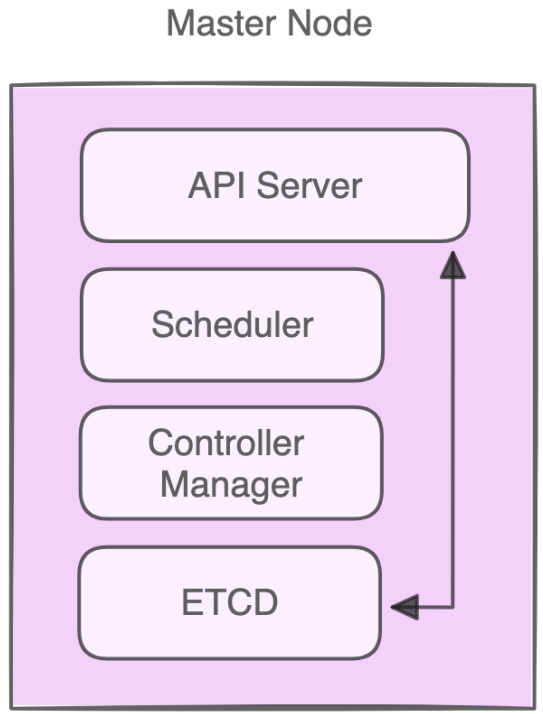


Kubernetes Architecture

Master node

ETCD

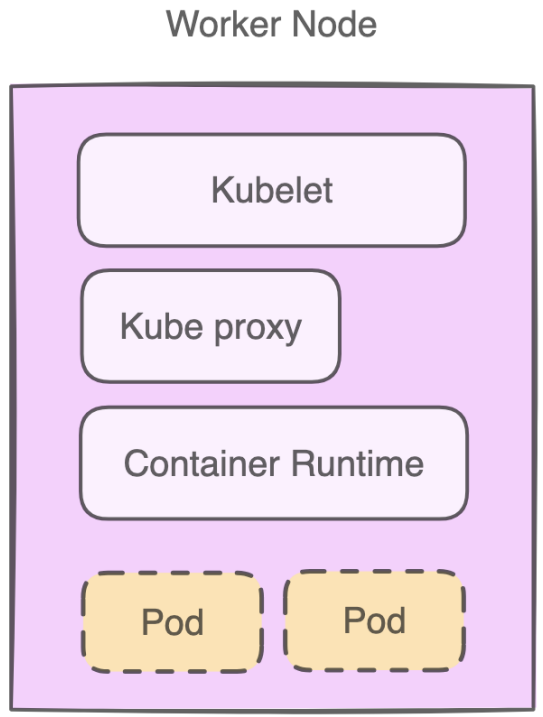
- ETCD is a distributed reliable key-value store use by Kubernetes to store all data used to manage the cluster
- When you have multi master node in cluster, ETCD stores all information in a distributed manner
- Application data doesn't store in ETCD



Kubernetes Architecture

Worker node

- Responsible about applications
- Include 3 components: **Container Runtime, Kubelet** and **Kube Proxy**

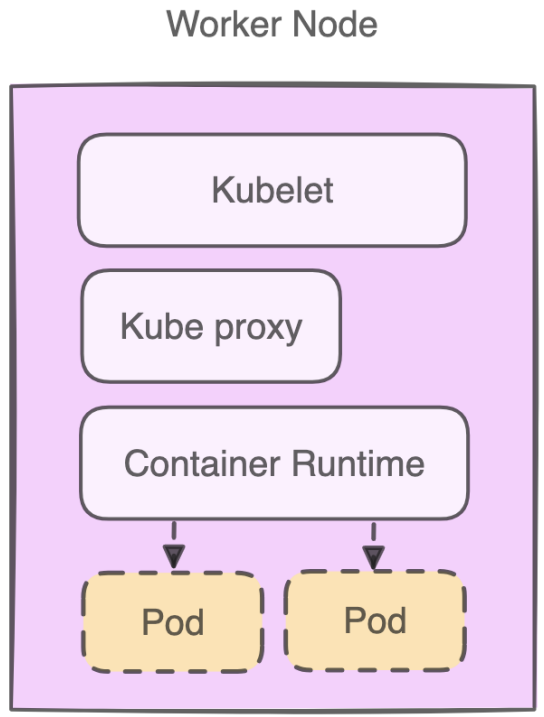


Kubernetes Architecture

Worker node

Container Runtime

- Container runtime which runs the containers like Docker, crio or containers
- Pulling the images and and run the containers

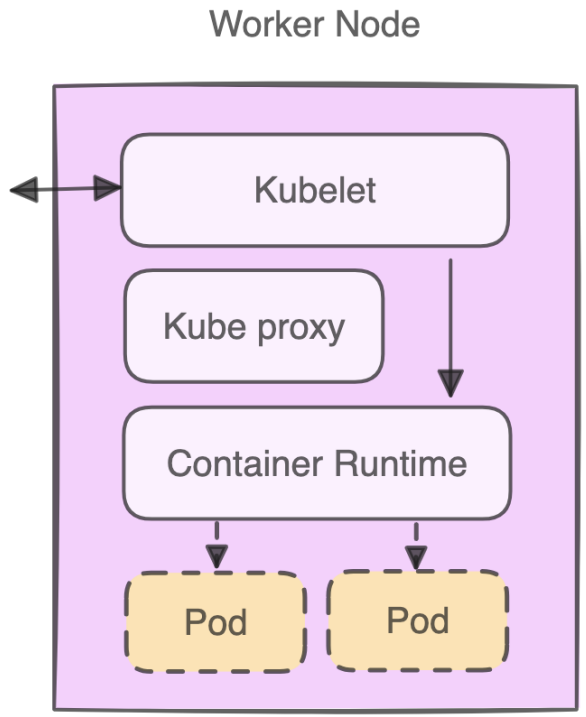


Kubernetes Architecture

Worker node

Kubelet

- Interacting with the master node
- Contact the container runtime for create pod depend on Po
- Provide health information of the worker node

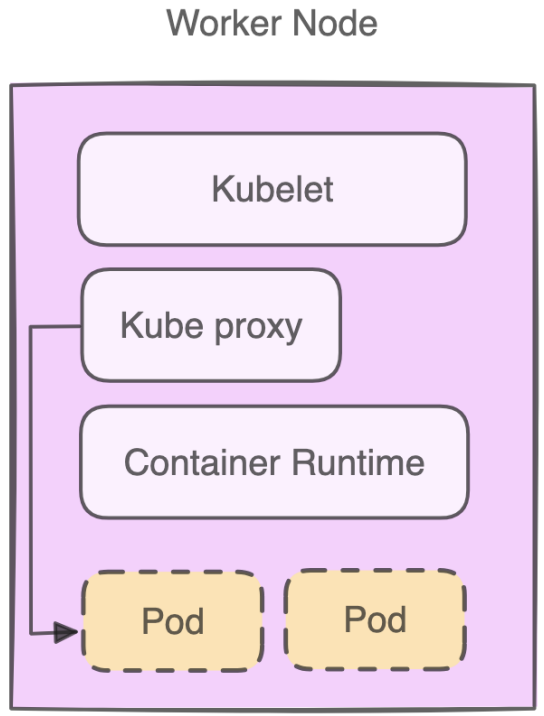


Kubernetes Architecture

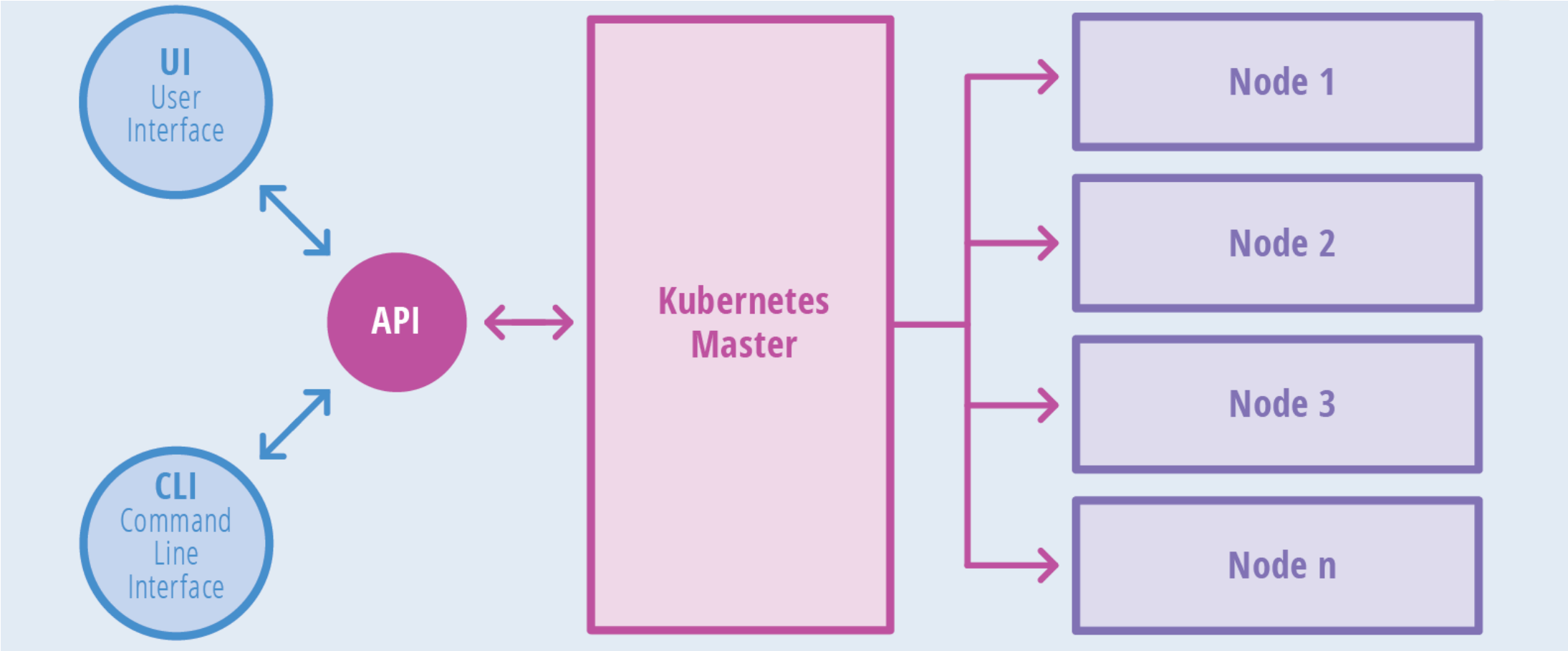
Worker node

Kube Proxy

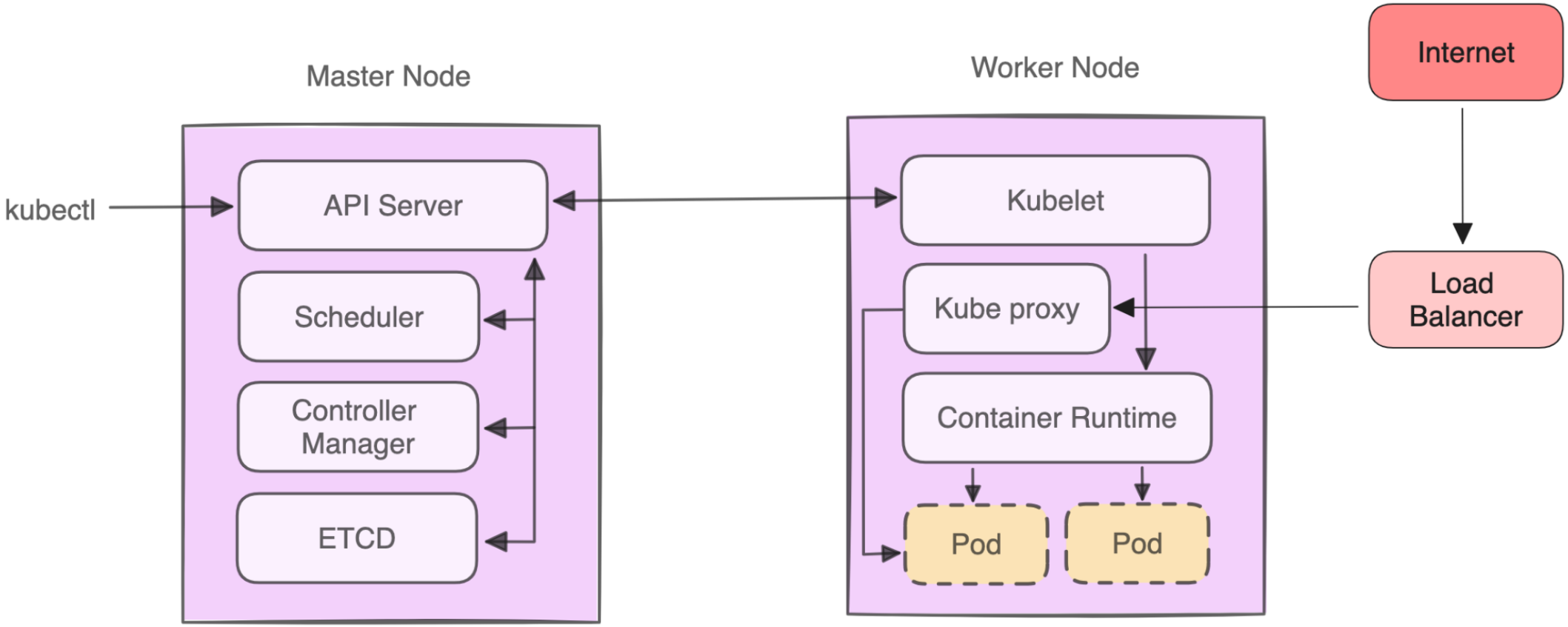
- Responsible for ensuring network traffic is routed properly to internal and external services
- Manage route for reduce latency time for example pod will communicate with another pod on same node first



Kubernetes Architecture



Kubernetes Architecture



Kubectl

Kubectl

- Command line
- Control the Kubernetes cluster manager
- CRUD Kubernetes resources

Kubectl example

- $ kubectl apply
- $ kubectl create
- $ kubectl get
- $ kubectl describe
- $ kubectl delete

Cluster

Cluster

- On Cloud
- On Premise

Cluster on cloud

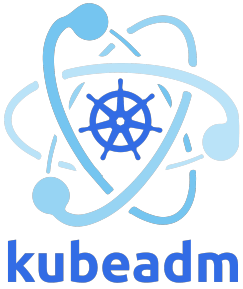
Amazon **EKS**

Google Kubernetes Engine

Azure Kubernetes Service (AKS)

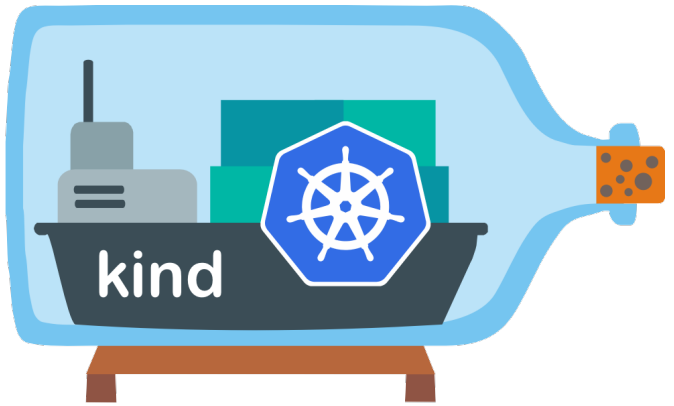
Alibaba Cloud

Cluster on premise



Cluster on local development

K3D



minikube

Cluster comparing

Feature	Kubernetes on Cloud	Kubernetes On-Premise
Setup & Maintenance	- Easy to set up with automated updates and patches - Minimal operational overhead	- Requires manual setup - Ongoing maintenance and upgrades are the user's responsibility
Cost	- Hight depend cloud provider - Costs tied to usage - No upfront infrastructure investment	- Lower costs - High initial investment for hardware and infrastructure
Scalability	- Highly scalable with near-infinite resources. - Automatic scaling options available	- Limited by on-premise hardware resources. - Scaling may require additional hardware and setup.
Disaster Recovery	- Built-in disaster recovery and backups.	- Must design and implement your own
Customization	- Limited by cloud provider's offerings and configurations.	- Full control over configurations and infrastructure.

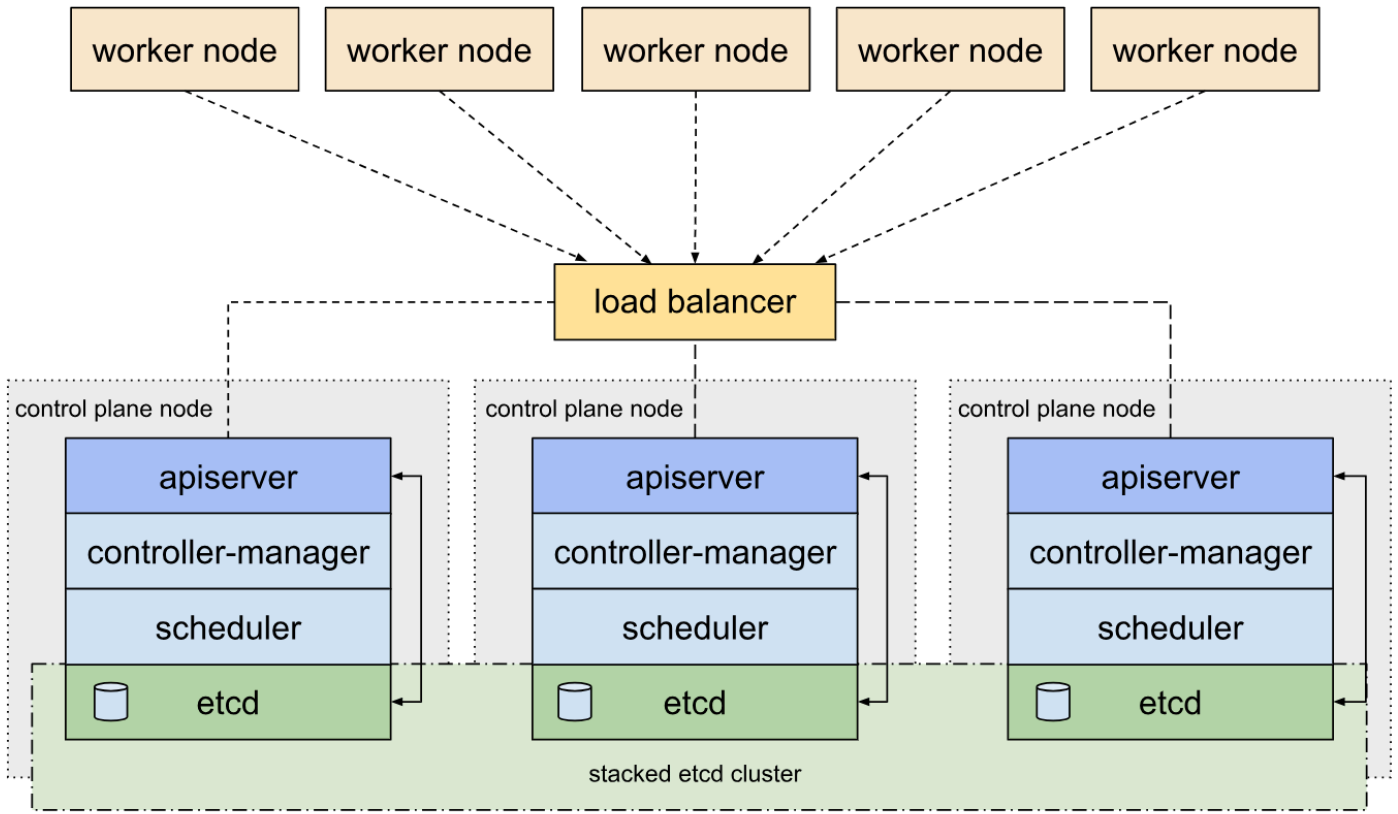
Cluster comparing

Feature	Kubernetes on Cloud	Kubernetes On-Premise
Security	- Security managed by cloud providers	- Full control over security configurations
Networking	- Advanced networking features available (e.g., VPCs, load balancers). - Simplified setup with integration to other cloud services	- Full control over network configurations. - Requires in-house expertise for complex networking setups
Upgrades & New Features	- Automatic access to the latest features and updates	- Full control over when and how to upgrade - Delayed to the latest features unless manually updated

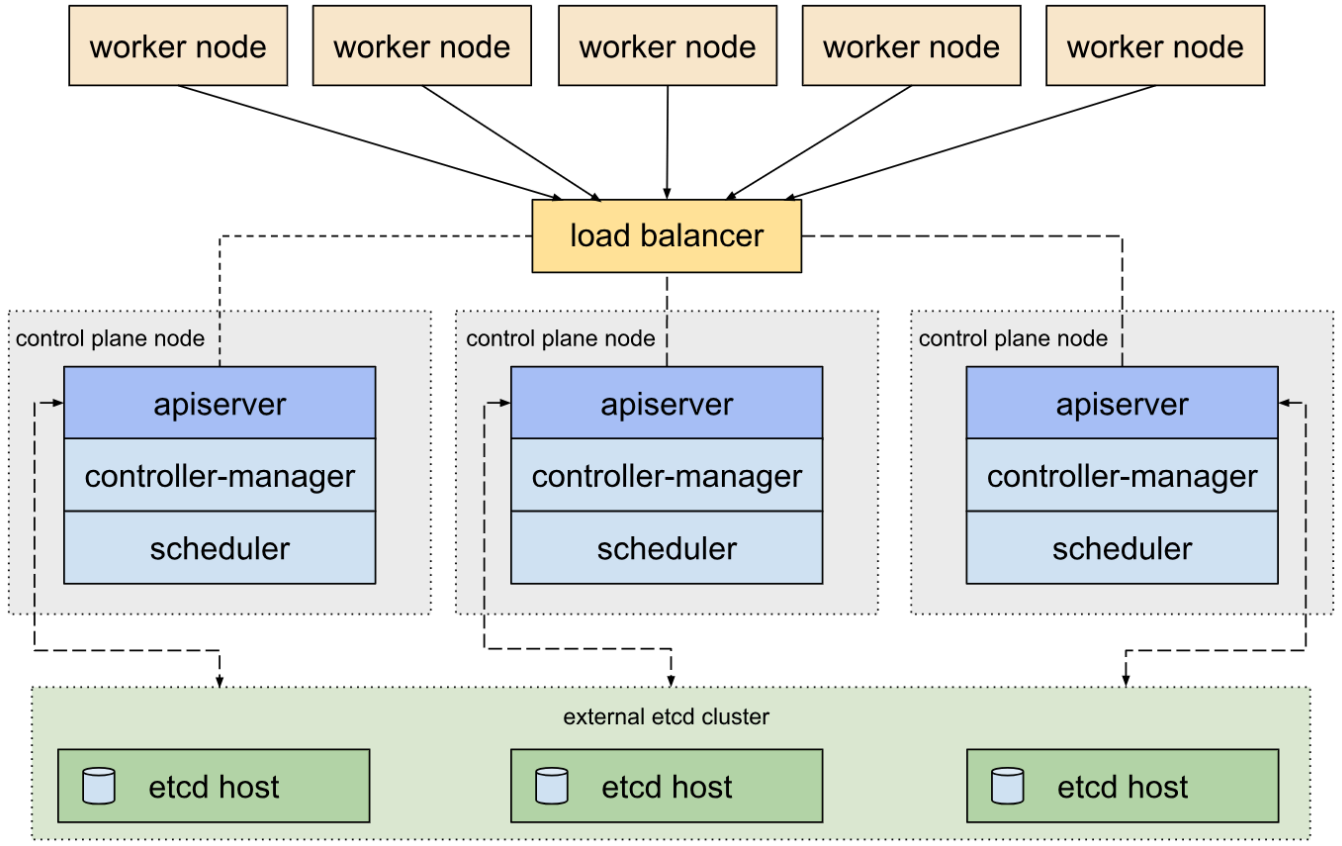
Options for Highly Available

- Stacked etcd topology
- External etcd topology

Stacked etcd topology



External etcd topology



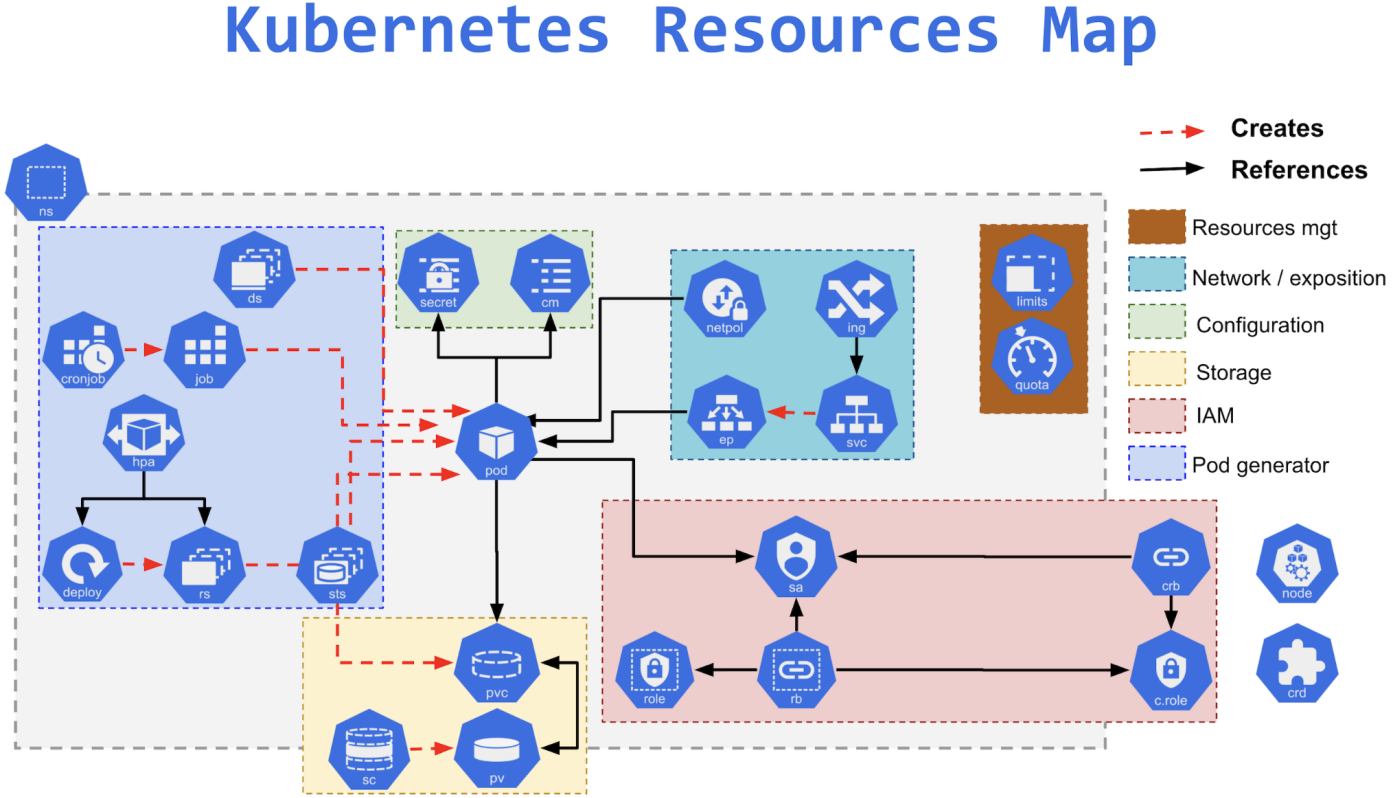
Fault Tolerance Table

Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	**1**
4	3	1
5	3	**2**
6	4	2
7	4	**3**
8	5	3
9	5	**4**

Resources / Objects

Resources / Objects

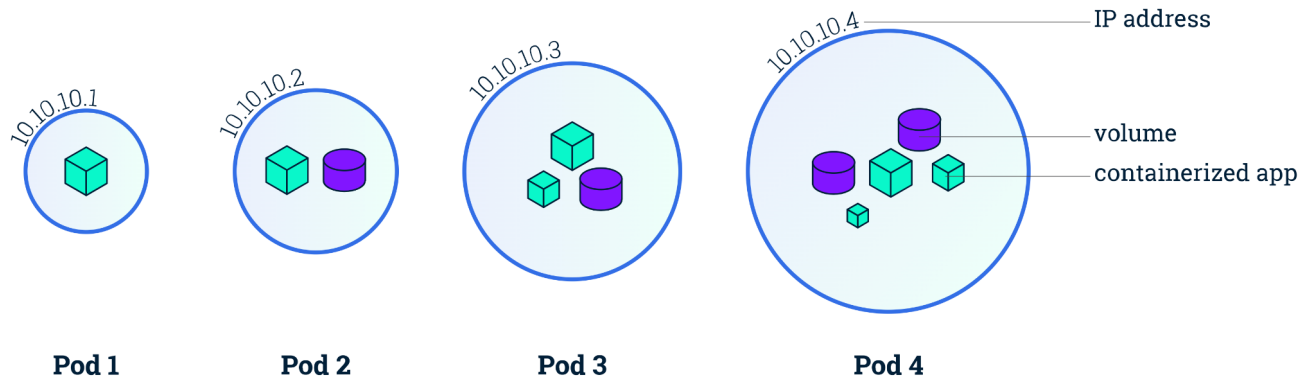
- Pod
- Deployment
- Service Ingress
- Config map
- Secret
- PV & PVC



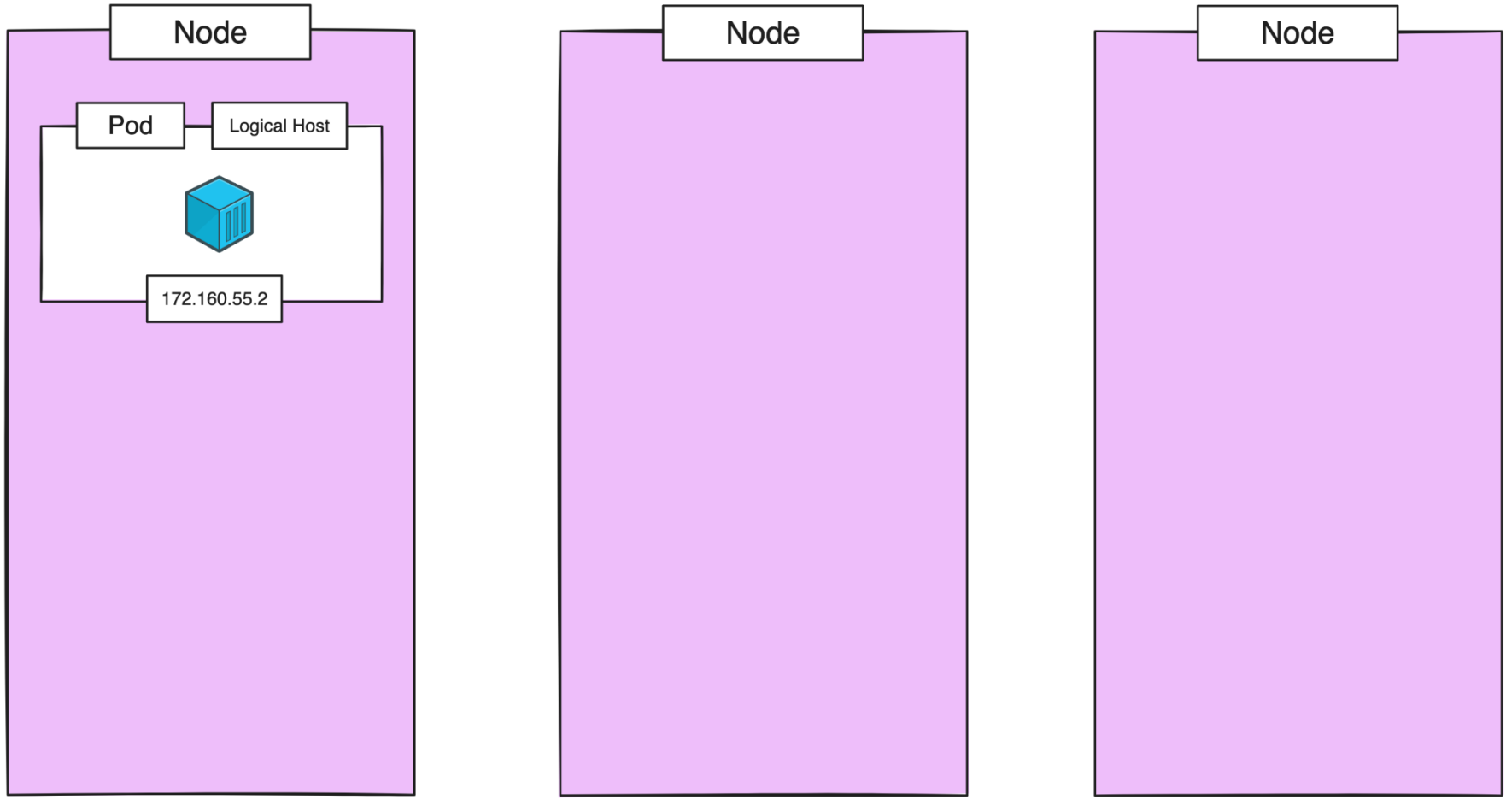
Pod

Pod

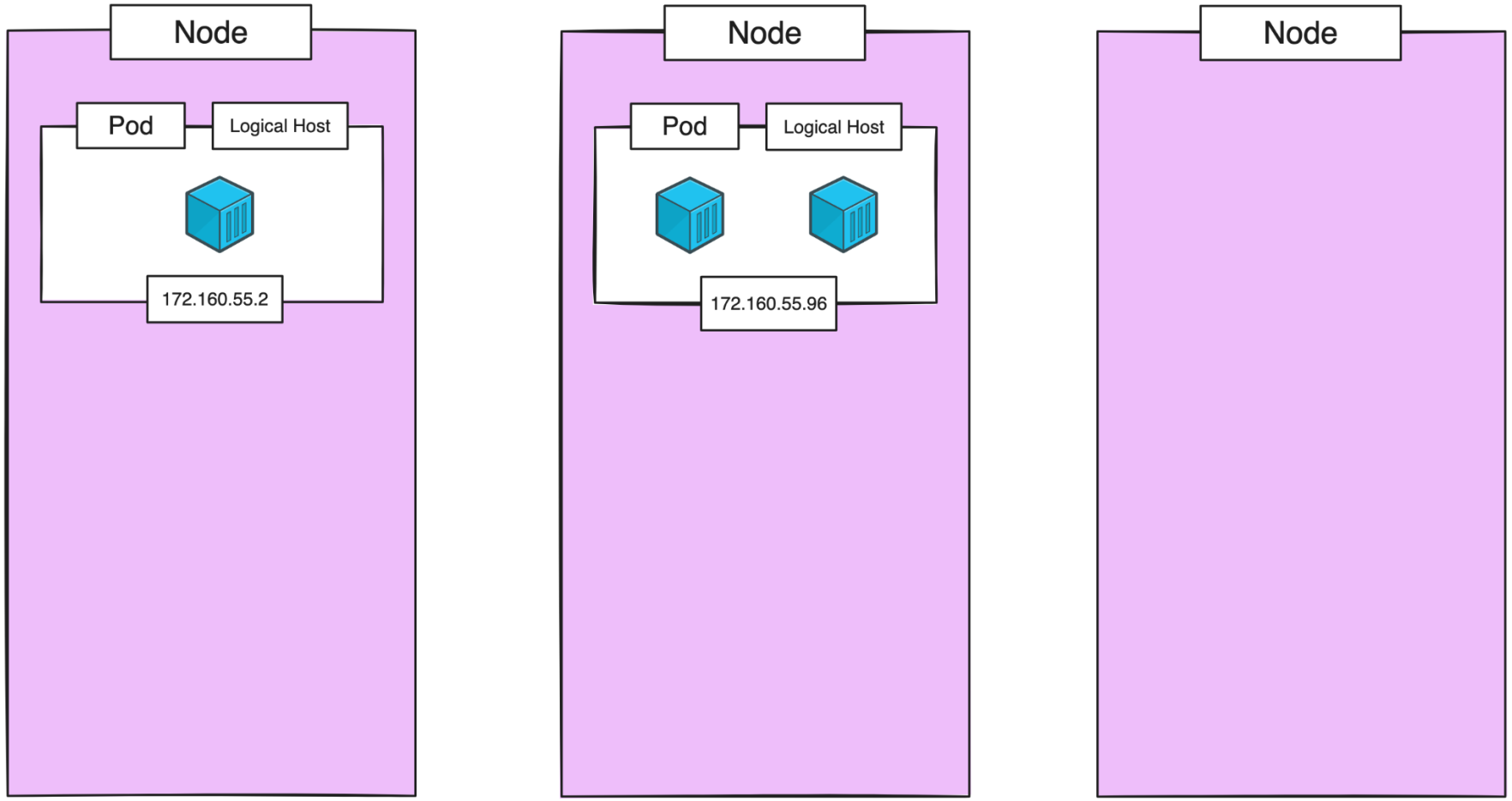
- Smallest deployable units of computing that you can create and manage in Kubernetes
- Logical Application
- One or more container in Pod (recommend 1 container per pod)
- Any containers in same pod will share storage volumes and network resources
- Each pod is assigned a unique ip



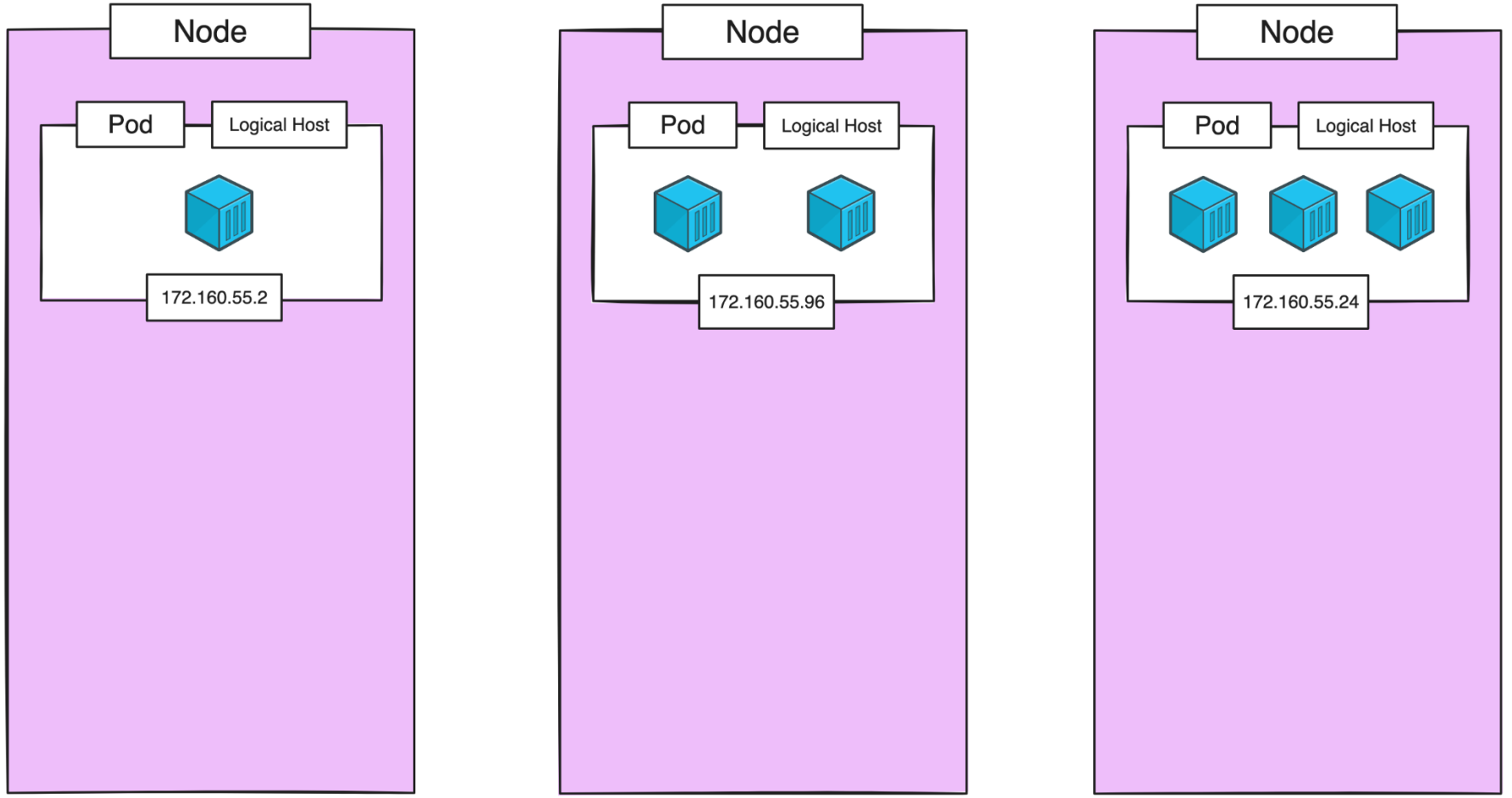
Node & Pod



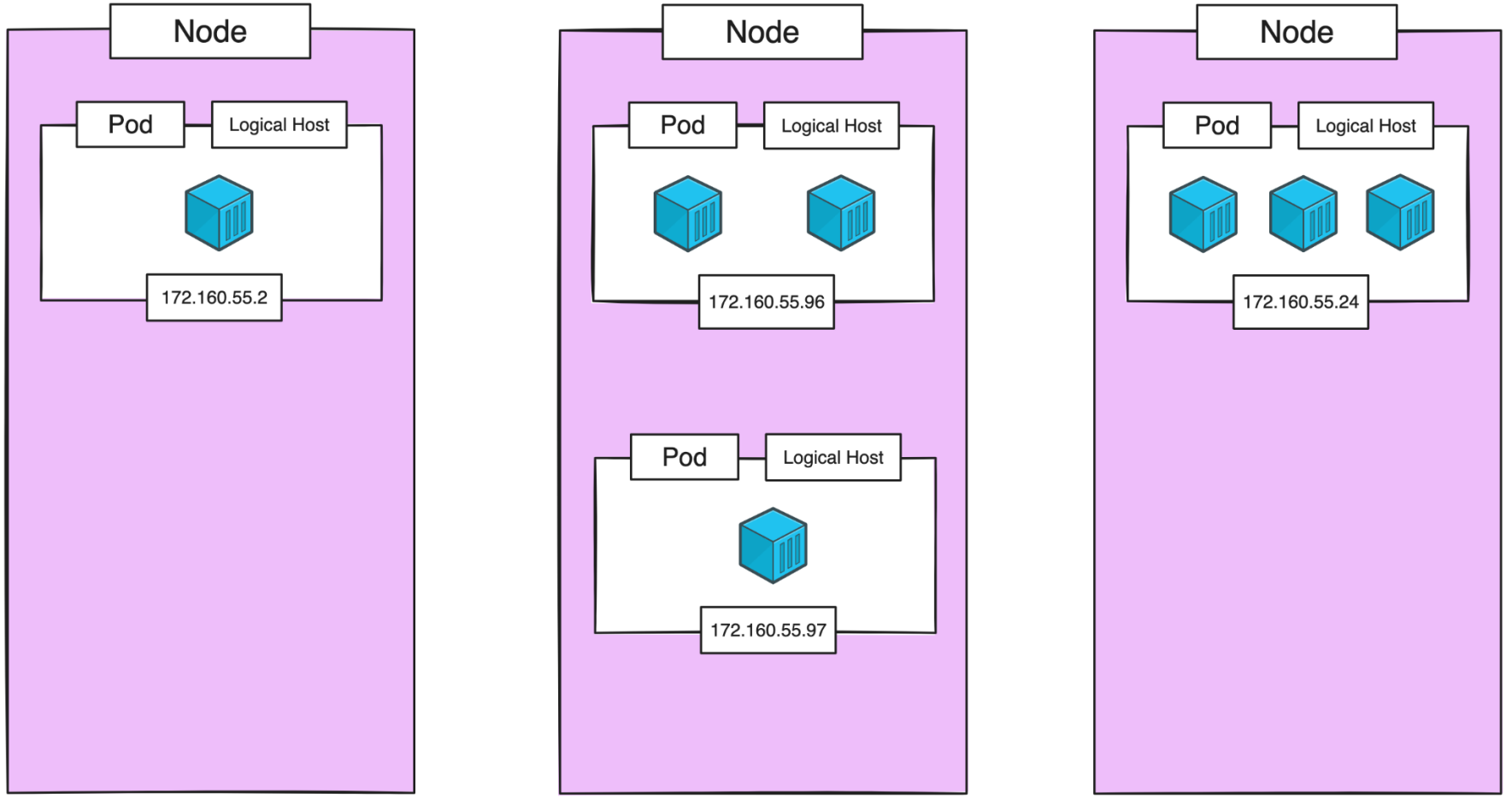
Node & Pod



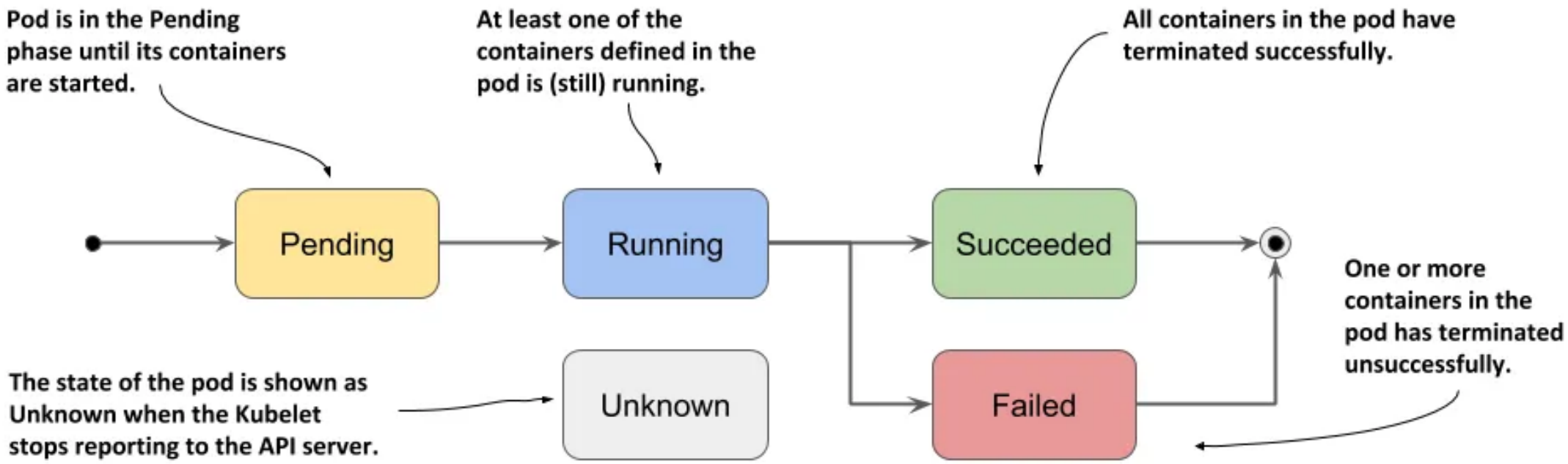
Node & Pod



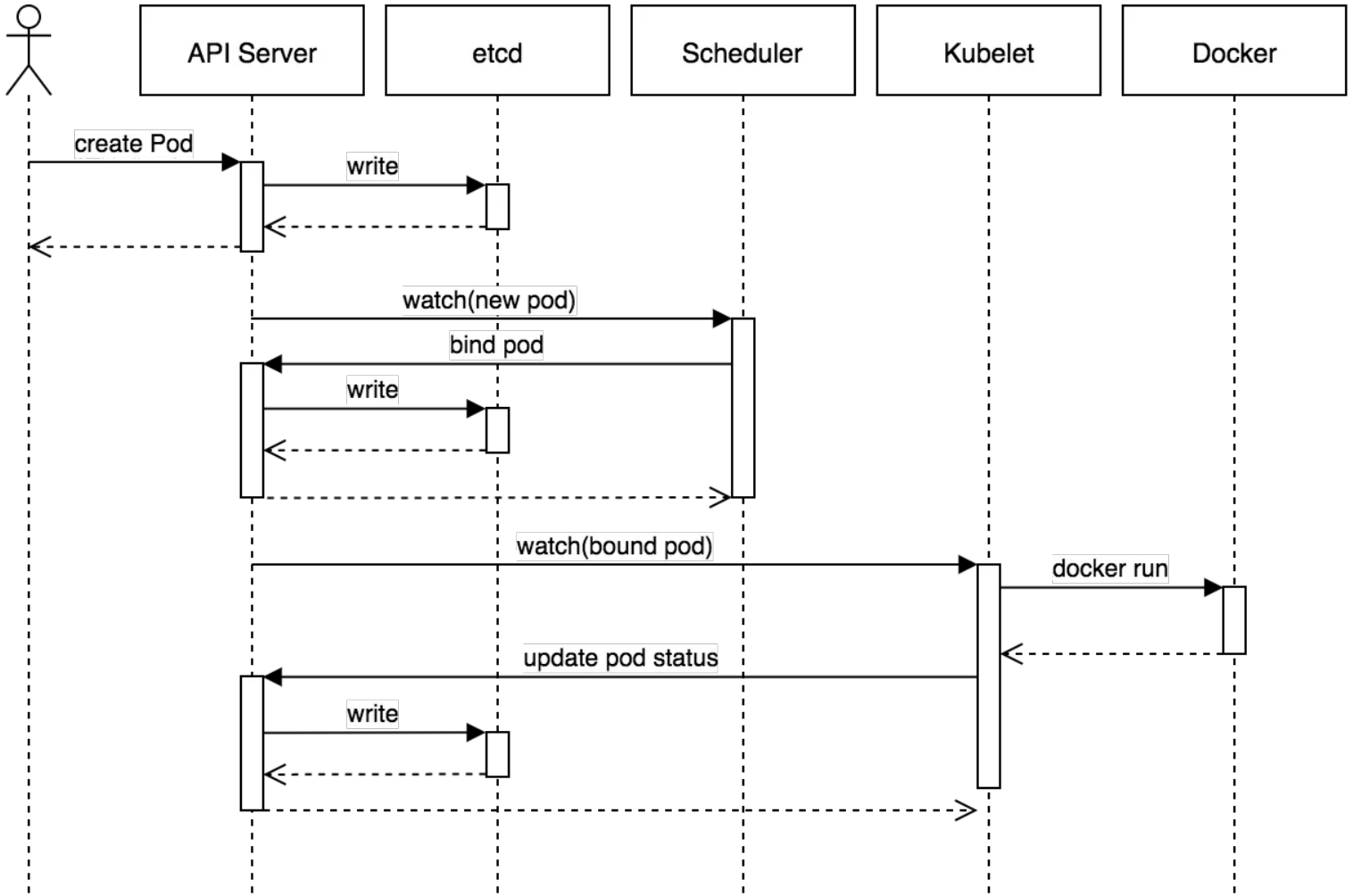
Node & Pod



State of Pod



Creation of Pod

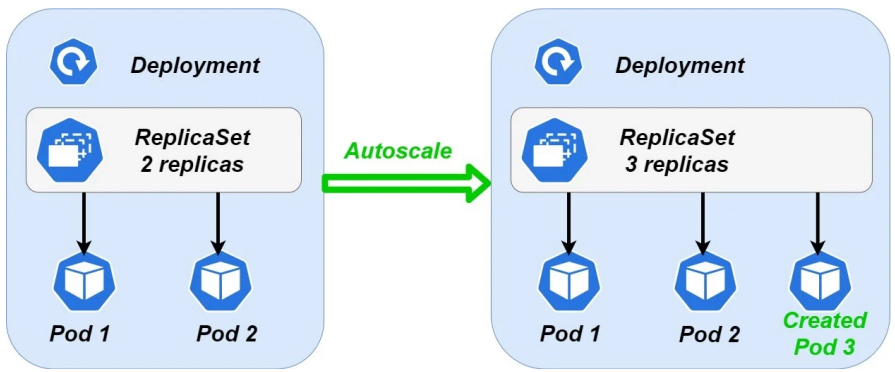


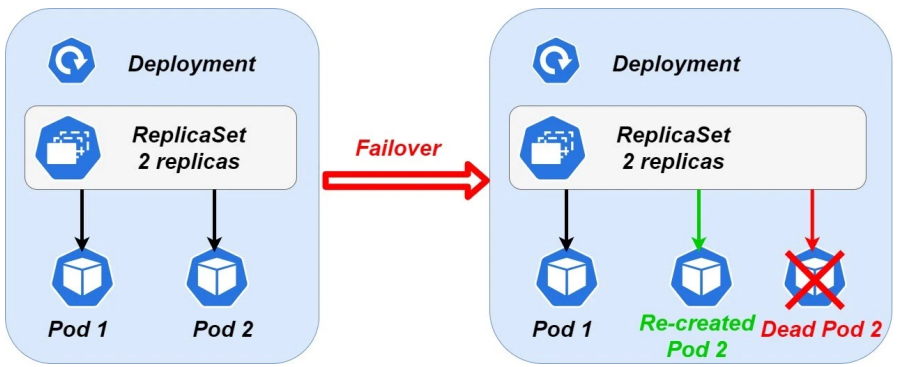
Workshop pod

Deployment

Deployment

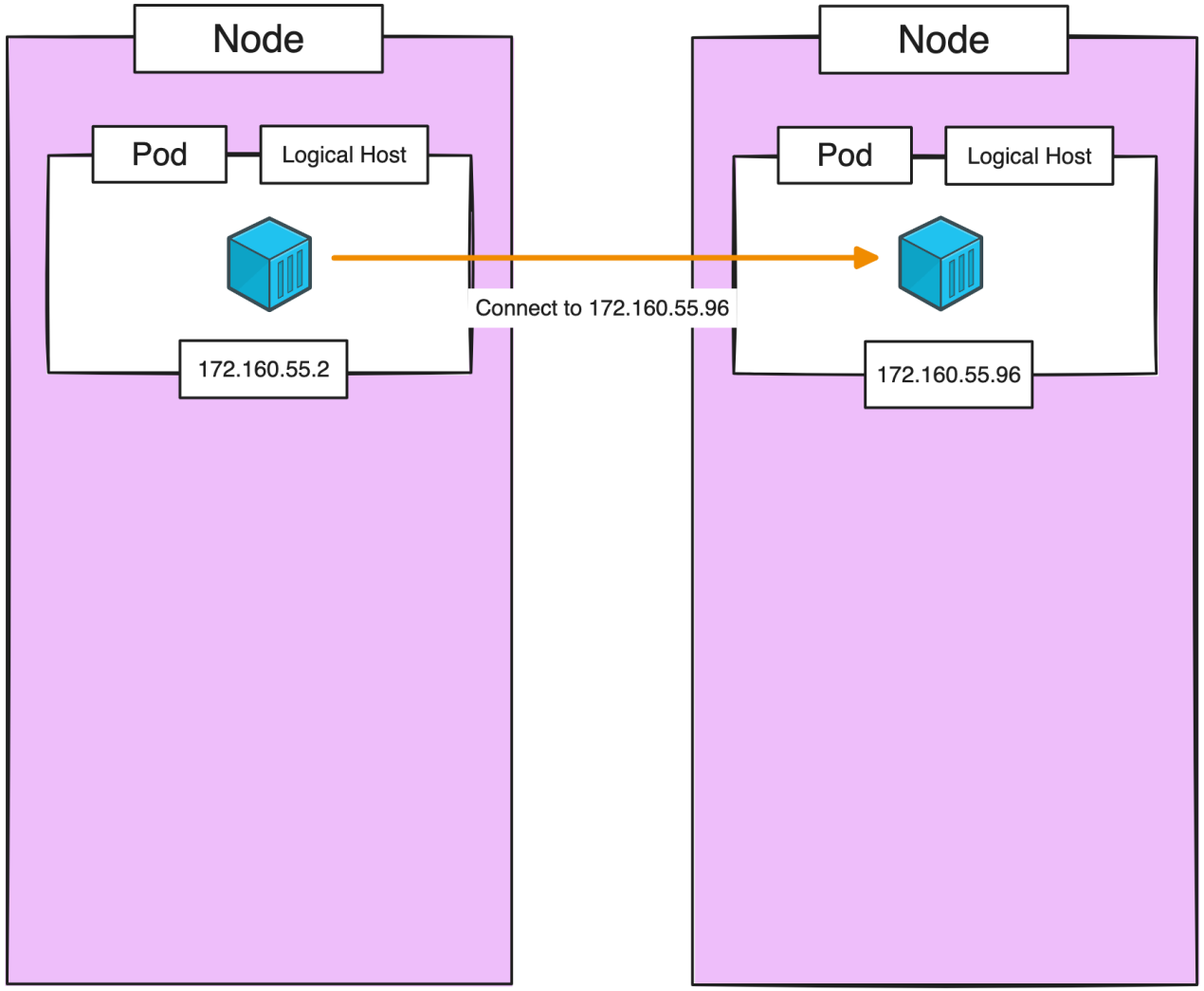
- A Deployment provides declarative updates for Pods and ReplicaSets
- You describe a **desired state** in a Deployment, and the Deployment Controller **changes the actual state to the desired state** at a controlled rate
- Manage for rollout and rollback
- Deployment => ReplicaSet => pod



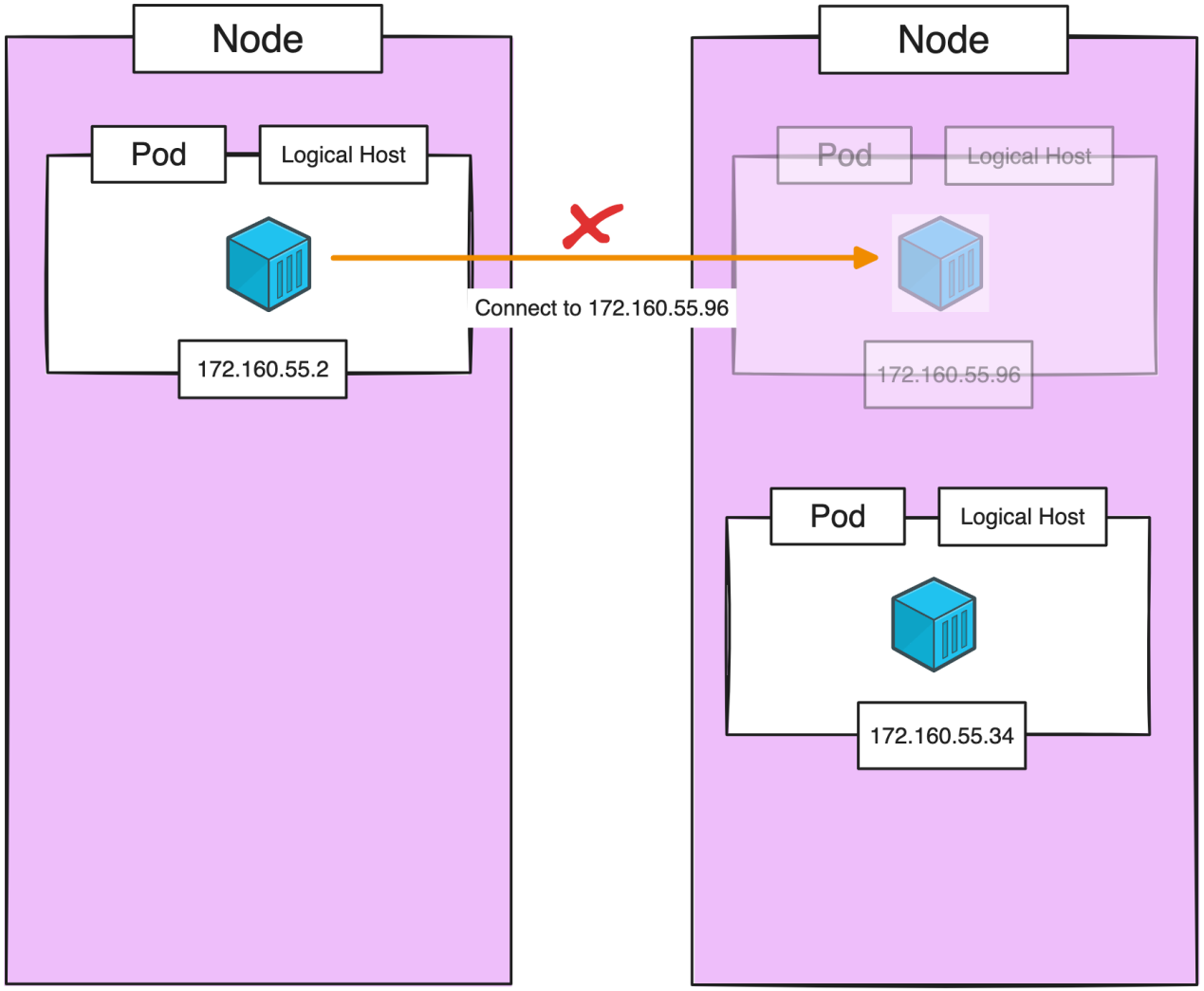


Workshop deployment

Problems



Problems

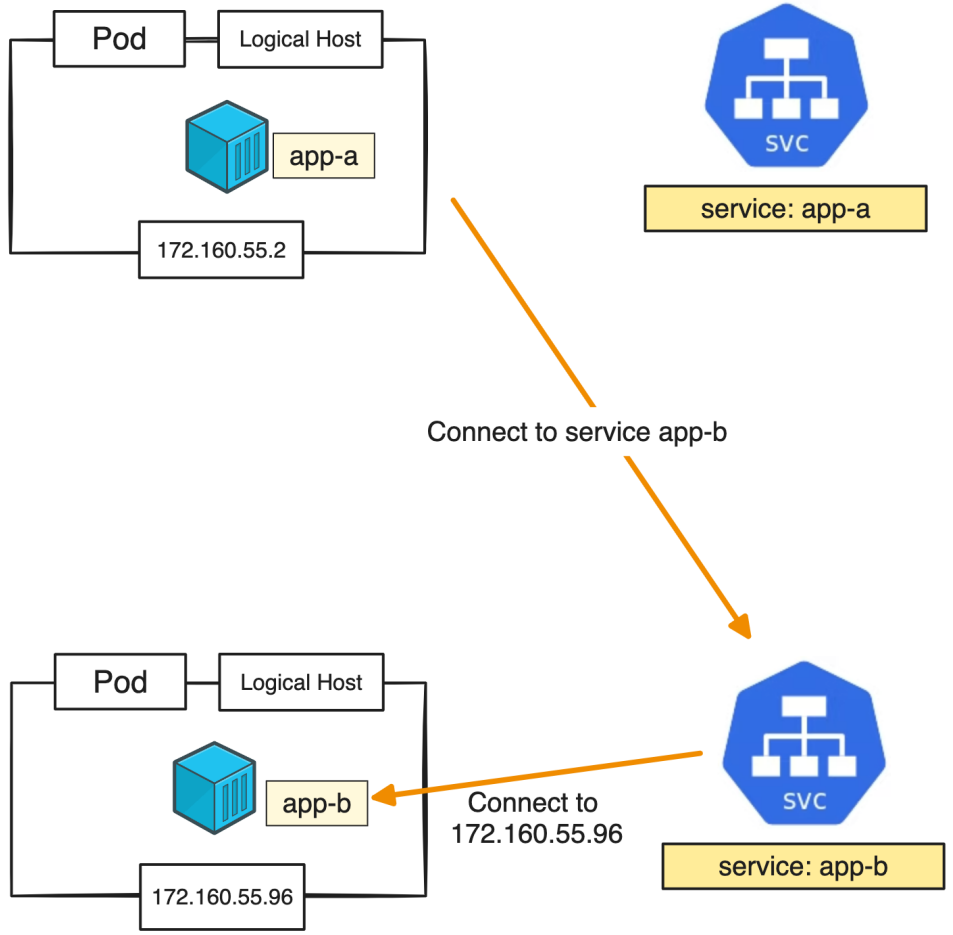


Service

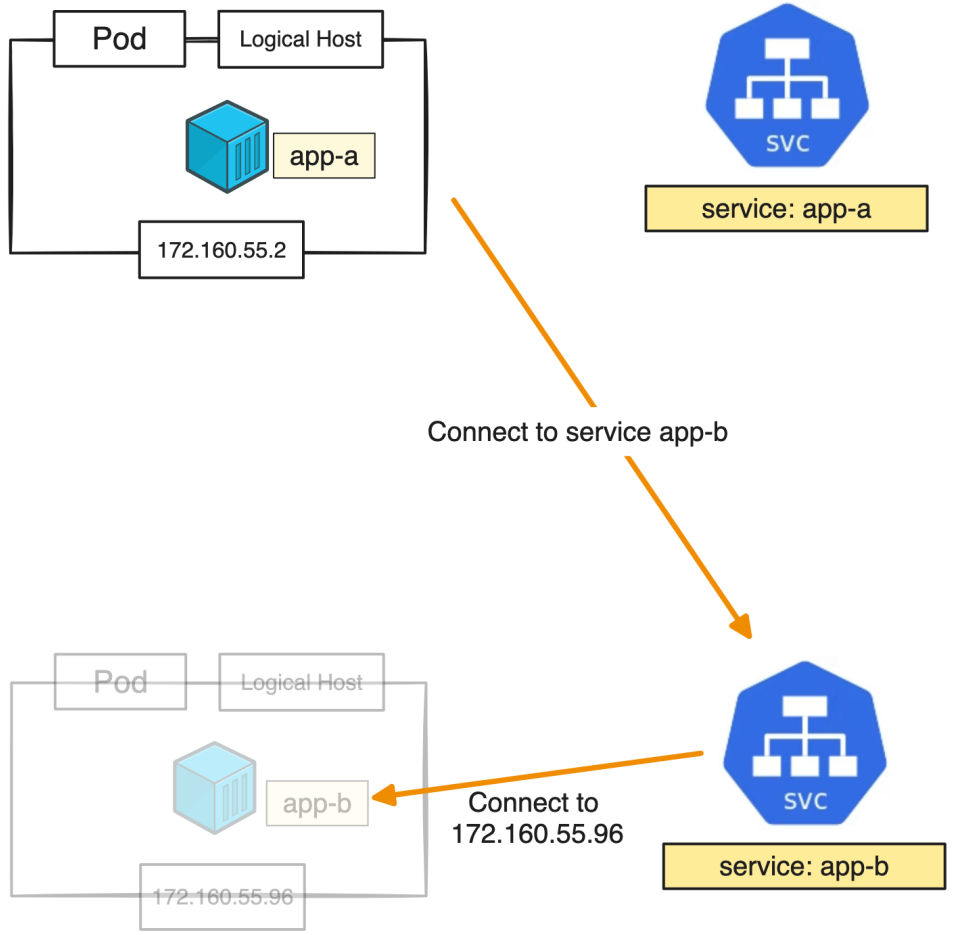
Service

- Acts as a virtual endpoint for your application
- Providing a stable and discoverable way to access your pods without worrying about their individual IP addresses or network details
- Logical load balancer
- Communicating for outside and inside of cluster
- 4 types:
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName

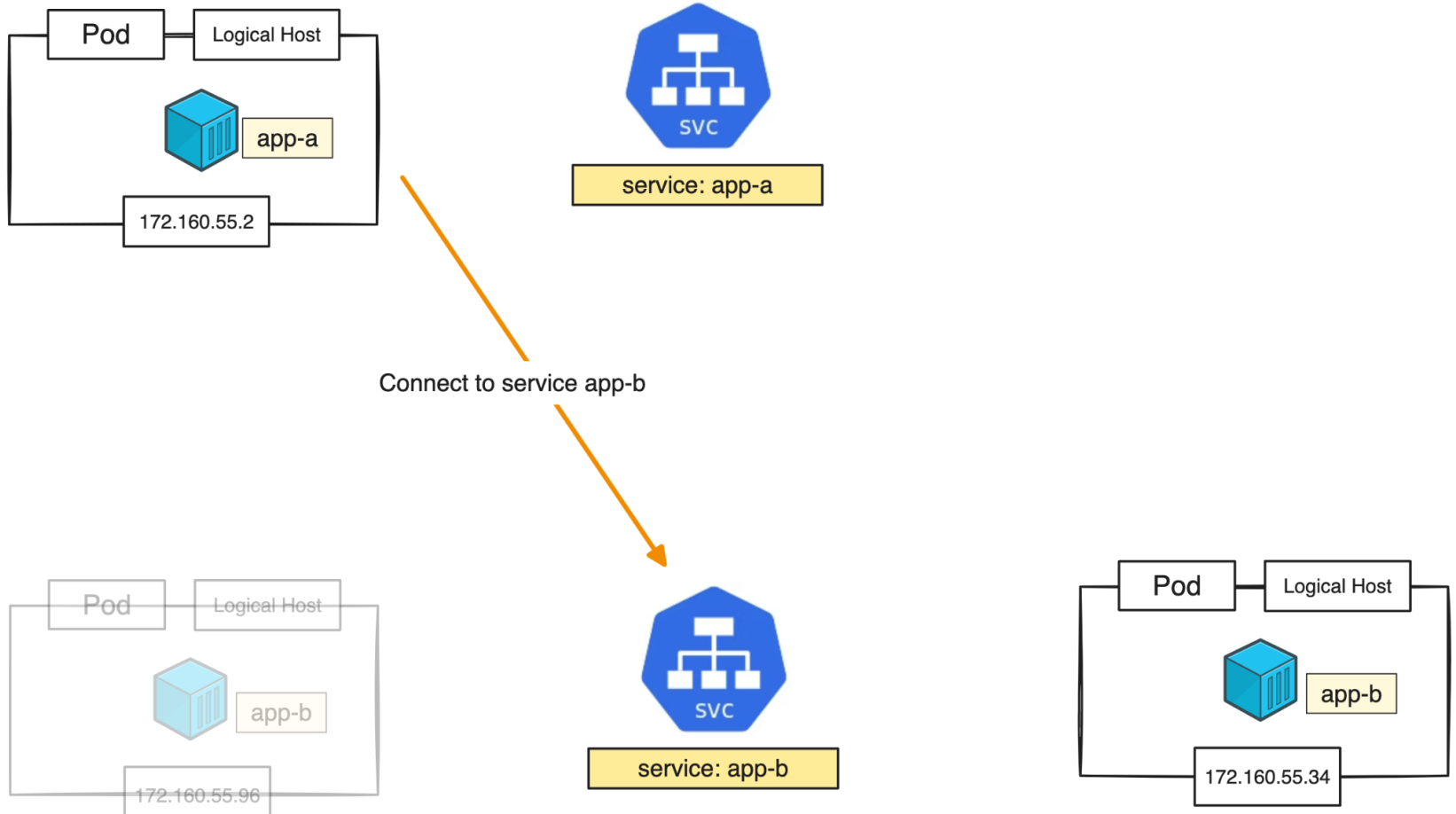
Service: Stable endpoint



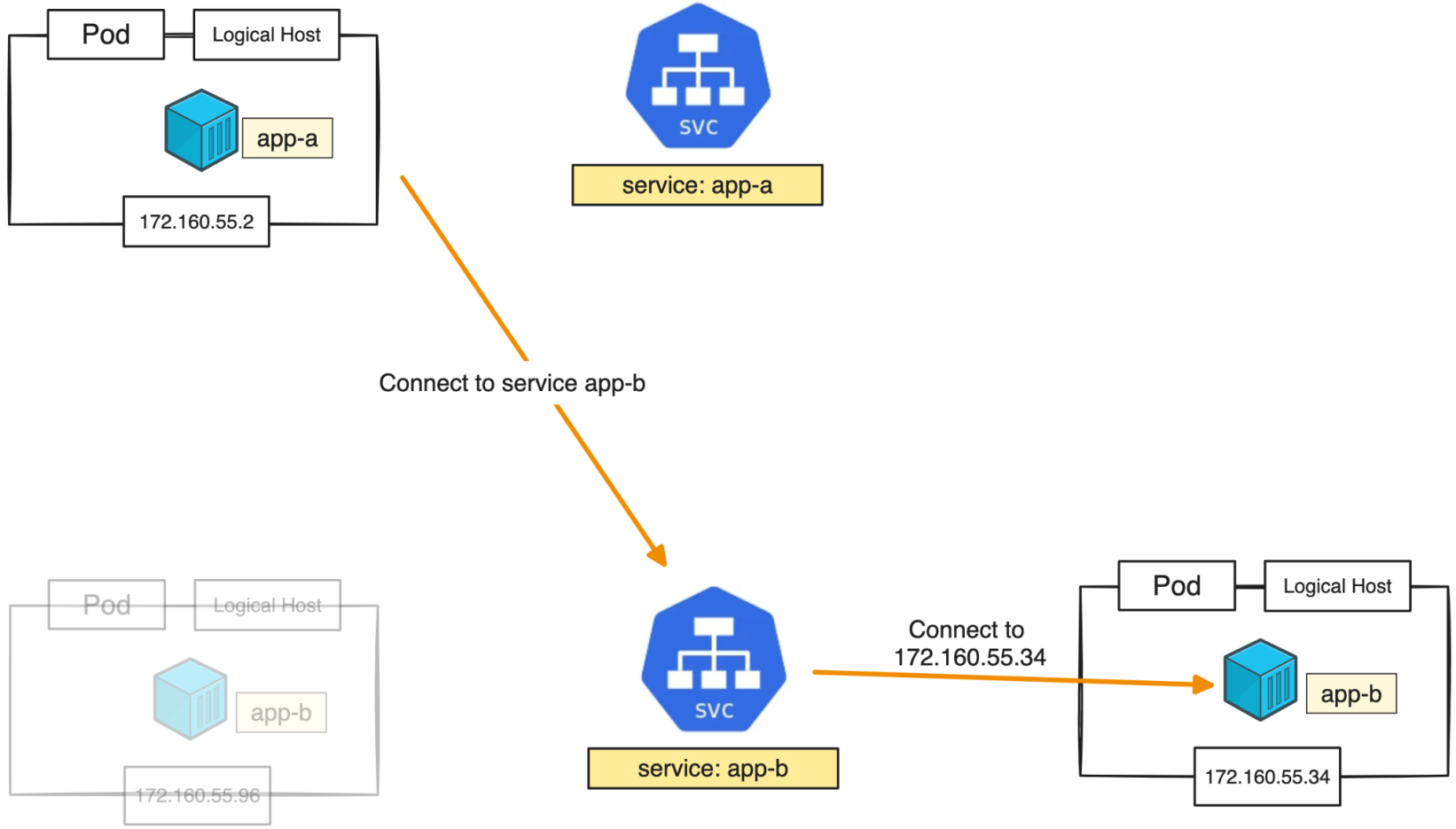
Service: Stable endpoint



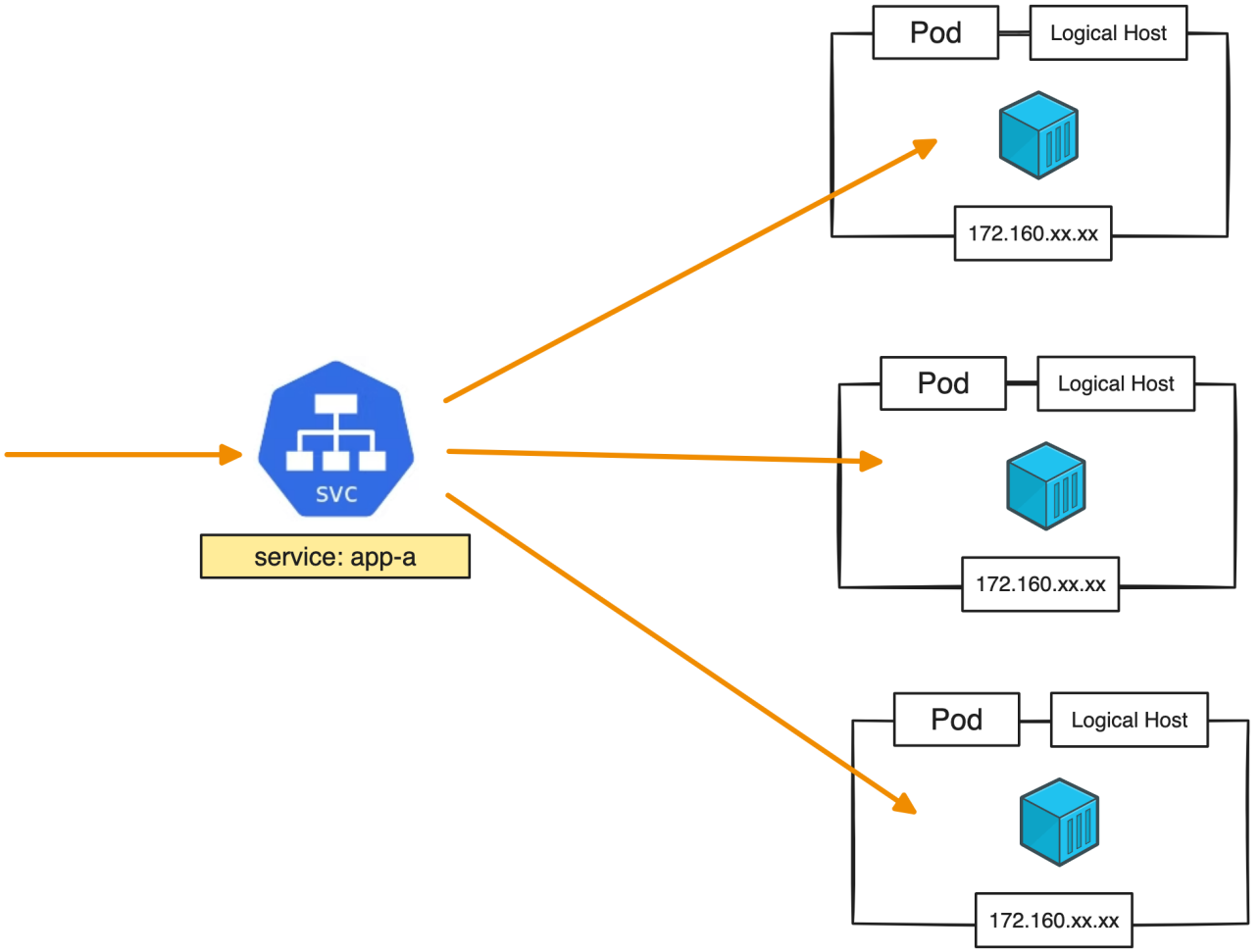
Service: Stable endpoint



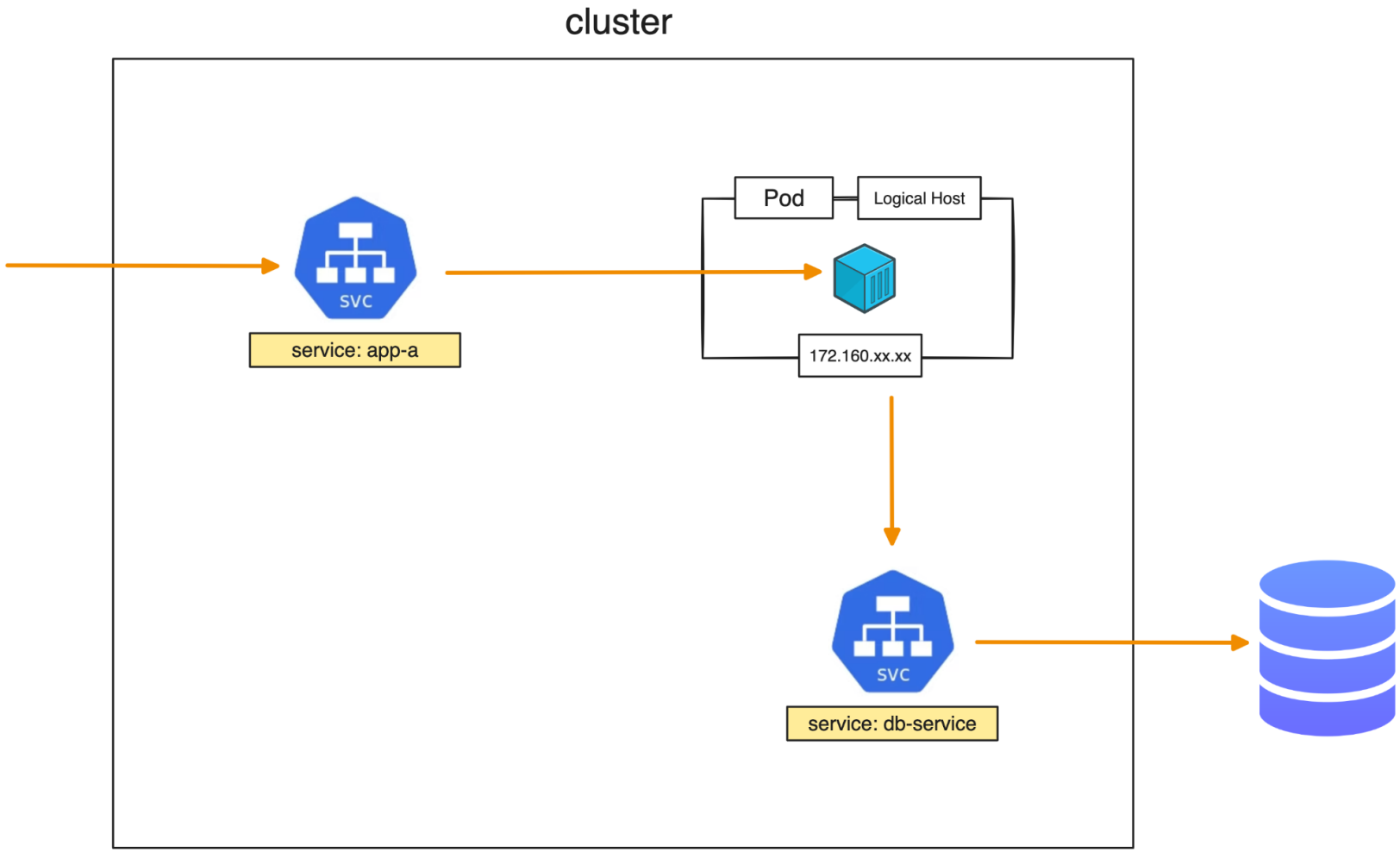
Service: Stable endpoint



Service: Load balancer

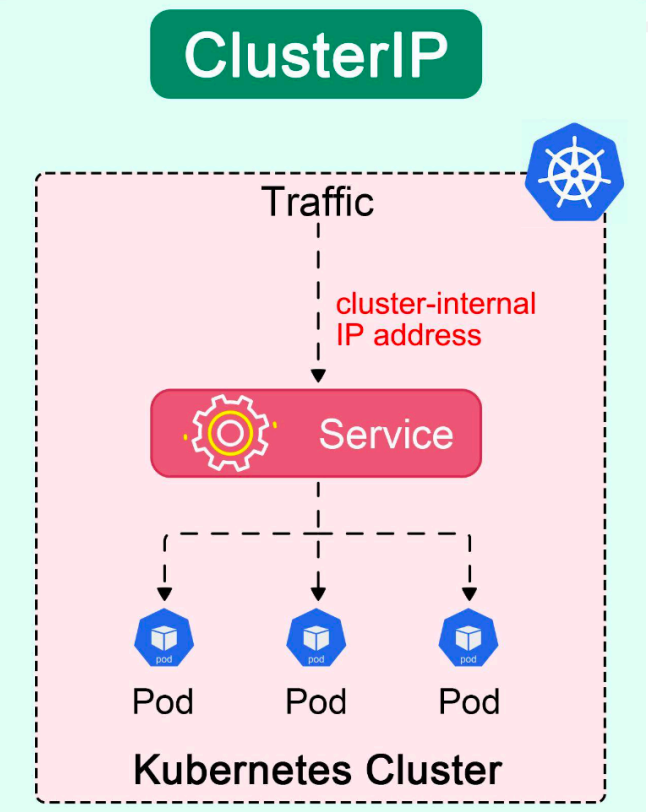


Service: Communicating inside & outside of cluster



Service type: ClusterIP

- ClusterIP is the default service type in Kubernetes
- It exposes the service on an internal IP address reachable **only within** the cluster



Advantages

- Suitable for inter-pod communication and internal services

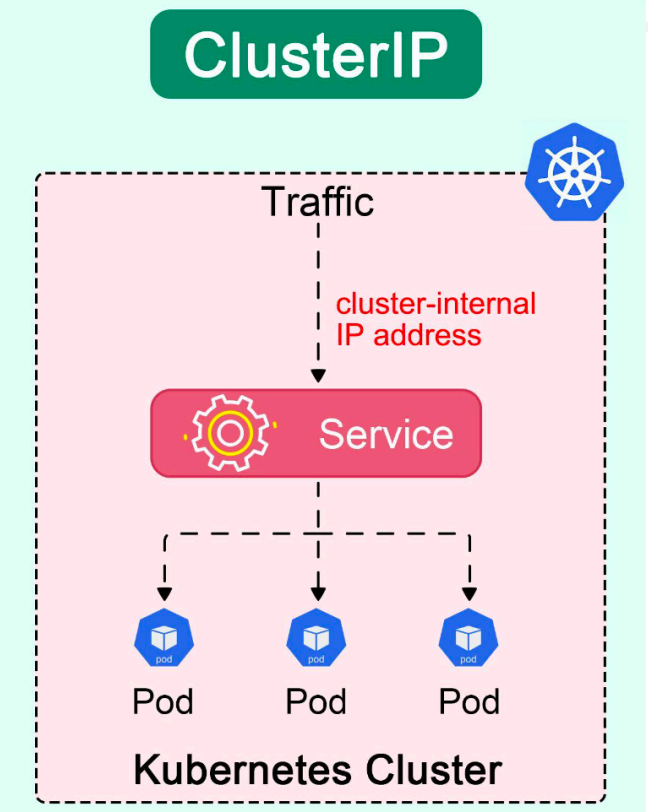
Disdvantages

- Limited to internal cluster communication
- Not accessible externally

Service type: ClusterIP

Use Cases

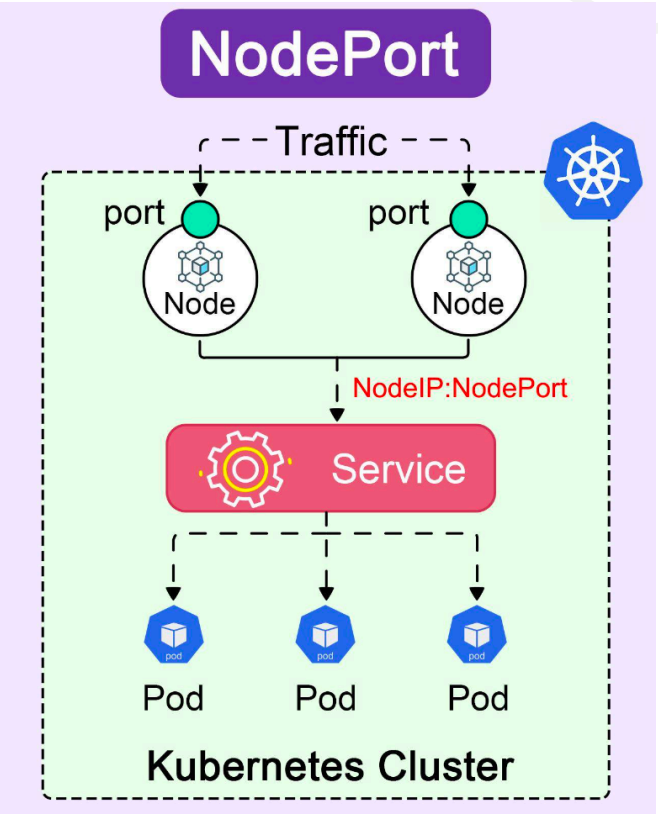
- Inter-service communication within the cluster. For example, communication between the front-end and back-end components of your app



Workshop ClusterIP Service

Service type: NodePort

- It enables **external access** to the service using the cluster nodes' **IP address** and the assigned **NodePort**
- When a NodePort service is created, Kubernetes allocates a port from a predefined range (default: **30000-32767**)



Advantages

- Provides external access to the service
- Suitable for development and testing environments

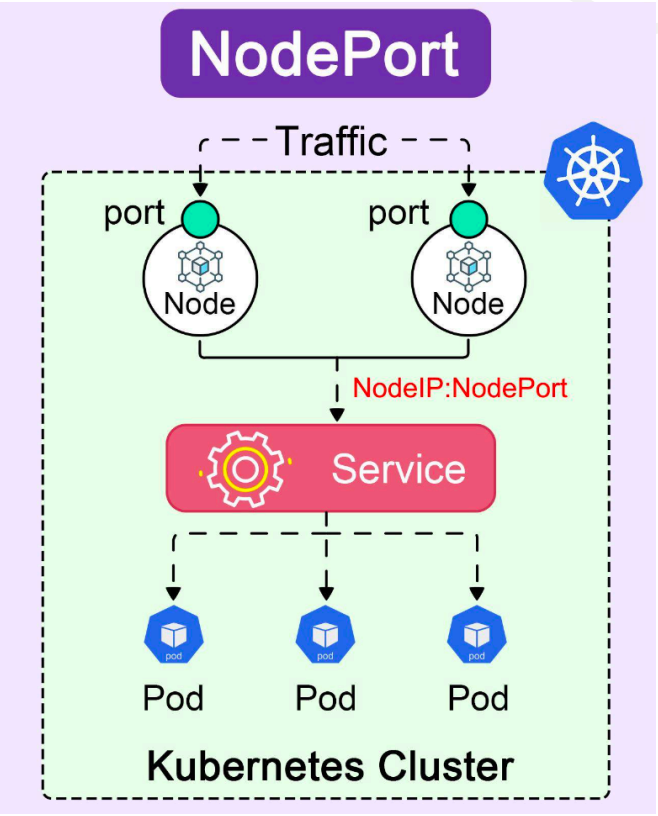
Disdvantages

- Requires manual management of port allocations
- IP Node maybe have changed
- Should not use in production

Service type: NodePort

Use case

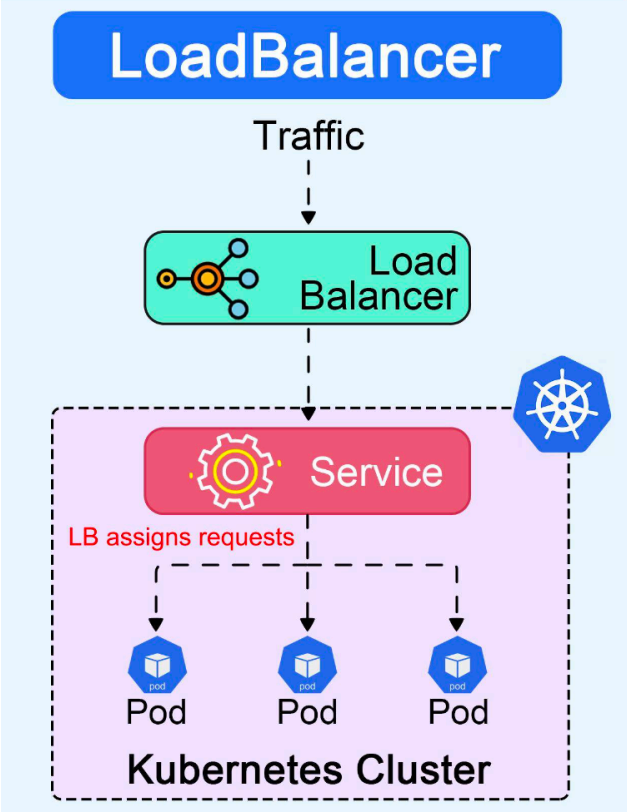
- When you want to enable external connectivity to your service
- Best for testing access for a moment time



Workshop NodePort Service

Service type: LoadBalancer

- LoadBalancer is a service type that **provisions** an **external load balancer** to distribute traffic across the pods
- Typically provided by a cloud provider's infrastructure
- When a LoadBalancer service is created, Kubernetes interacts with the cloud provider's API to an external load balancer



Advantages

- Provides external access to the service
- Automatically provisions an external load balancer for the service
- Supports scaling and high availability

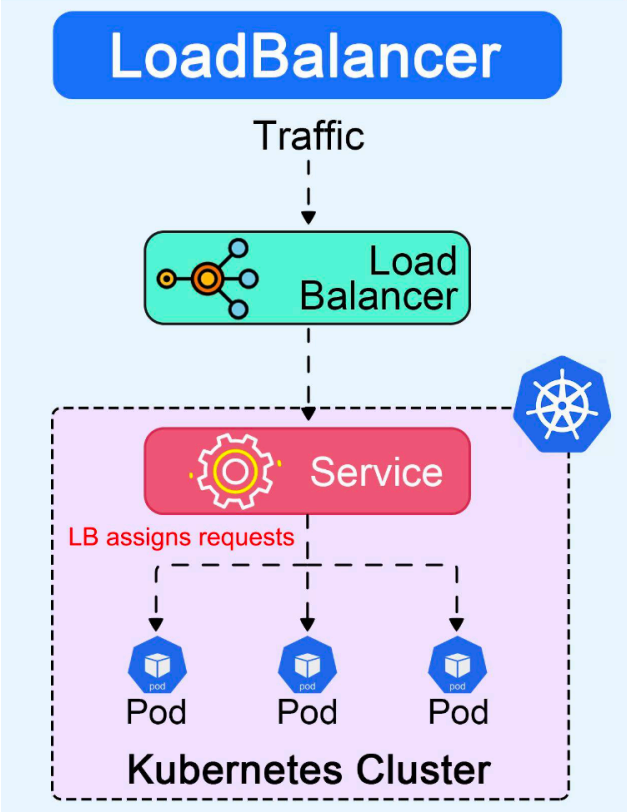
Disdvantages

- Requires a cloud provider
- Additional costs for using the load balancer service on cloud provider

Service type: LoadBalancer

Use case

- When you want to enable external connectivity to your service
- When you are using a cloud provider to host your Kubernetes cluster



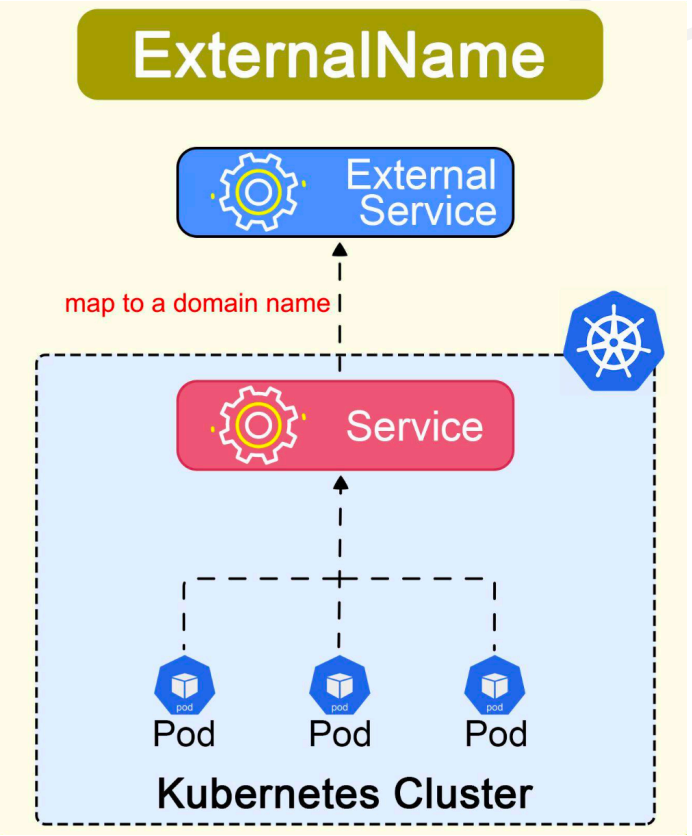
Workshop LoadBalancer Service

Service type: ExternalName

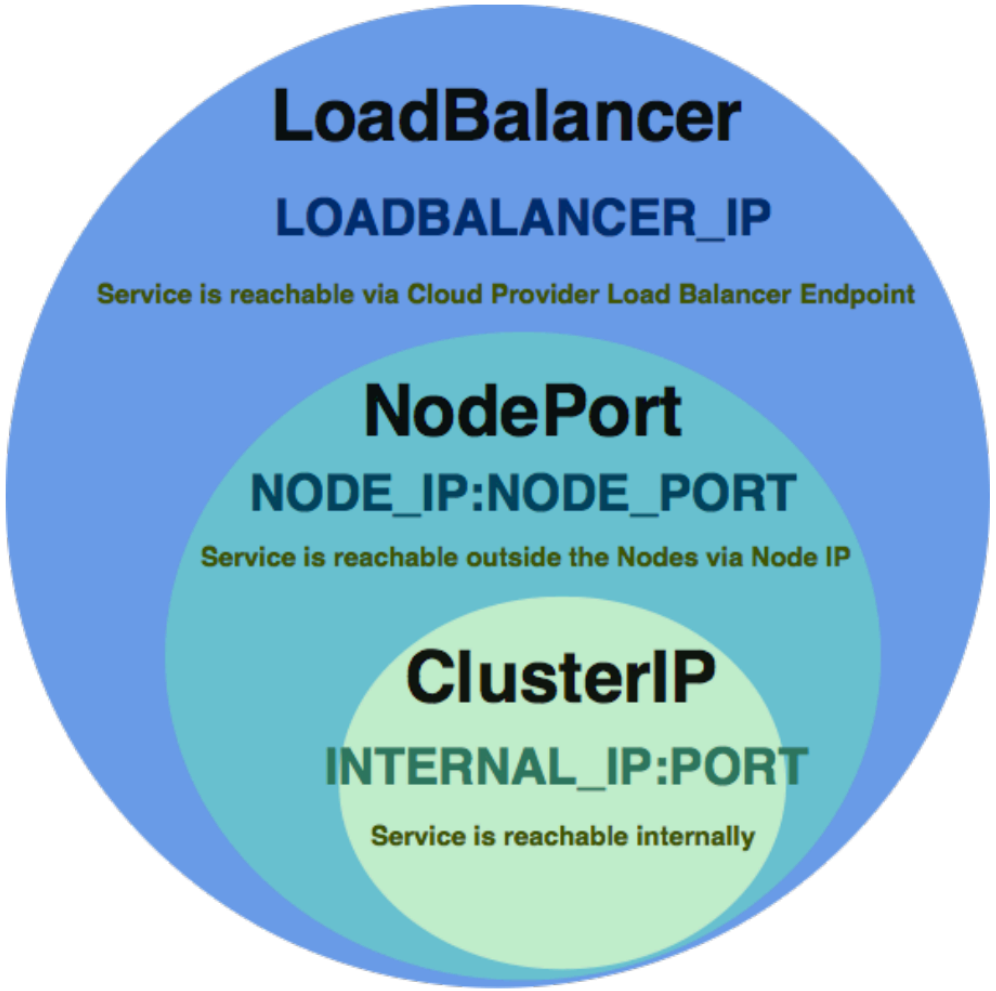
- An ExternalName Service is a special type of Kubernetes service that allows you to expose an existing external service (like a database or a third-party API) to your cluster without deploying any Pods

Use cases

- Accessing an external database to your Kubernetes cluster
- Connecting a third-party API from within your cluster



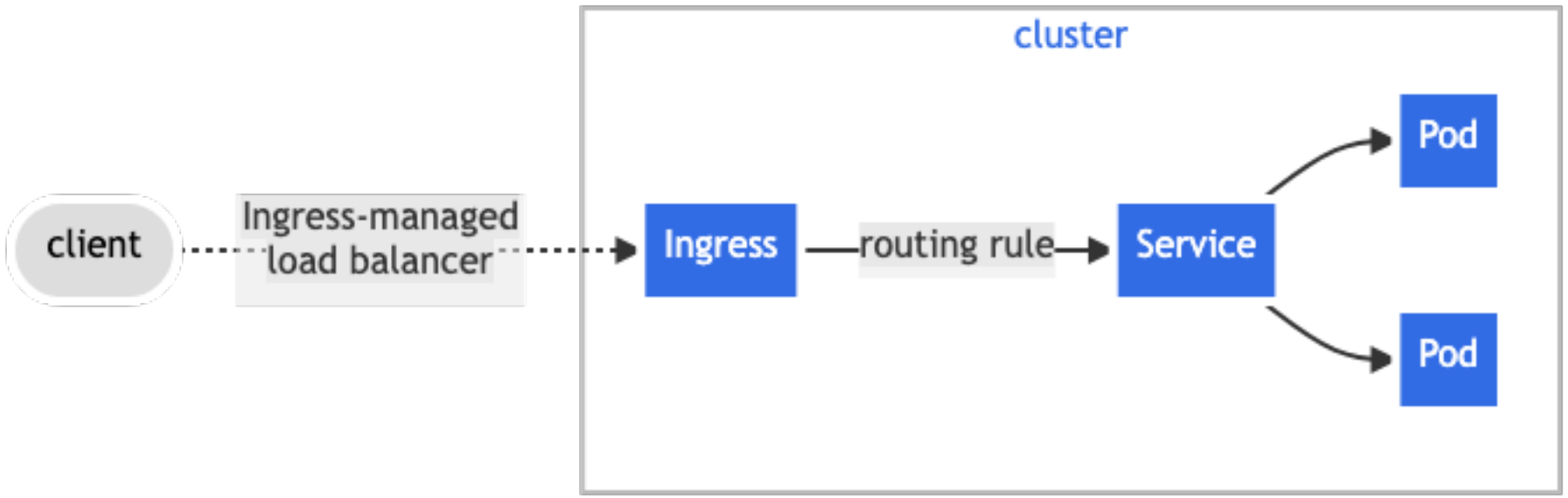
Service type



Ingress

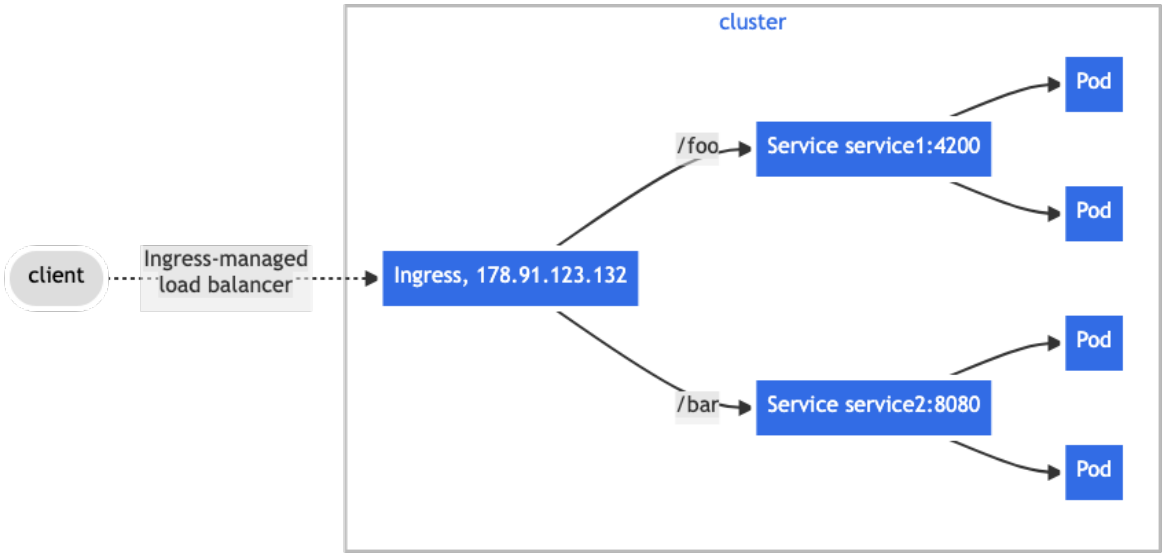
Ingress

- Routing from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the ingress resource
- An **Ingress controller** is responsible for fulfilling the Ingress



Ingress Controllers

- The Ingress Controller is an application hosted inside a Kubernetes cluster that actively **manages traffic** following Ingress Resources and pre-defined Ingress Rules
- Ingress controllers doesn't come with standard, have to be deployed separately
- Popular controllers include ingress-nginx, HAProxy, Traefik

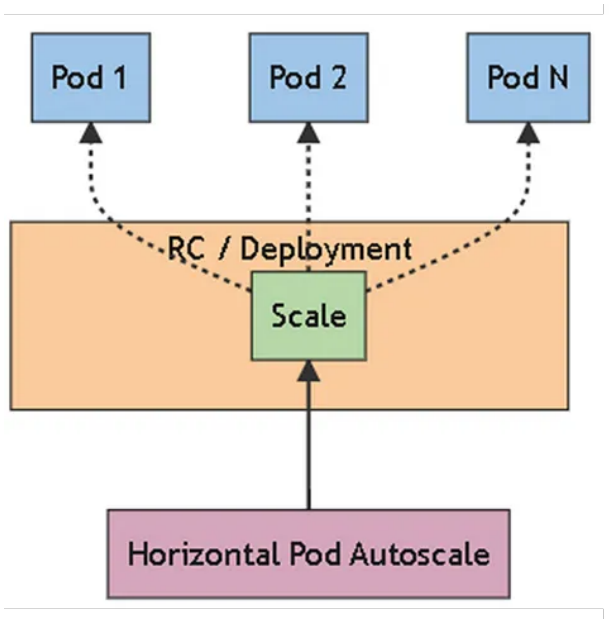


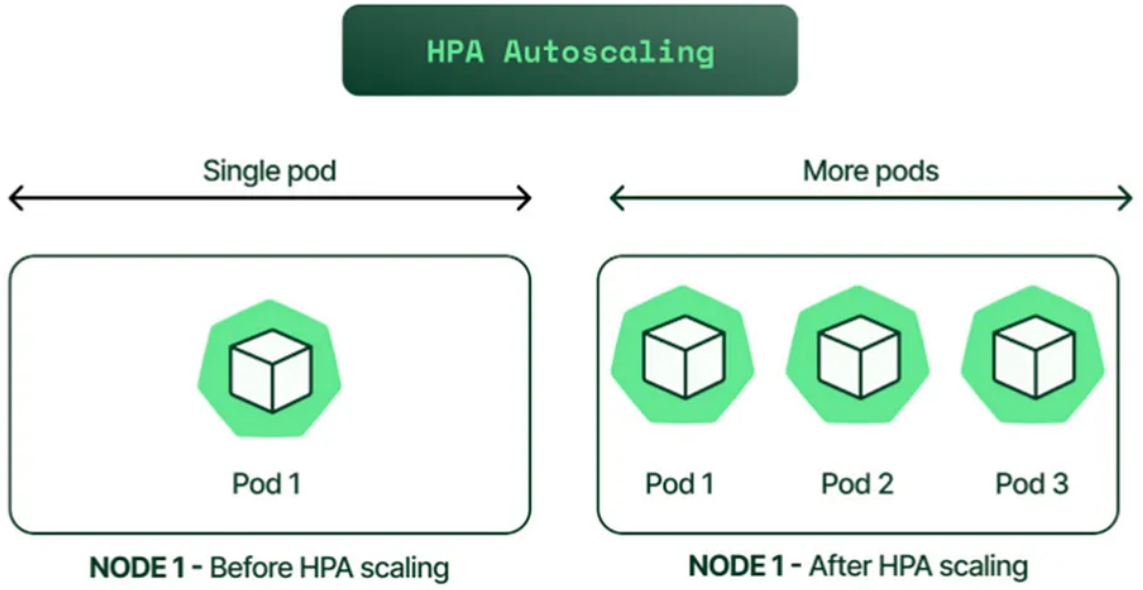
Workshop Ingress

HPA (Horizontal Pod Autoscaler)

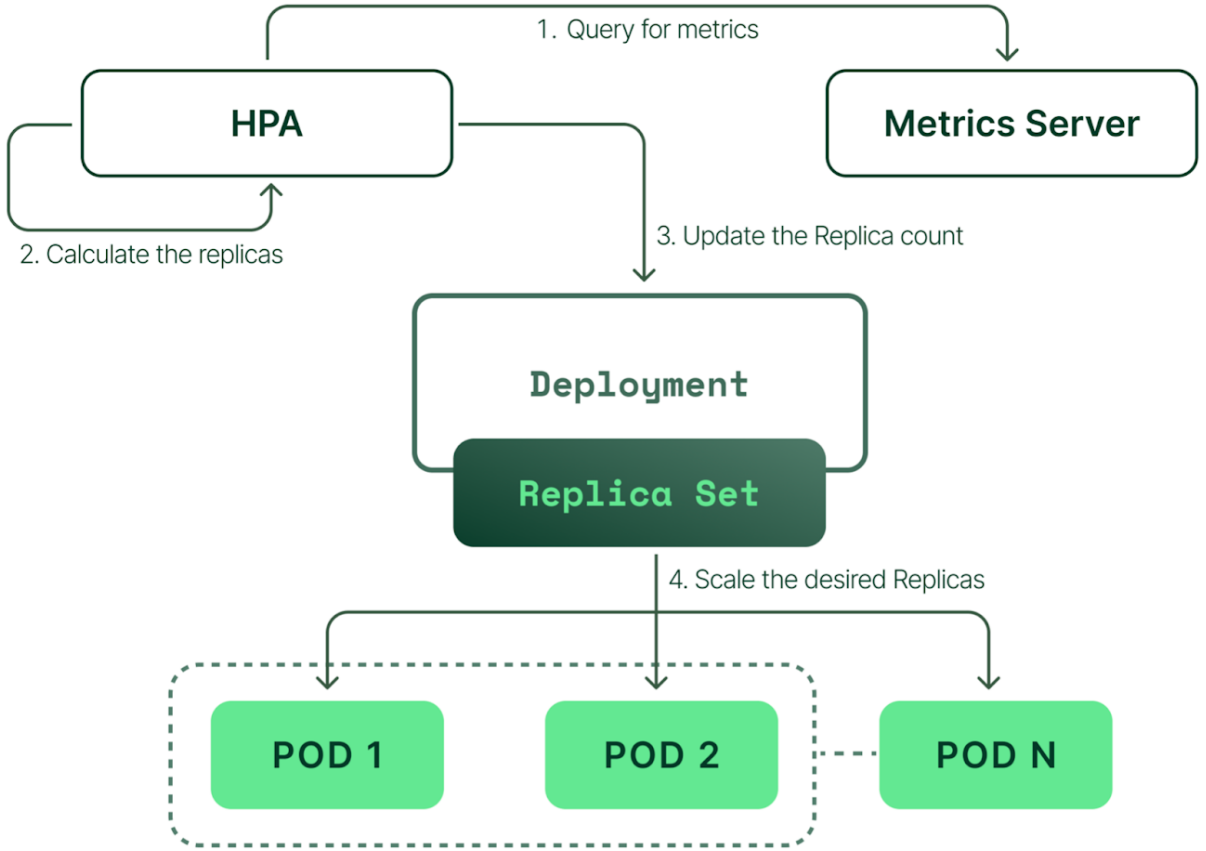
HPA

- A HorizontalPodAutoscaler (HPA for short) automatically updates a workload resource, with the aim of automatically scaling the workload to match demand





How does HPA work?



Workshop HPA

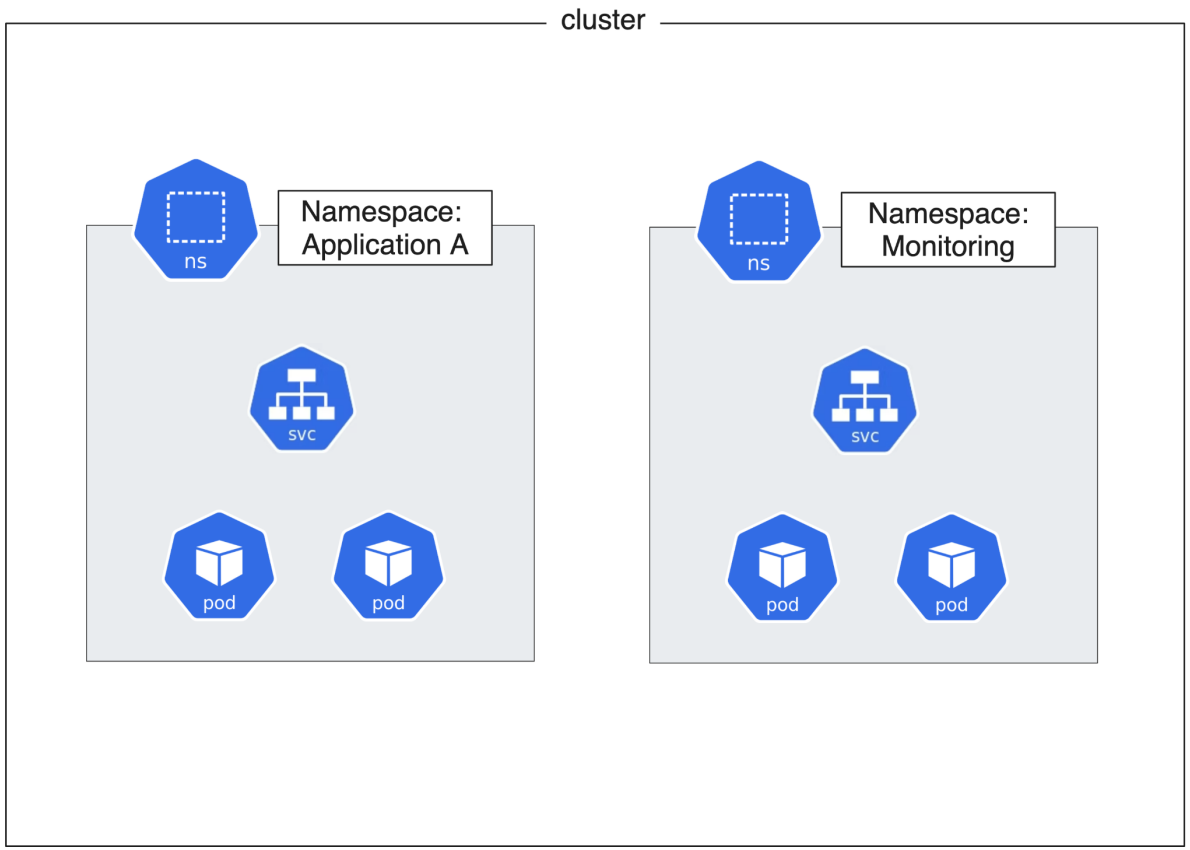
Namespace

Namespace

- Partition a single kubernetes cluster into multiples clusters
- 4 defaults namespaces
  - **kube-system**: core system processes
  - **kube-public**: public accessible data, ex: ConfigMaps & Secrets
  - **kube-node-lease**: heartbeat of nodes, so that the control plane can detect node failure
  - **default**: your resources/objects

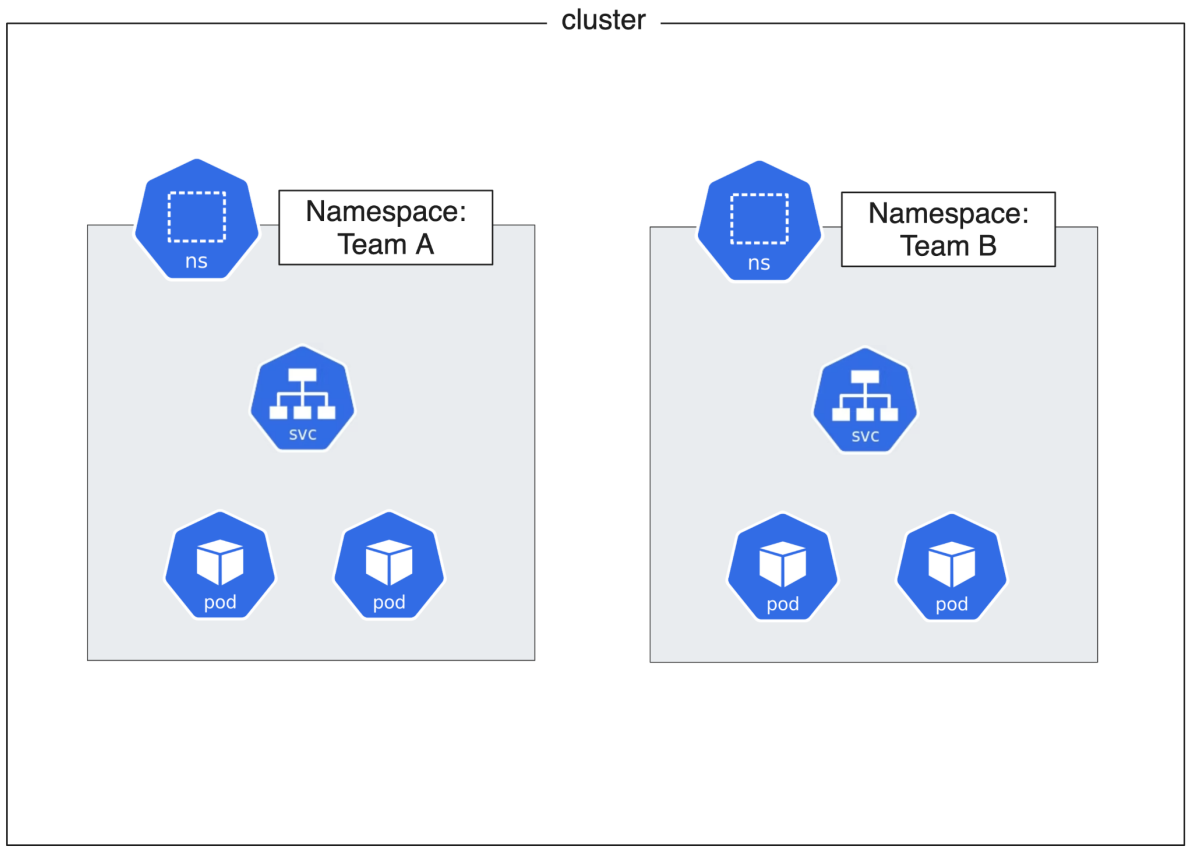
Namespace use cases

- Application domain



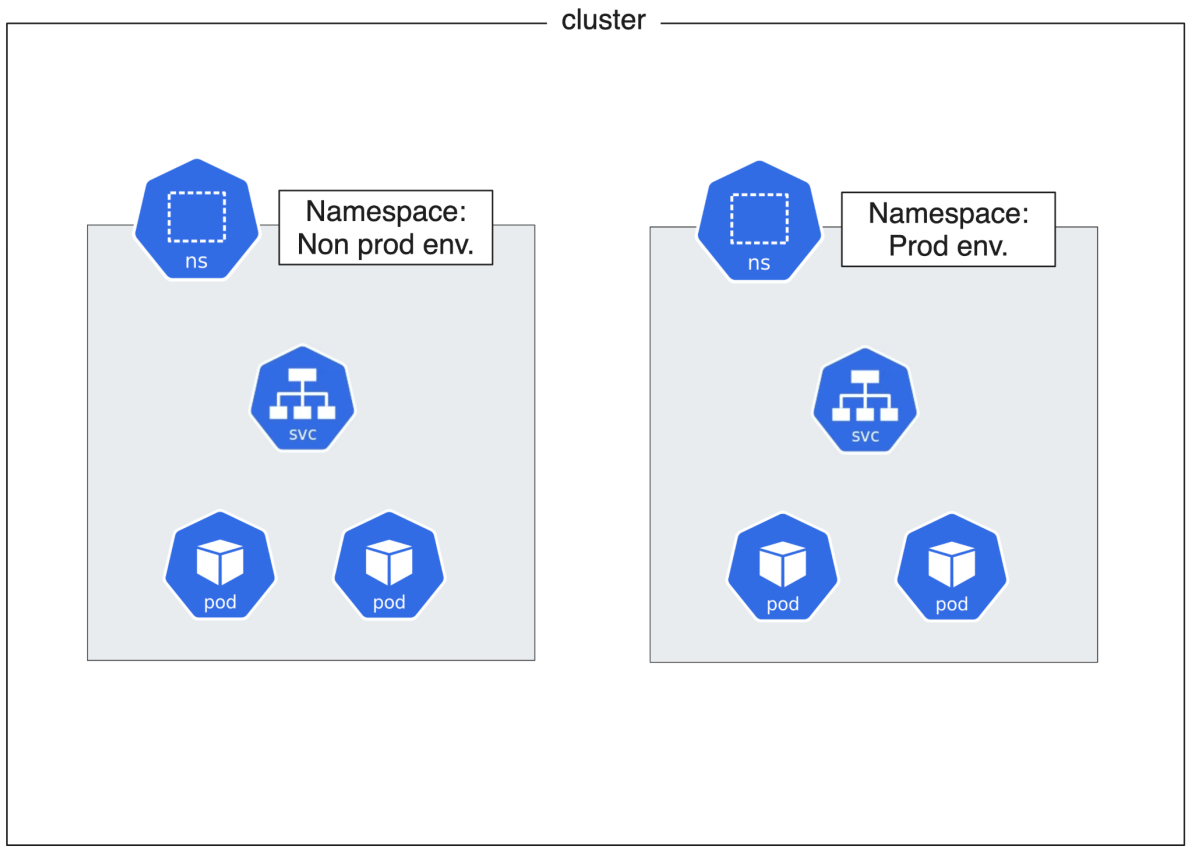
Namespace use cases

- Teams driven



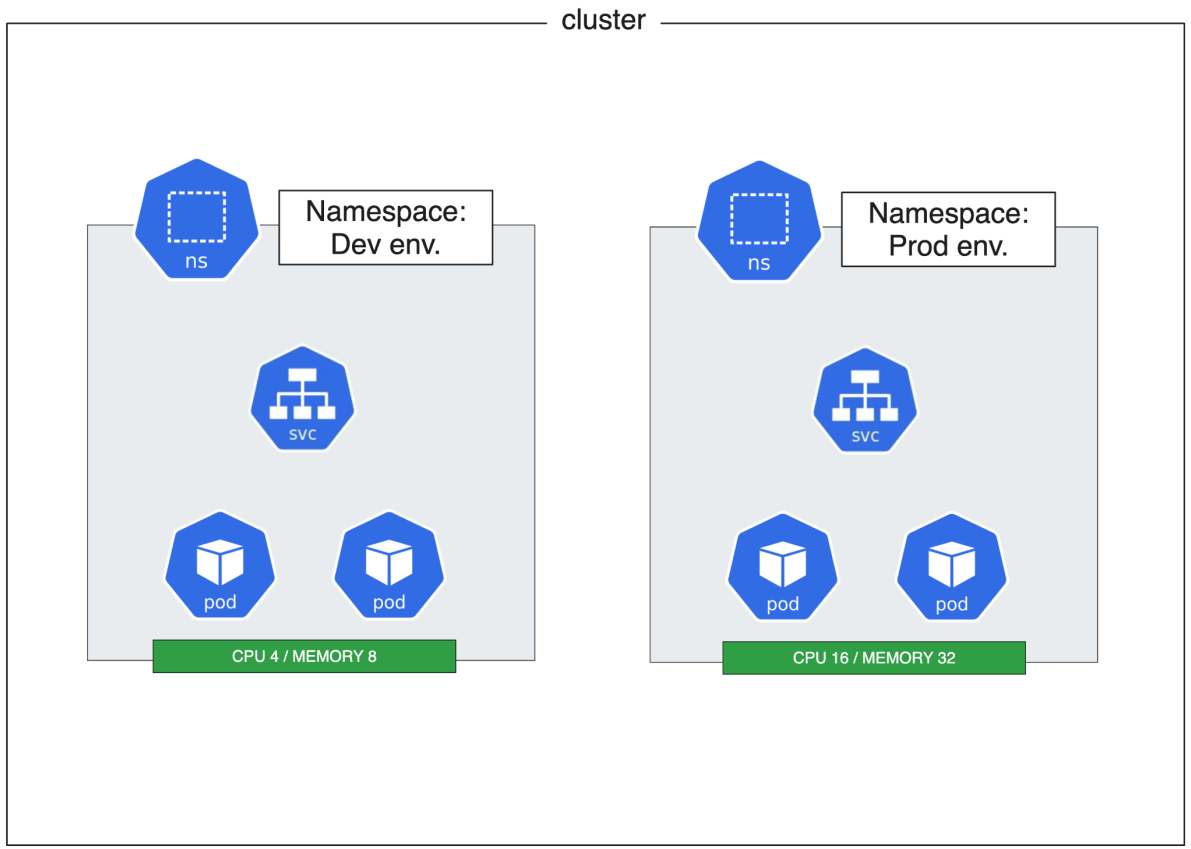
Namespace use cases

- Environments



Namespace use cases

- Resources quota limit



HELM

HELM

HELM is the package manager for Kubernetes

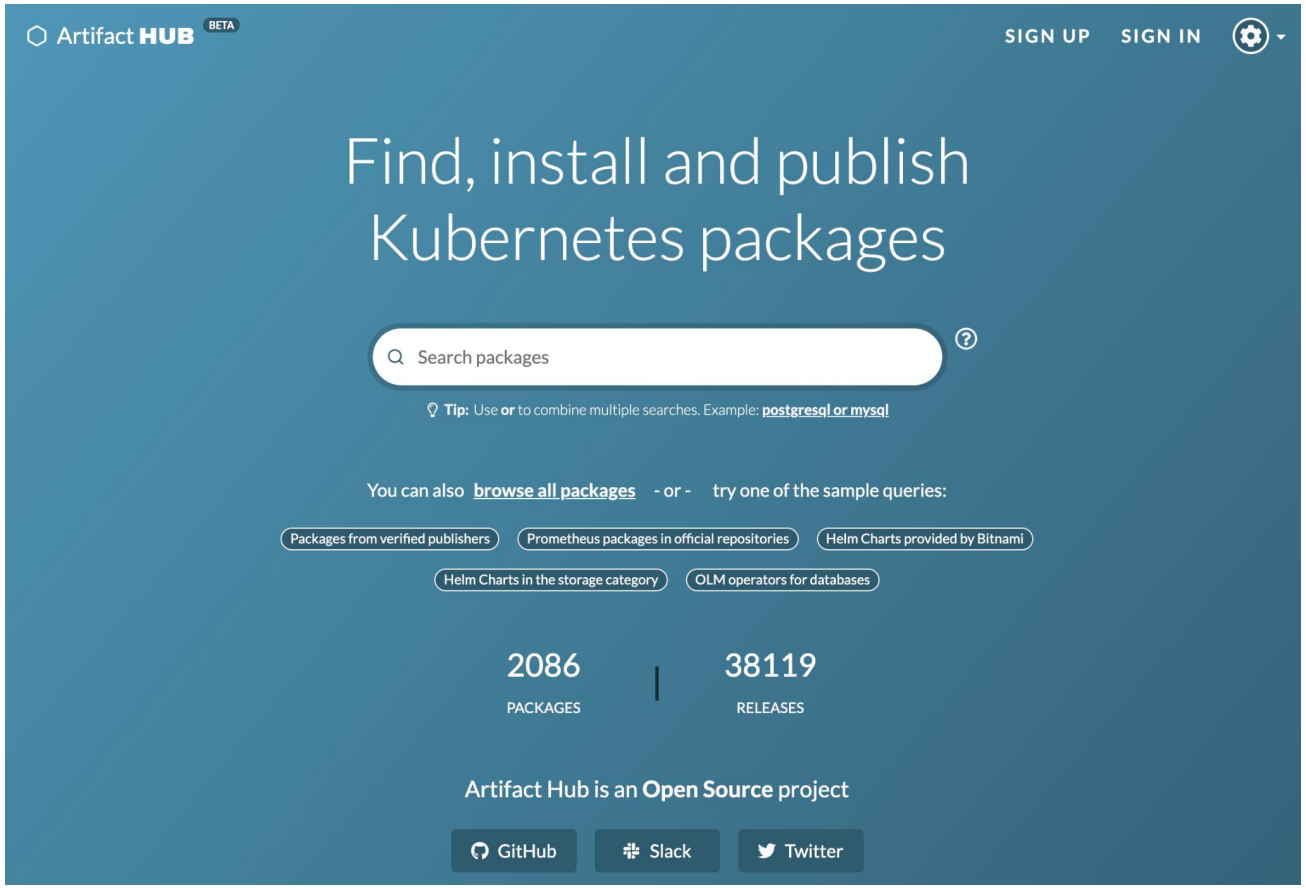
HELM

HELM is the package manager for Kubernetes

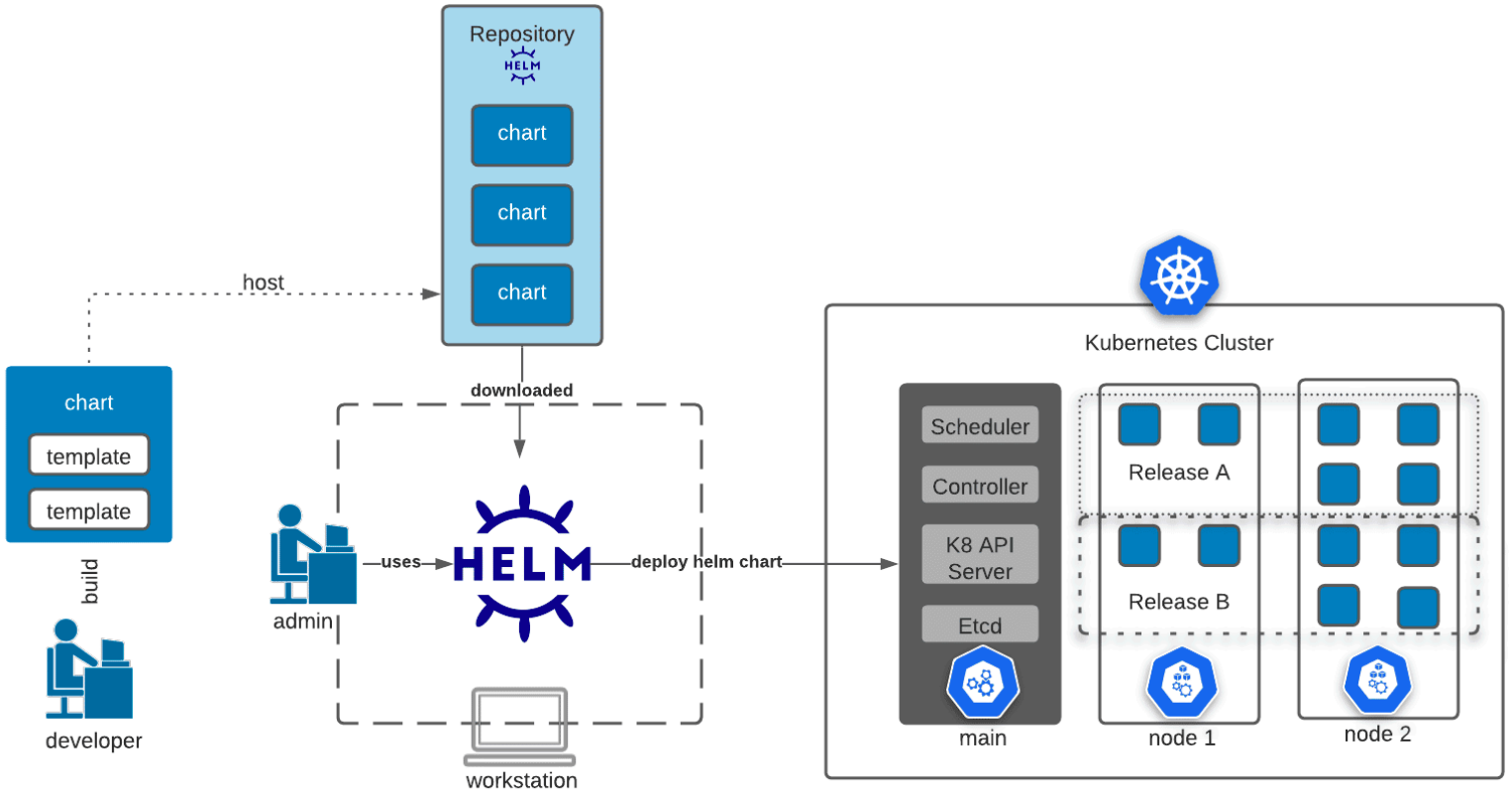
- Boosts productivity
- Reduces complexity
- Simple sharing
- Versioning and Upgrades

HELM Charts

Public HELM Chart: https://artifacthub.io/



HELM Charts



Helm Workflow

Workshop HELM