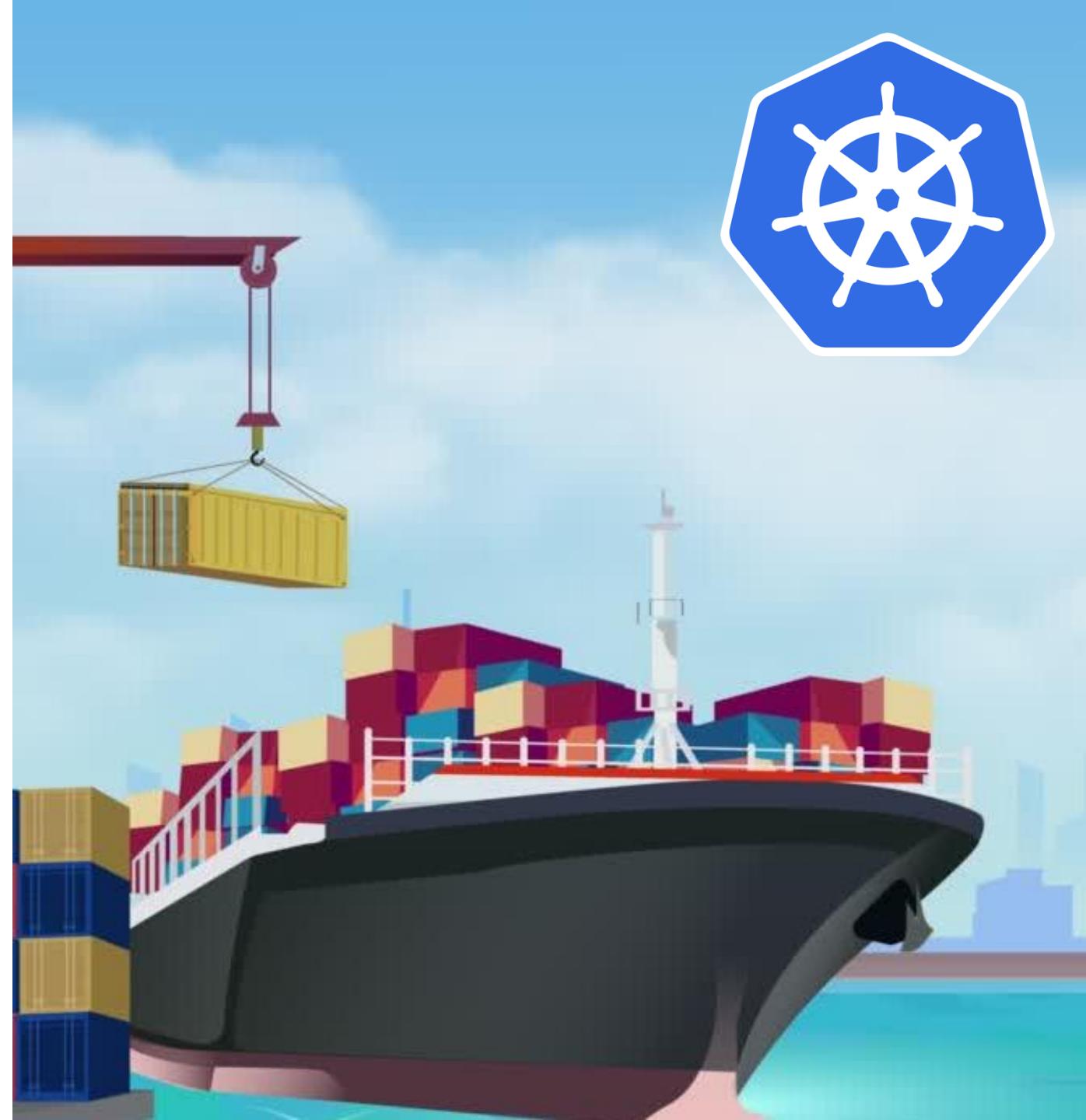




You Trust, We Deliver!

Practical Kubernetes - Beyond CKA and CKAD Hands-on





Yogesh Raheja



Puppet for the Absolute
Beginners – Hands-On



SaltStack for the Absolute
Beginners – Hands-On



Infrastructure Automation
with OpenTofu – Hands-On



AI Ecosystem for the Absolute
Beginners - Hands-On



Mastering Prompt
Engineering for GenAI



Mastering Docker Essentials -
Hands-on



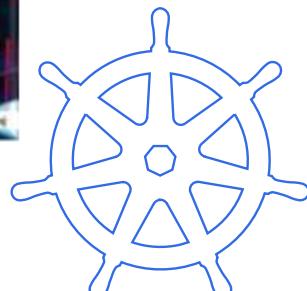
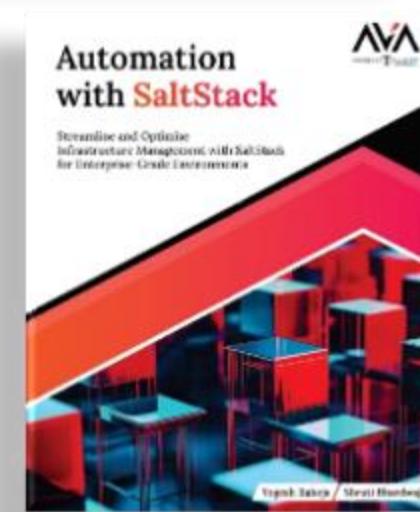
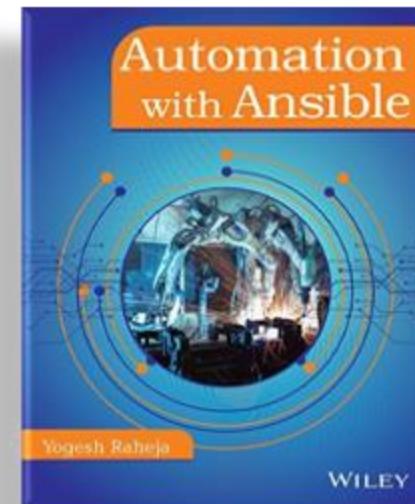
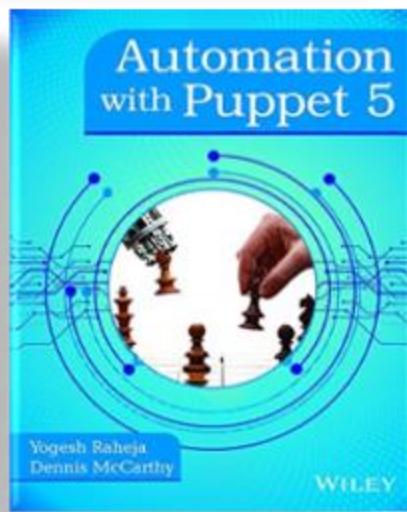
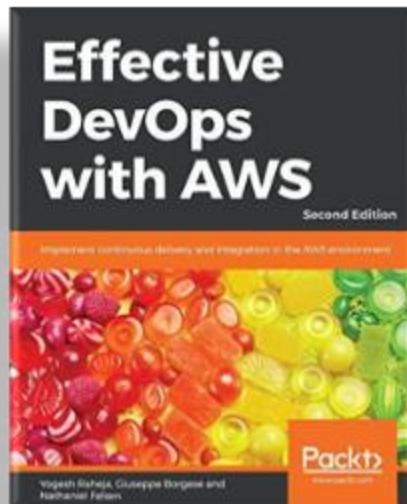
Unlocking Python for the
Absolute Beginners



Podman for the Absolute
Beginners - Hands-On



Yogesh Raheja

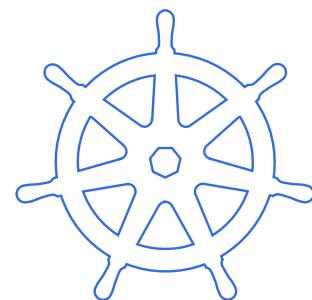




Yogesh Raheja



Shruti Bhardwaj



Course Workflow

Foundational Concepts

Container Orchestration

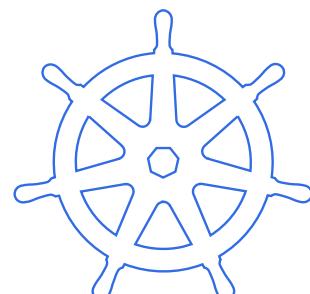
Kubernetes Architecture

Setup Methods

Fundamental Objects

Networking

Manifest File Creation



Course Workflow



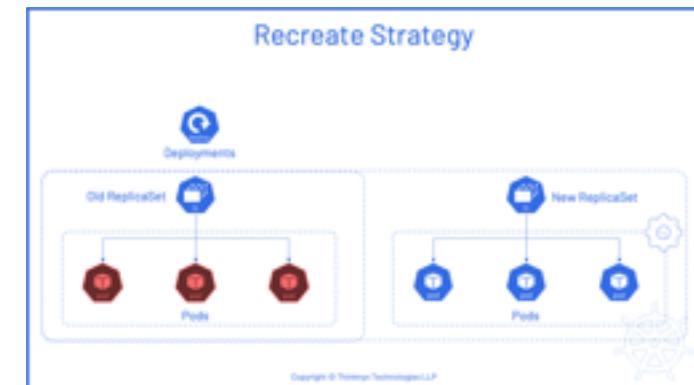
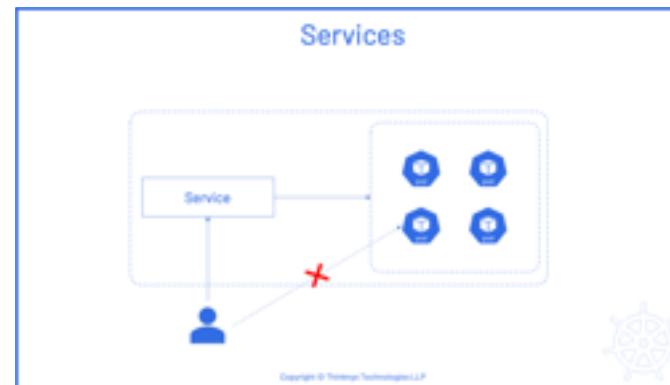
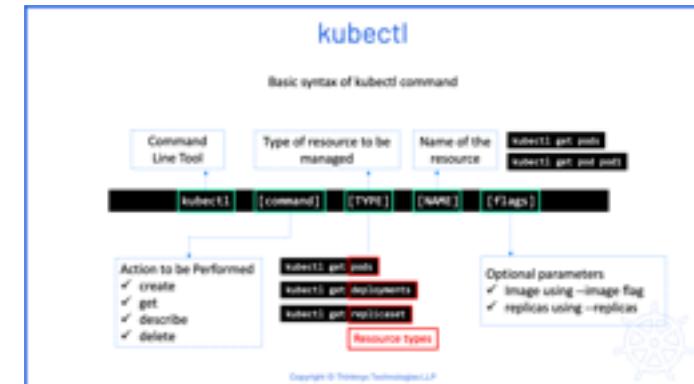
Lectures



Live Demonstrations



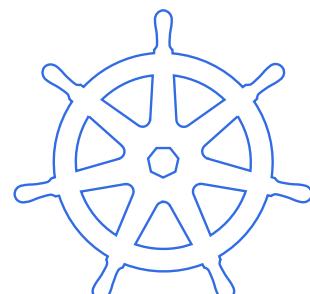
Assignments



Course Workflow

Intermediate & Advanced Topics

| | |
|----------------------------------|------------------------------|
| Application Upgrades & Rollbacks | Monitoring & Dashboarding |
| Cluster Maintenance | Ingress Controllers |
| Advanced Pod Configurations | Helm Package Manager |
| RBAC Security | Private Registry Integration |
| Network Policies | Advanced Storage Management |



Course Objective



- Container Evolution & Limitations
- Necessity & features of container orchestration
- Kubernetes Cluster Setup & Architecture
- Kubernetes Objects
- Networking
- Manifest Files
- Troubleshooting
- Scaling Clusters
- Managing Nodes
- Handling Rollouts & Rollbacks



Course Objective

Pod Scheduling

Storage Management with Persistent Volumes

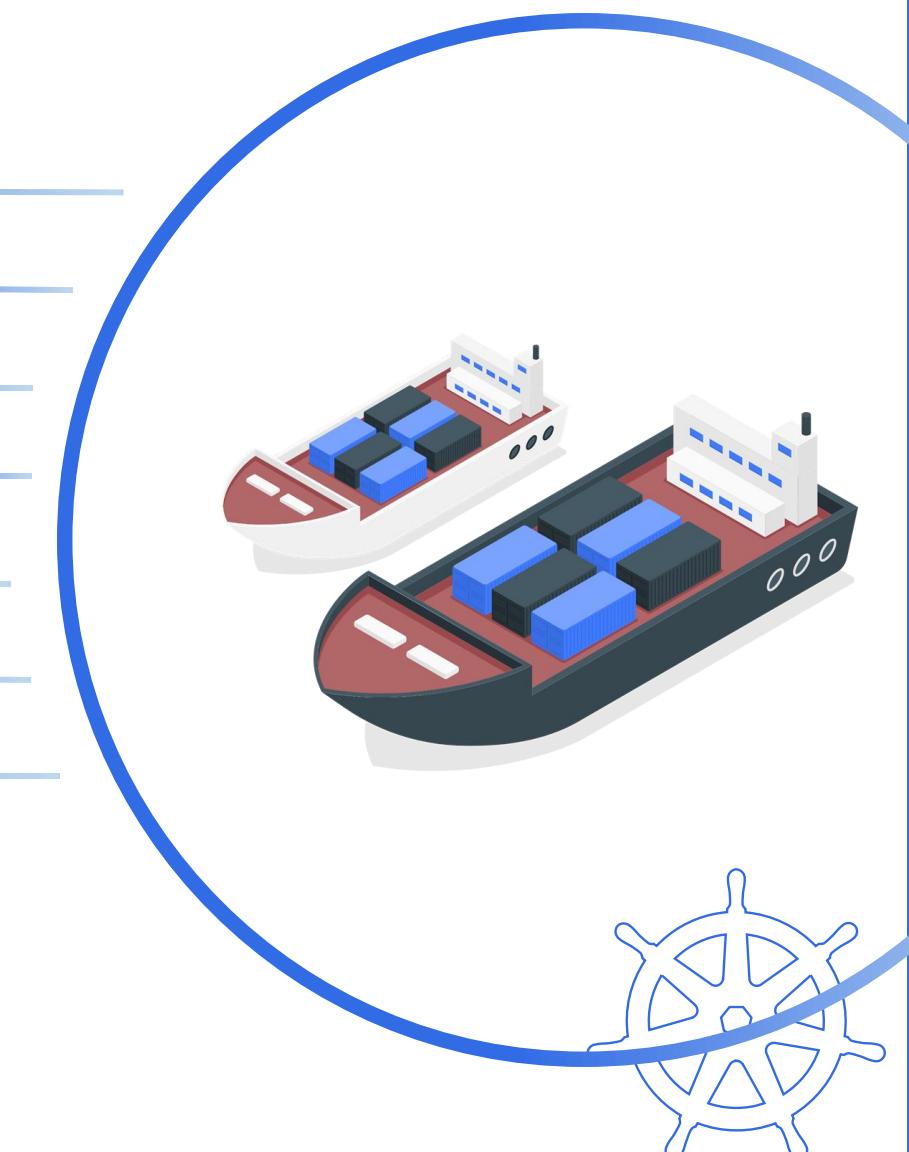
Security with RBAC & Network Policies

Managing external access through Ingress and TLS/SSL

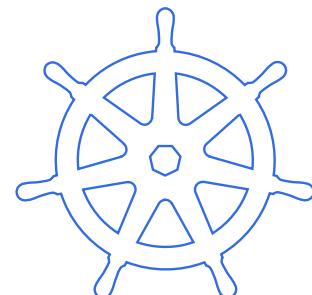
Monitoring & Dashboarding

Managing private registries

Helm Charts



Course Objective

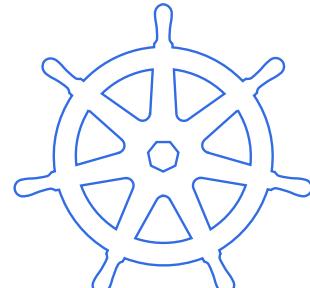


Section: 1

Section Overview

Introduction to Container Orchestration

- ↳ Evolution of Computing
- ↳ Limitations of Containers
- ↳ Introduction to Container Orchestration
- ↳ Popular Container Orchestrators
- ↳ Standard Features of Orchestrators



Evolution of Containers

From Bare Metal to Containers

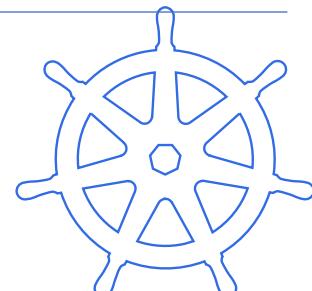
Data Centre Infrastructure

Special buildings that hold
the physical servers



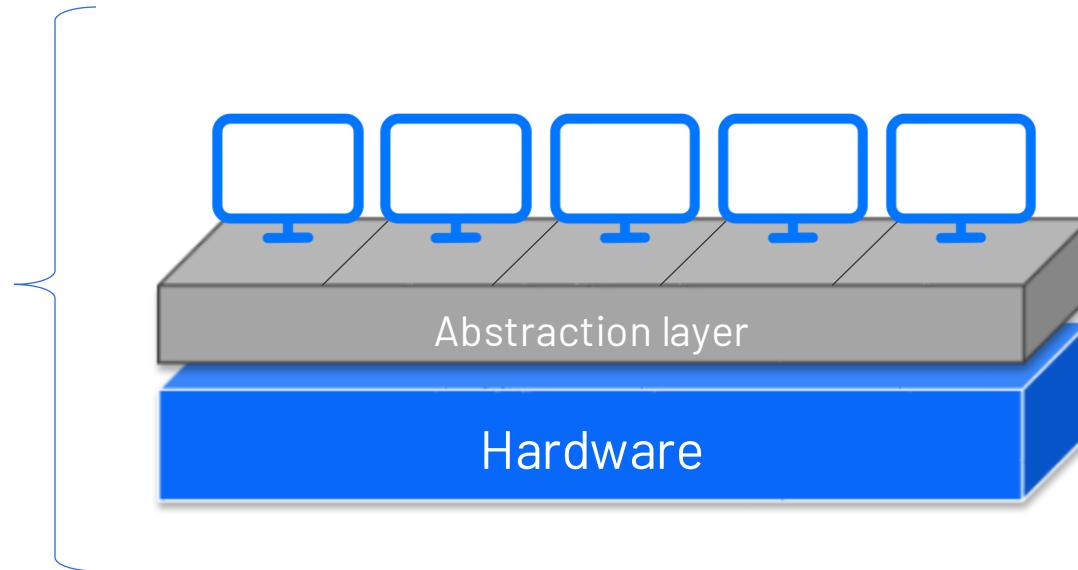
Data Centre

Servers were underutilized
and resulting in wasted
resources



Virtualization

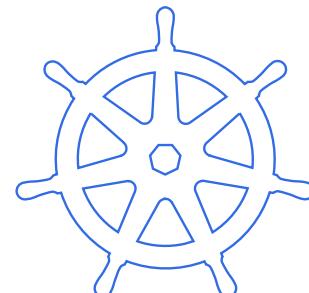
Improves resource utilization by running multiple workloads



- Processors
- Memory
- Storage

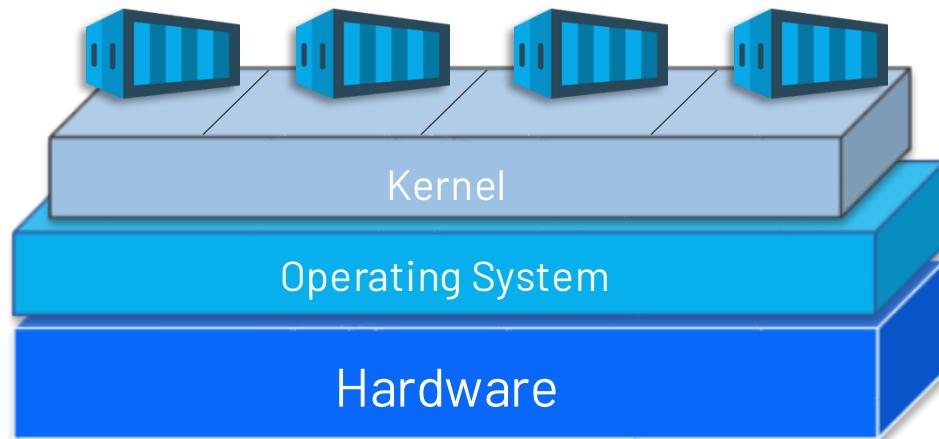


- Time consuming
- Resource intensive

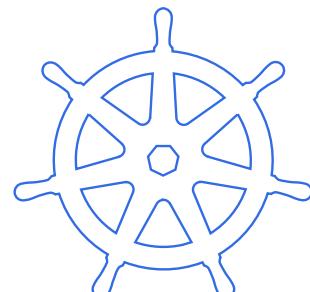


Containers

- More lightweight & modular
- Allow packaging & isolation of applications with their runtime environment

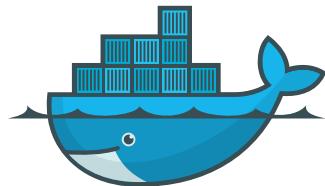


- ✓ Reduce deployment time & resource requirements
- ✓ Run containers in any cloud infrastructure



Container Runtime Tools

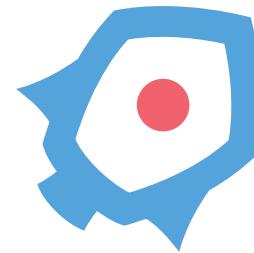
Container Runtime Tools



Docker



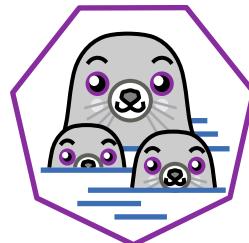
containerd



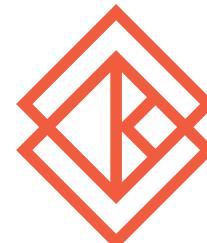
rkt



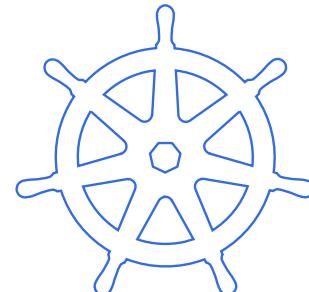
cri-o



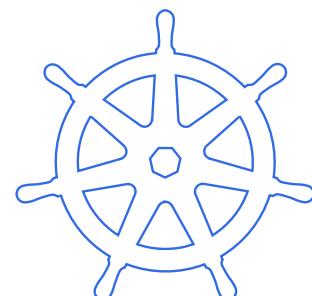
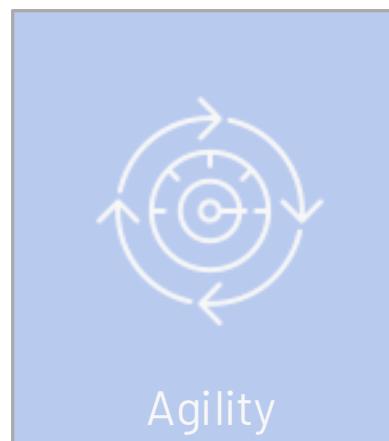
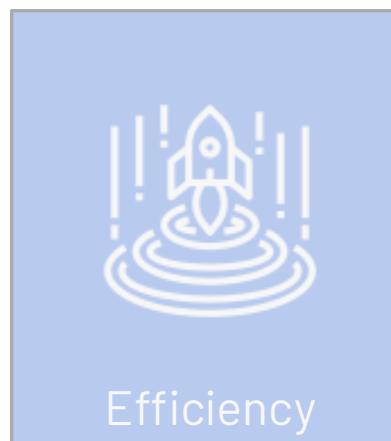
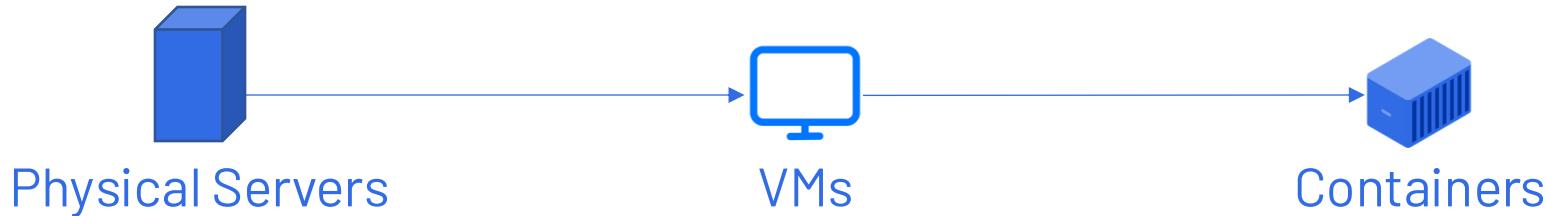
podman



Kata Container



Evolution of Containers



VMs vs Containers

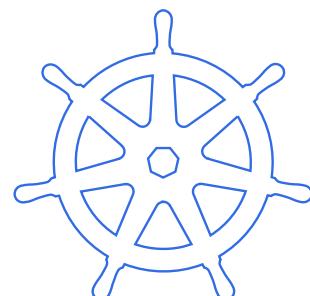
| Feature | Virtual Machines(VMs) | Containers |
|----------------------|---|---|
| Virtualization | Hardware Virtualization | Operating System Virtualization |
| Definition | Full operating system environment running on a hypervisor | Lightweight, portable, and efficient way to package and run applications |
| Relevance | Suitable for workloads requiring a full OS | Ideal for modern, microservices-based applications |
| Startup Time | Slower | Faster |
| Resource Utilization | More resource-intensive | Efficient resource utilization |
| Portability | Less portable due to OS dependencies and larger size | Highly portable across different environments |
| Example Use Cases | Legacy applications or applications with specific database versions | Social media platforms containerizing services like user feed, notification system, and search engine |



Containers



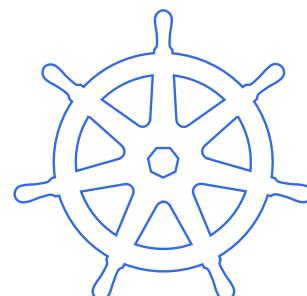
Containers provide lightweight, portable, and efficient way to package and run applications



Limitations of Containers

Limitations of Containers

-  Lack of Self Healing
 - No built-in mechanism to automatically re-create containers
-  High Availability Challenges
 - When server fails, entire application becomes unavailable, causing significant downtime
-  Scaling hurdles
 - Lacks automatic scaling
-  Load Balancing Challenges
 - Doesn't directly handle load balancing,
-  Storage Constraints
 - Rely on the underlying host system's storage resources



The Road Ahead:

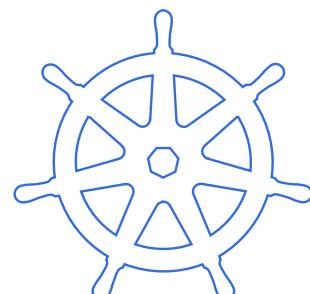
Mitigating Limitations with Container Orchestrator



Containers

Container Orchestration

- ✓ High availability
- ✓ Automated scaling
- ✓ Efficient networking
- ✓ Seamless integration



Container Orchestration

Container Orchestration

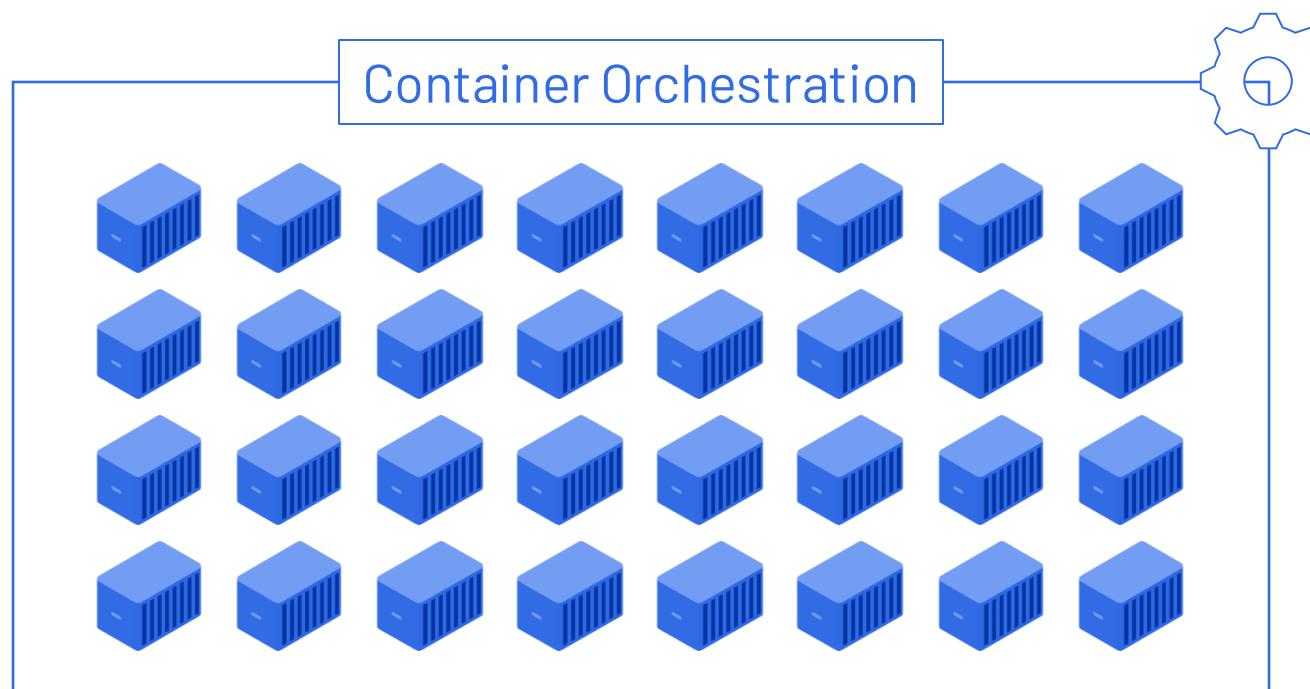


Agility
Efficiency



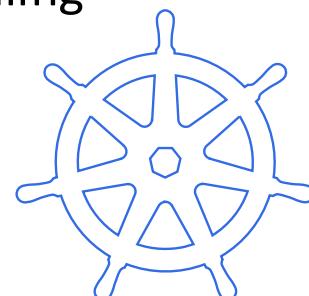
Manages lifecycle
of containerized
applications

Address many of
limitations inherent
to containers



Automate Tasks like:

- ✓ Deployment
- ✓ Scaling
- ✓ Health Checks
- ✓ Self-healing

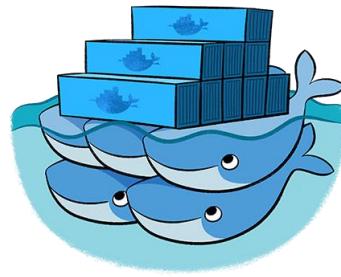


Container Orchestration



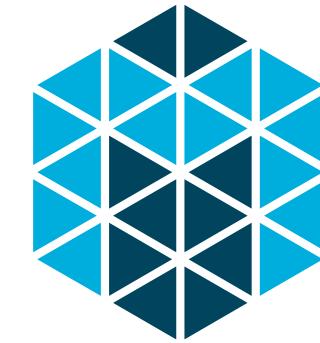
Kubernetes

- ✓ The Orchestral Maestro
- ✓ High availability, automated scaling, self-healing capabilities, and rich service discovery mechanisms
- ✓ Steeper learning curve



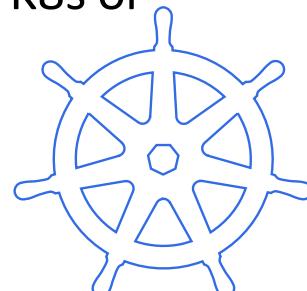
Docker Swarm

- ✓ Built-in Docker feature for managing multiple Docker engines as a cluster
- ✓ Easy to set up and use
- ✓ Less extensive compared to K8s



Apache Mesos

- ✓ Acts as a cluster kernel
- ✓ Flexible and resource optimization
- ✓ Requires more configuration effort compared to K8s or Swarm



Choosing Right Orchestrator

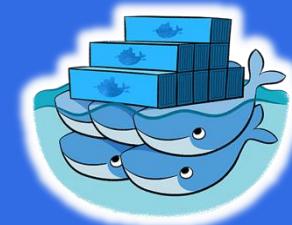
Choosing Right Orchestrator

Depends on specific requirement and existing environment



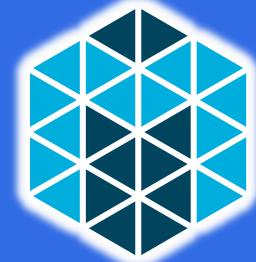
Kubernetes

For complex designed cloud native applications or demanding third party integrations



Docker Swarm

If simplicity and ease of setup are priorities



Apache Mesos

For Heterogeneous Workloads



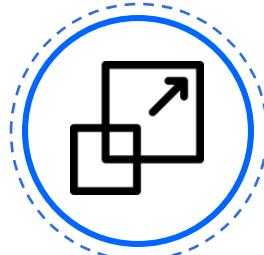
Features of Container Orchestrator

Features of Container Orchestrator

High Availability & Self-Healing



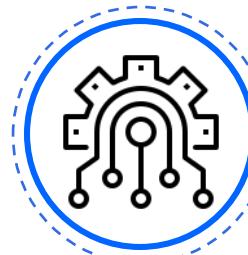
Scaling



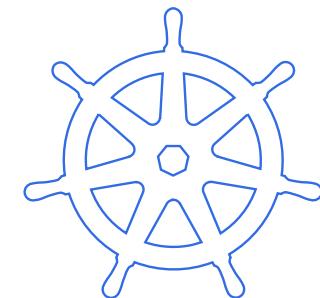
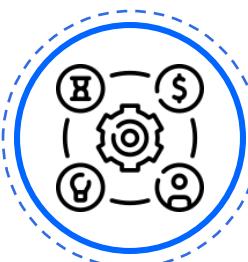
Zero Downtime Application Upgrades



Integration with External Tools



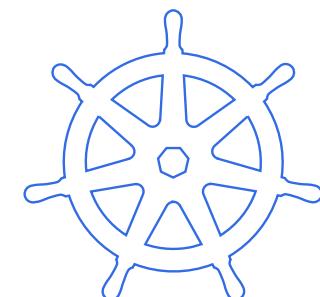
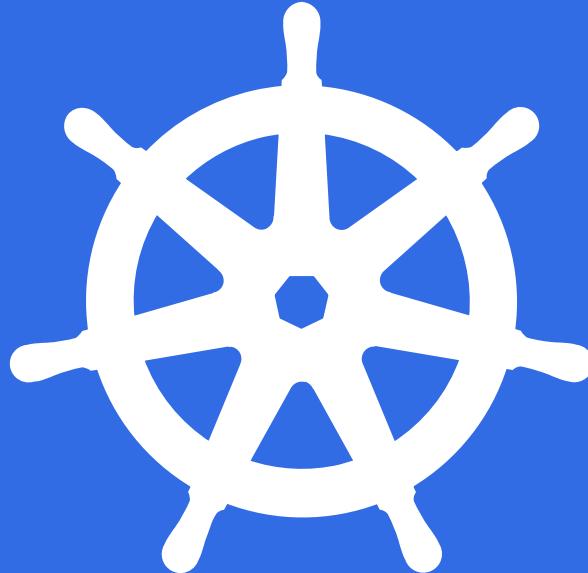
Resource Management & Scheduling



Summary

Summary

- ✓ Evolution of containers
- ✓ Container limitations
- ✓ How orchestrators address container limitations
- ✓ Key features of orchestrators

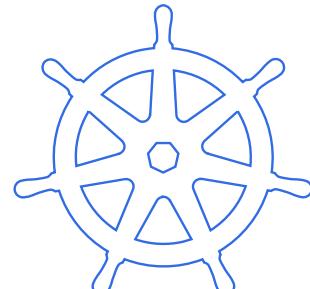


Section: 2

Kubernetes at a Glance

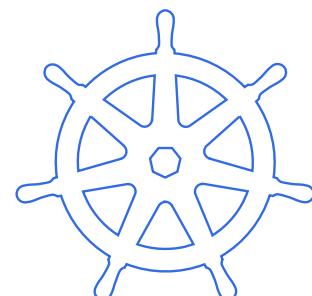
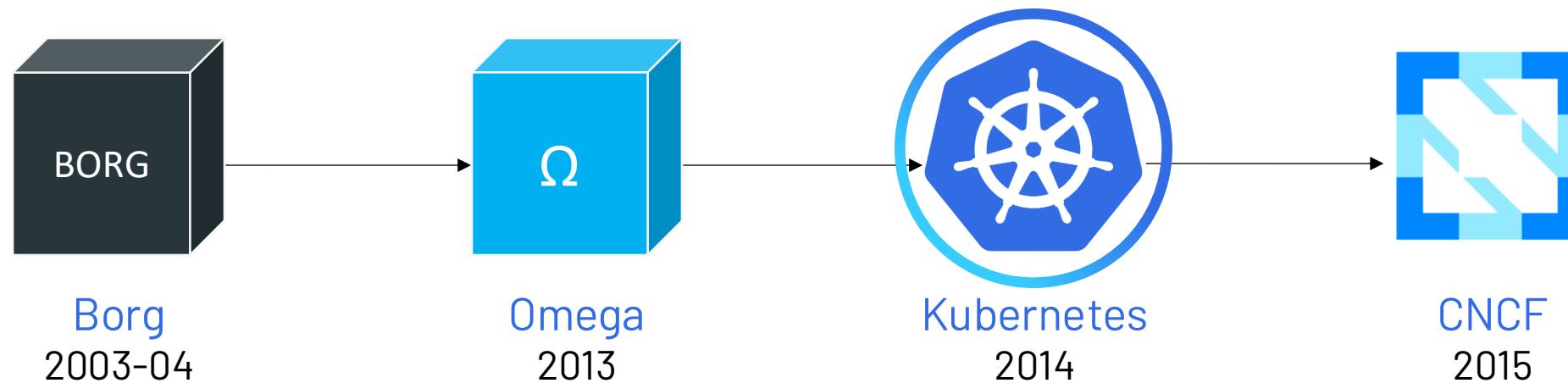
Section Overview

- **Introduction to Kubernetes**
- **Why Kubernetes**
- **Setup Options**



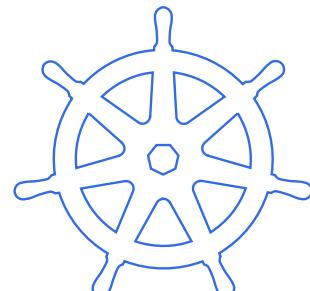
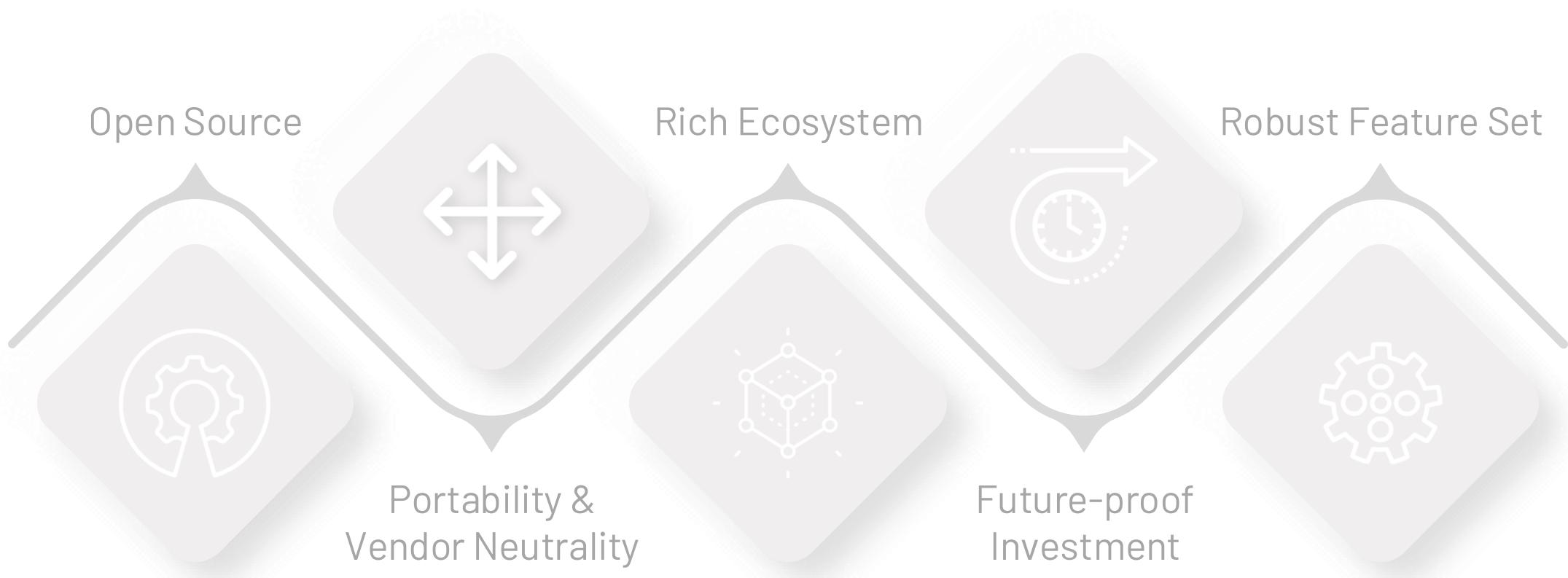
Introduction to Kubernetes

Introduction to Kubernetes

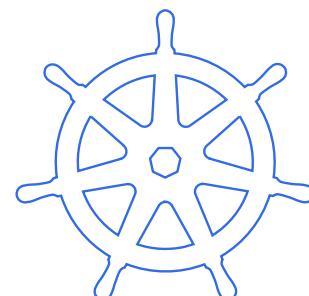


Why Kubernetes?

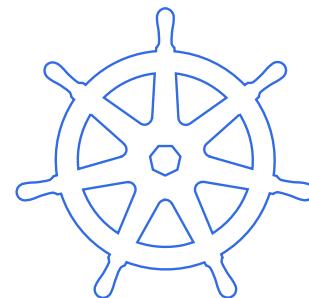
Why Kubernetes?



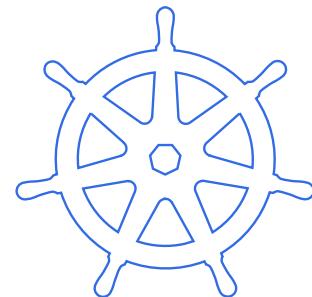
Why Kubernetes?



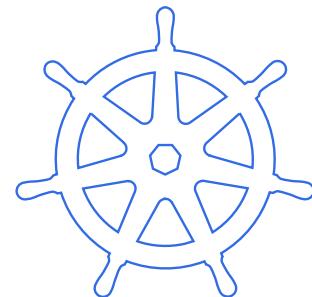
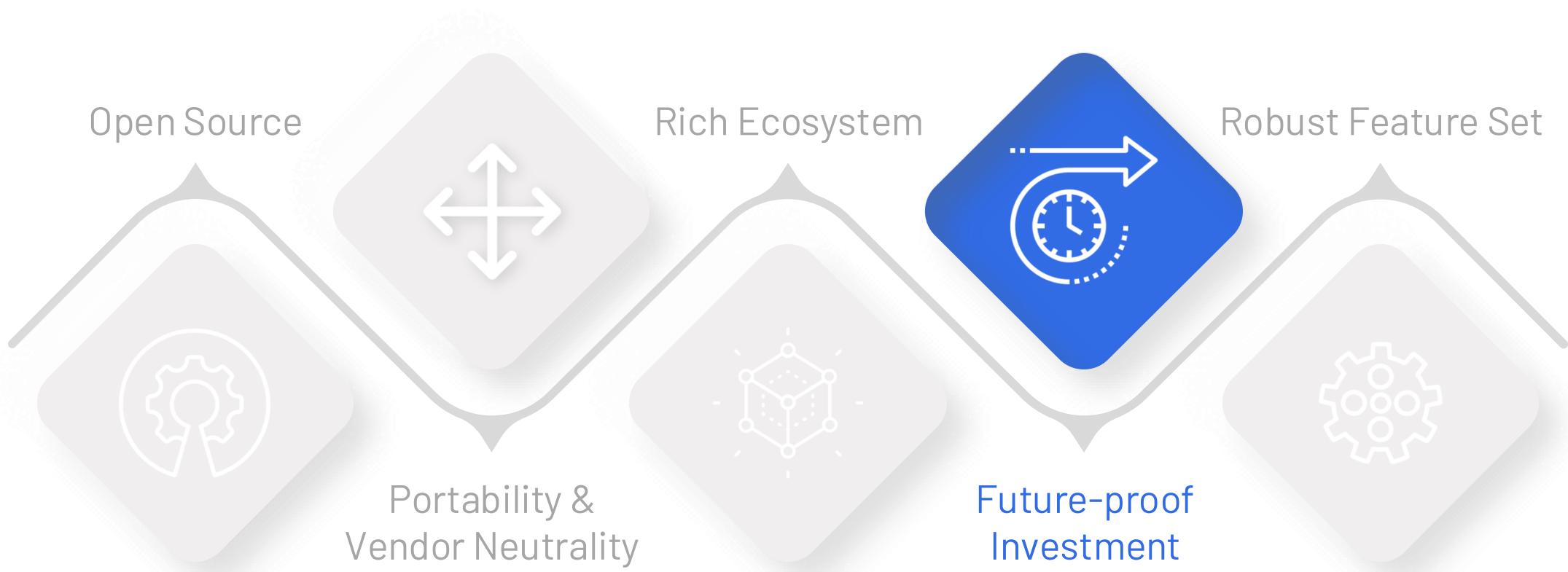
Why Kubernetes?



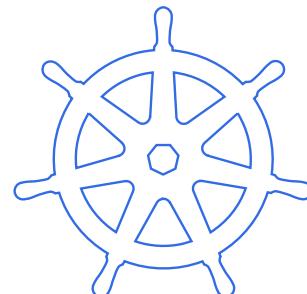
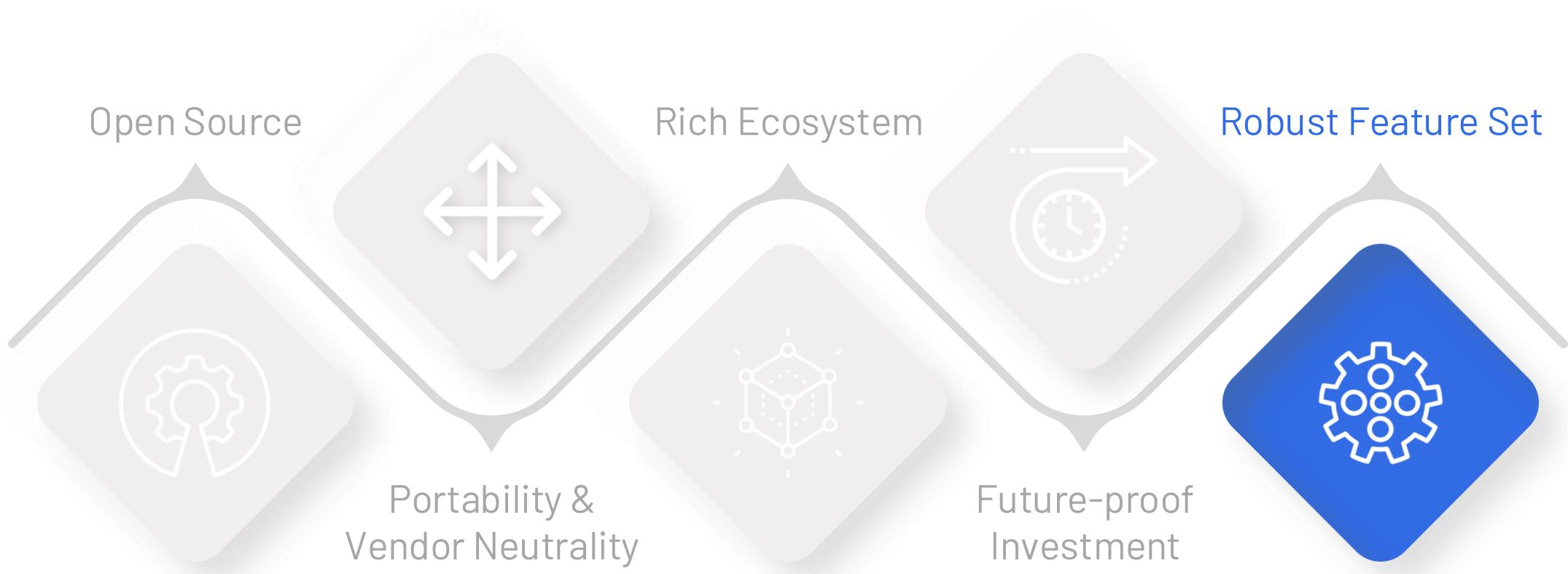
Why Kubernetes?



Why Kubernetes?



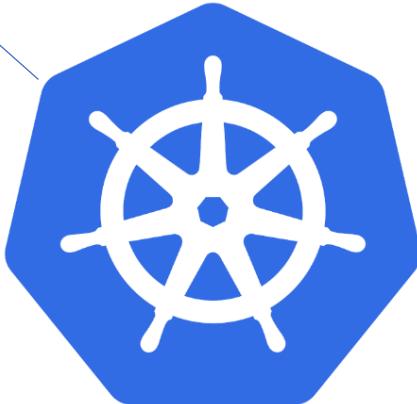
Why Kubernetes?



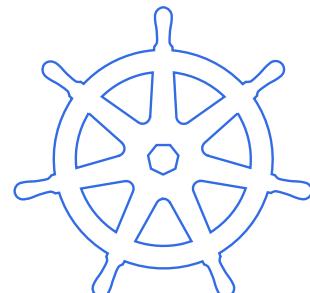
Why Kubernetes?

Modular Architecture

- ✓ Container runtimes
- ✓ Networking solutions
- ✓ Storage systems



Offers compelling combination of features, flexibility, and community support



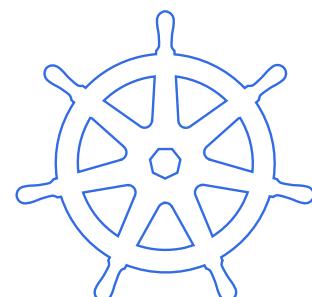
Setup Options

Setup Options

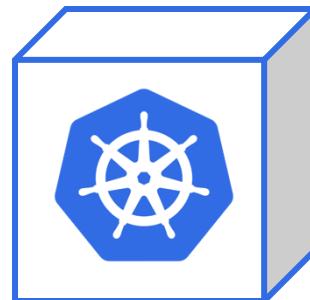


Native Kubernetes

- ✓ Full Control & Flexibility at no cost
- ✓ Requires Hands-On exposure
- ✓ Ideal for experienced users
- ✓ No official support (Only Community driven support)

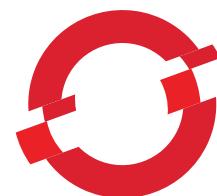


Setup Options



Enterprise Kubernetes

- ✓ Built upon Native Kubernetes
- ✓ Comes with official support
- ✓ Higher costs
- ✓ Higher resource requirements (for control plane and worker nodes)
- ✓ Ideal for Enterprises requiring immediate support and willing to make substantial investments



OpenShift



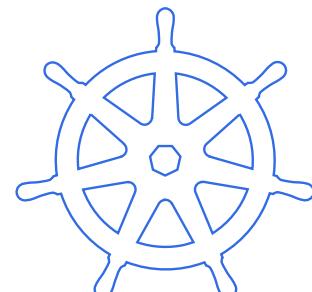
Rancher



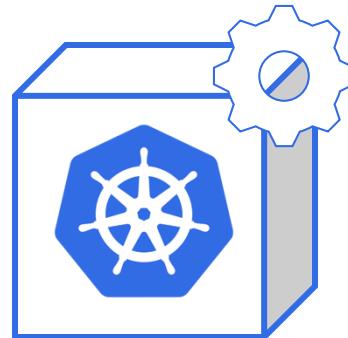
Tanzu



Rakuten CNP



Setup Options



Managed Kubernetes

- ✓ Popularly known as Container as a service(CaaS)
- ✓ Focus only on Applications development & deployment
- ✓ Integrating toolsets may require additional efforts
- ✓ Cost-effective for smaller deployments
- ✓ Ideal for organizations looking for a straightforward entry into Kubernetes



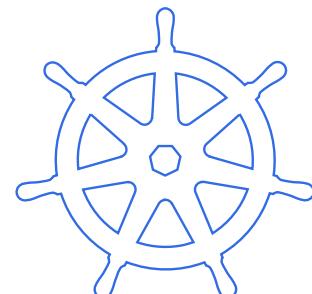
EKS



AKS



GKE

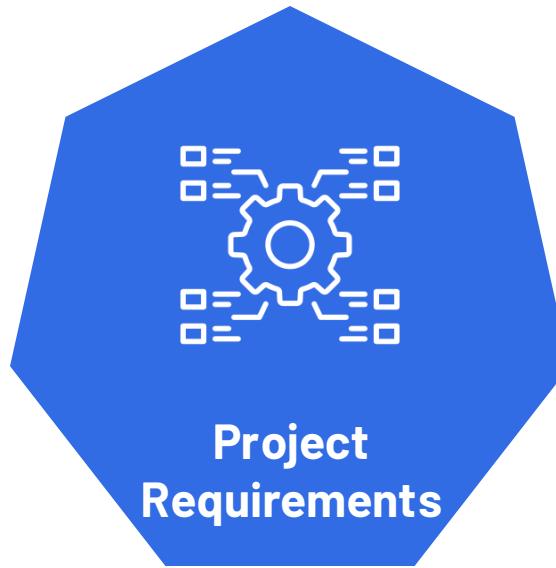


Choosing the Right Option

Choosing the Right Option



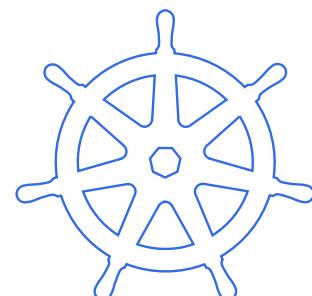
Expertise



Project Requirements



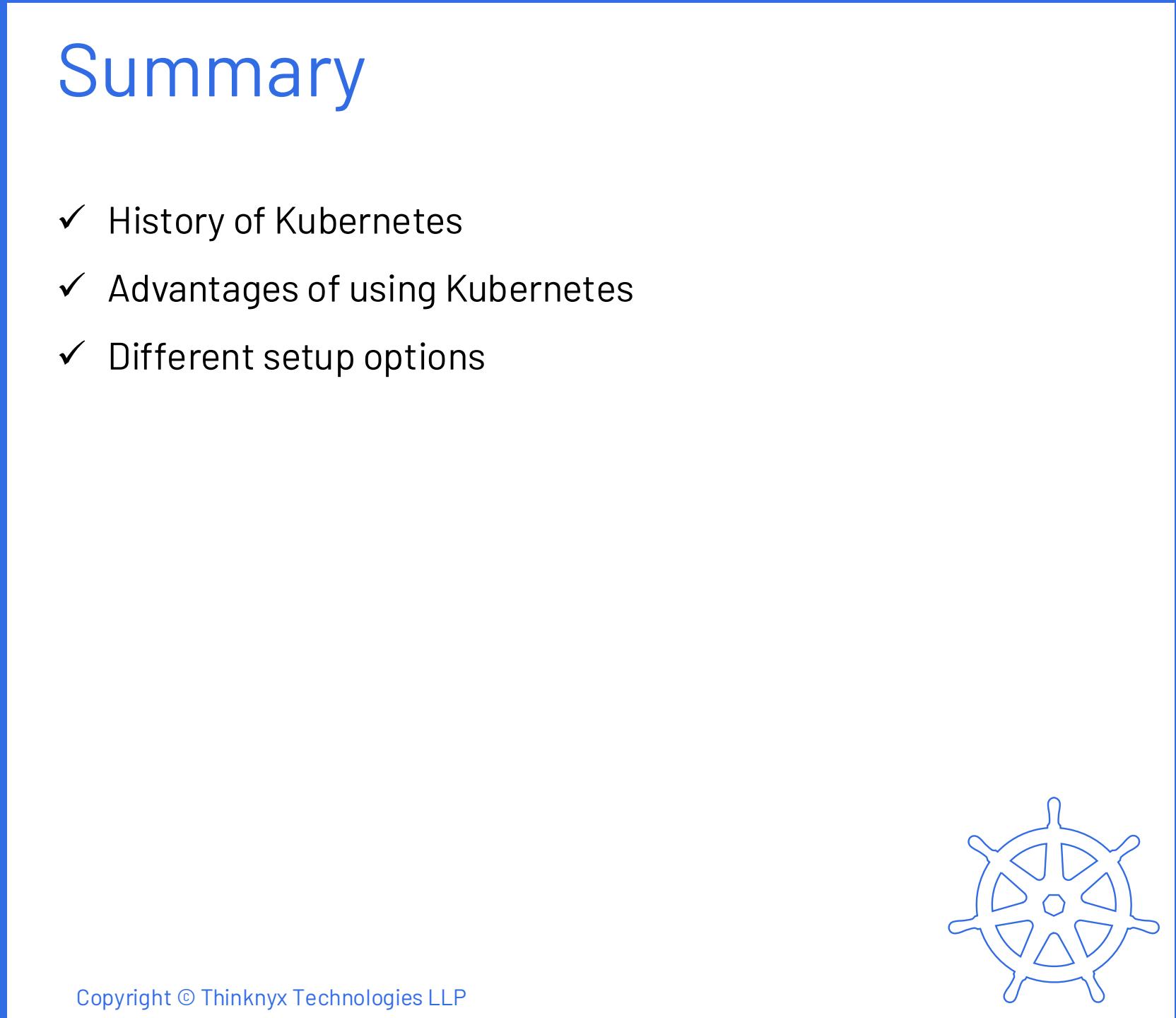
Budget



Summary



Summary

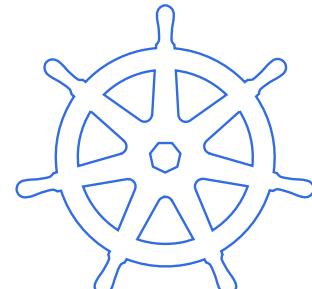
- ✓ History of Kubernetes
 - ✓ Advantages of using Kubernetes
 - ✓ Different setup options
- 

Section: 3

Understanding Kubernetes Architecture

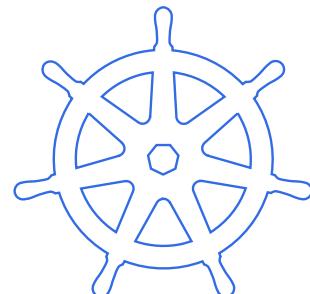
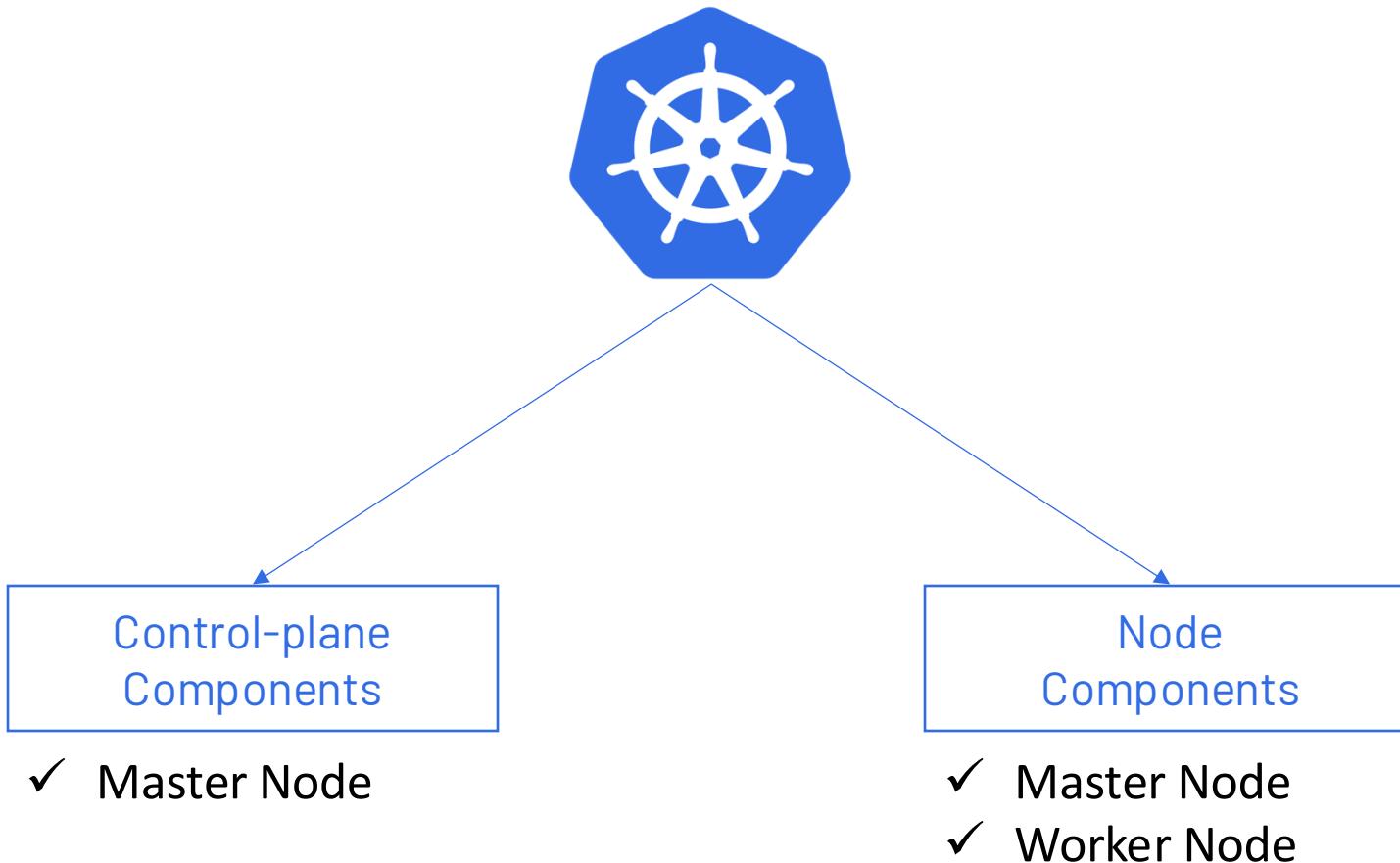
Section Overview

- ↳ Architectural components of Kubernetes
- ↳ Kubernetes Documentation
- ↳ Troubleshooting tips



Overview of Kubernetes Components

Overview of Kubernetes Components



Control-plane Components

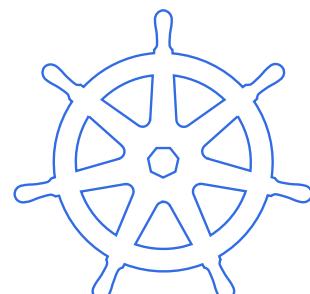
Control-plane Components



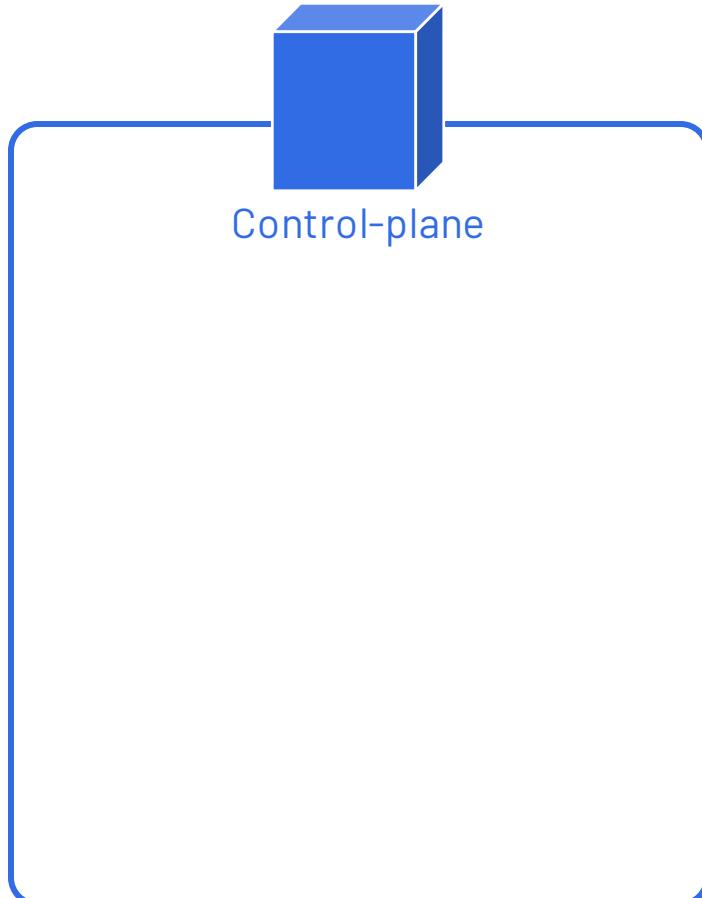
Control-plane

Central nervous system of the cluster

Manage, plan, schedule and monitor nodes and application containers

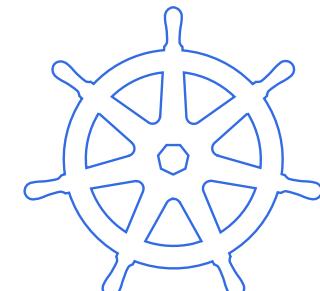


Control-plane Components

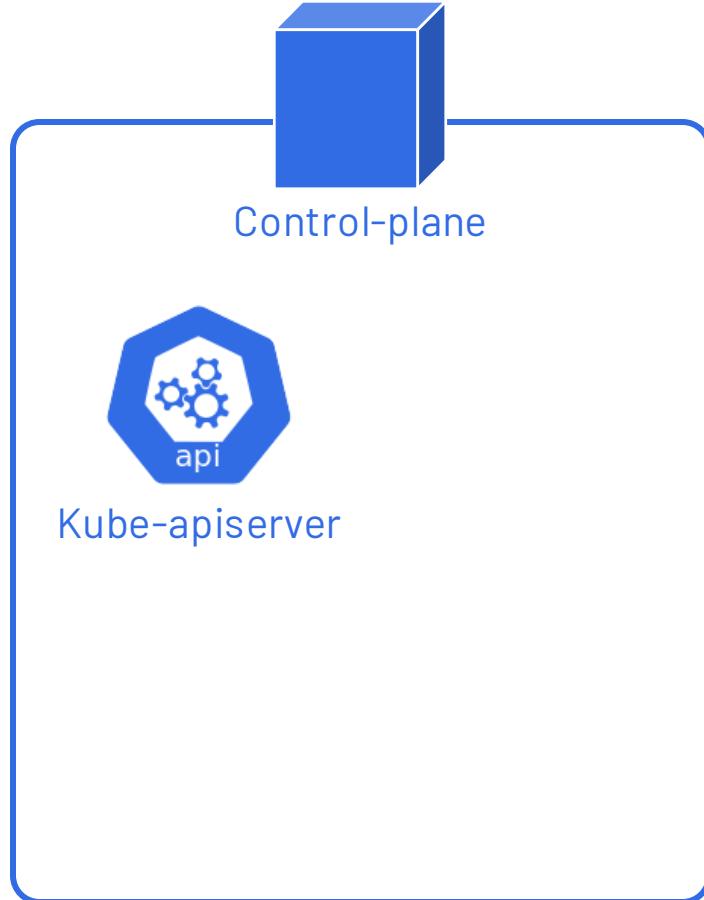


Kube-apiserver

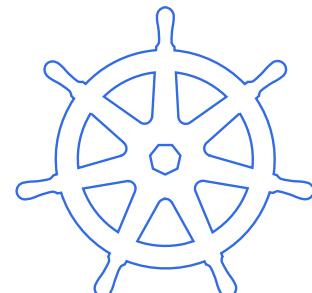
- ✓ Acts as a central point of communication & management
- ✓ Responsible for exposing K8s API's & orchestrating all operations within the cluster
- ✓ Serves as single point of entry for all user requests & interactions with the cluster



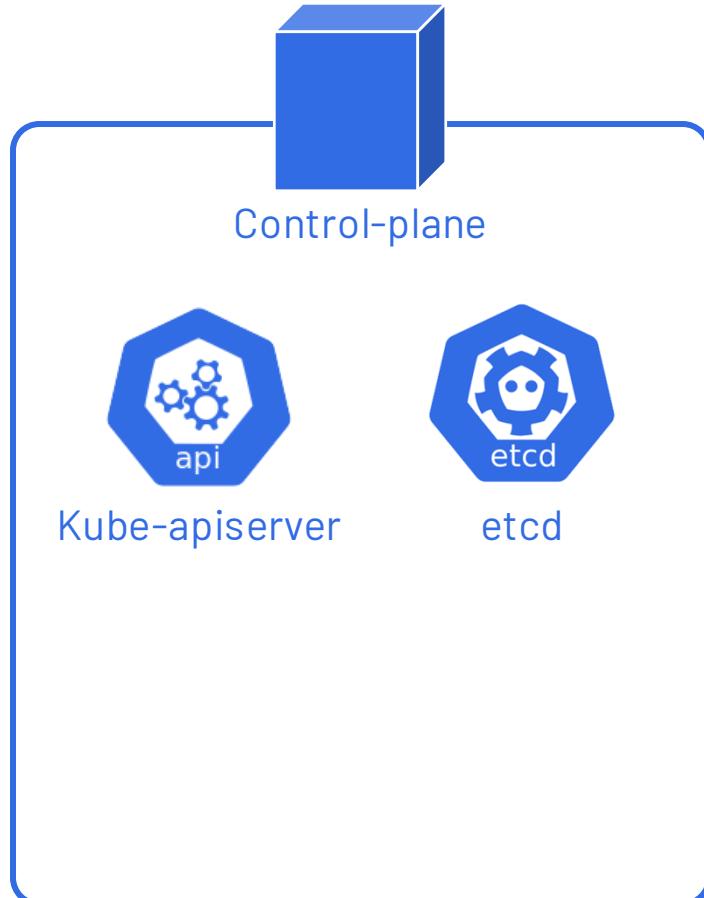
Control-plane Components



- ✓ Brain's memory for Kubernetes
- ✓ Securely stores all cluster configuration data & current state of all resources within the cluster
- ✓ Provides a single source of truth

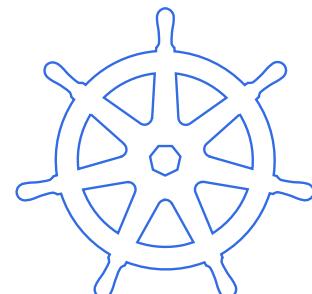


Control-plane Components

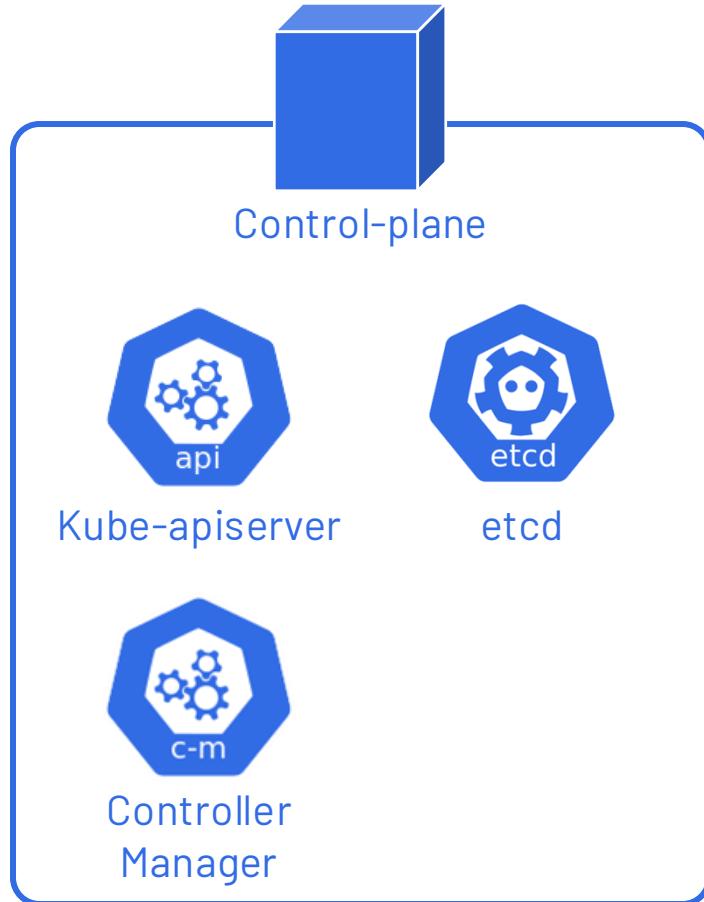


Controller Manager

- ✓ Diligent conductor of the Kubernetes
- ✓ Monitors current cluster state & compares it to desired state
- ✓ Orchestrates various specialized controllers, each responsible for a specific task

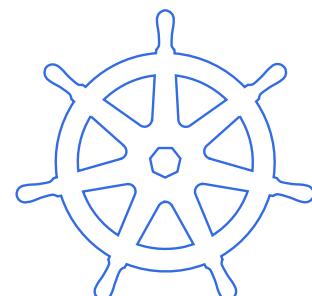


Control-plane Components

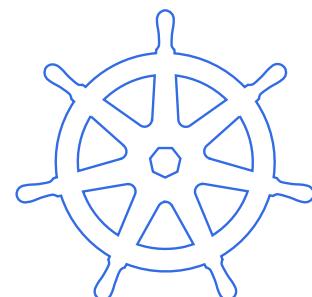
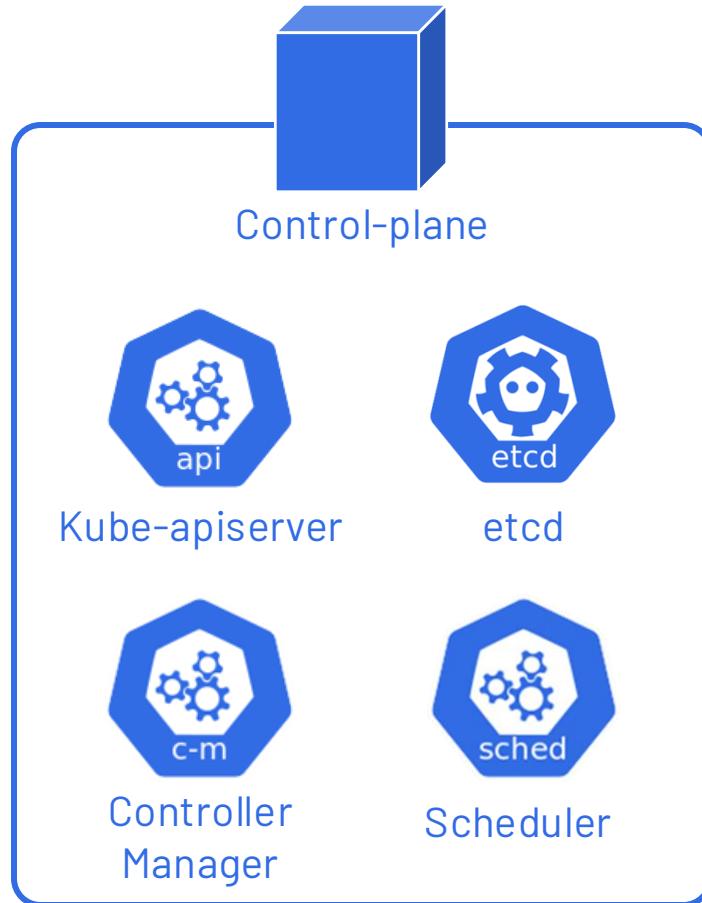


Scheduler

- ✓ Acts as the placement officer
- ✓ Analyzes factors of nodes
- ✓ On the basis of analysis, it makes informed decisions

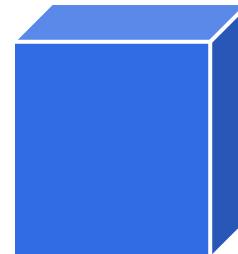


Control-plane Components



Node Components

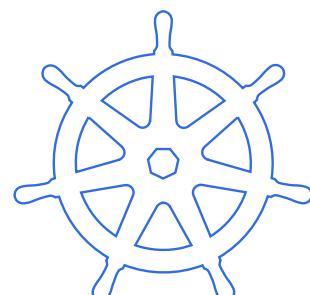
Node Components



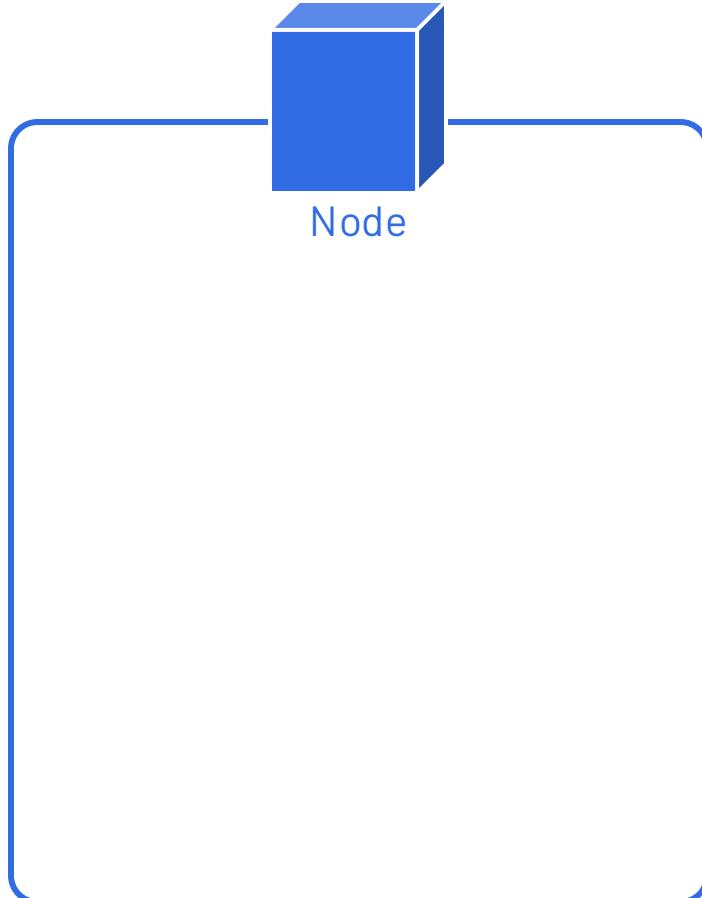
Node

Node components reside on every node in a cluster

Act as the workhorses that execute containerized applications

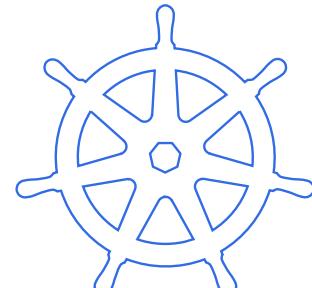


Node Components

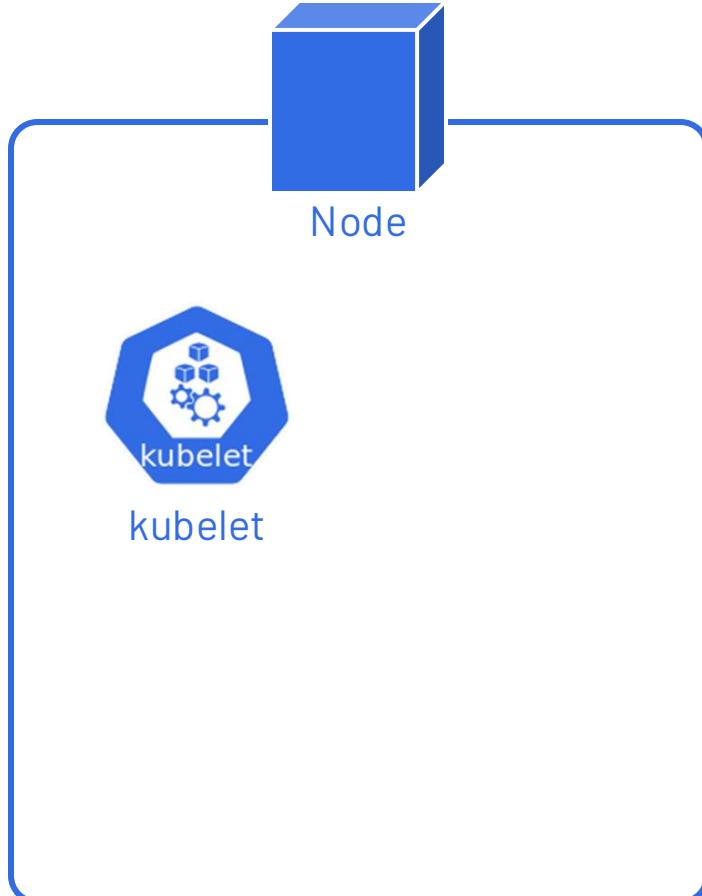


kubelet

- ✓ Functions as a captain on each node
- ✓ Receives instructions from kube-apiserver & manages lifecycle of pods
- ✓ Takes care of tasks like pulling container images from registries, allocating POD resources, reporting the health status

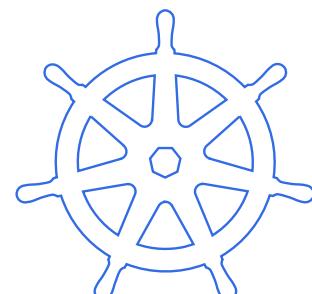


Node Components

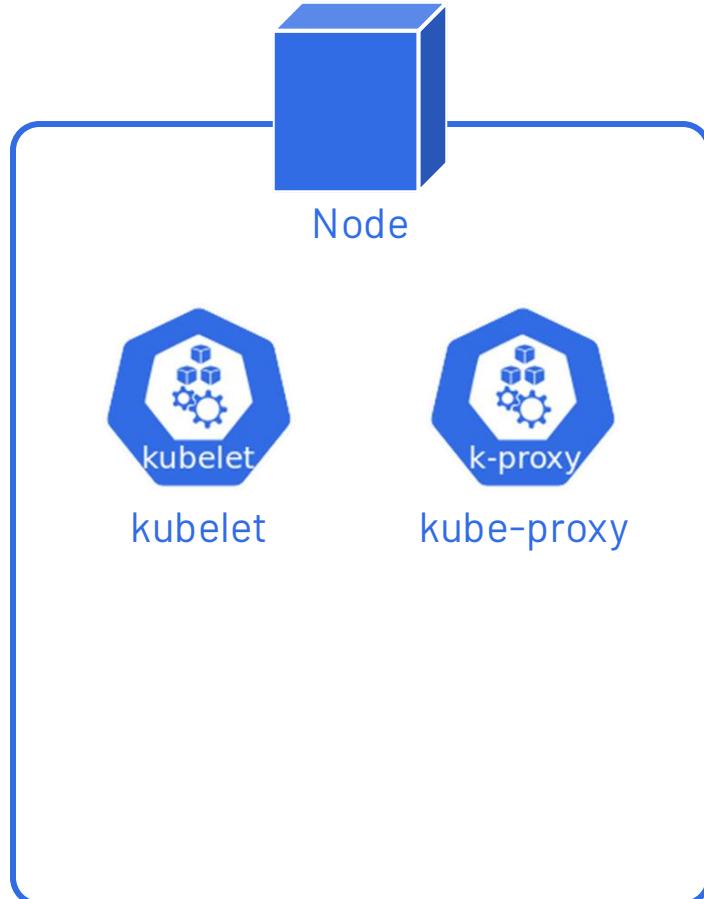


kube-proxy

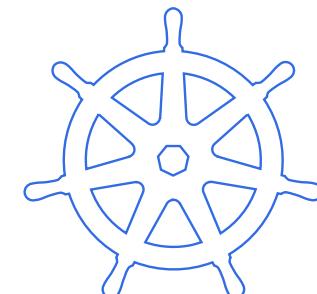
- ✓ Acts as the network traffic director
- ✓ Ensures pods can communicate with each other seamlessly
- ✓ Facilitates service discovery, allowing pods to locate & connect with services running within the cluster



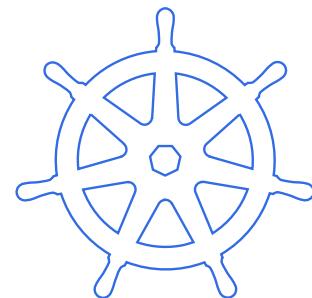
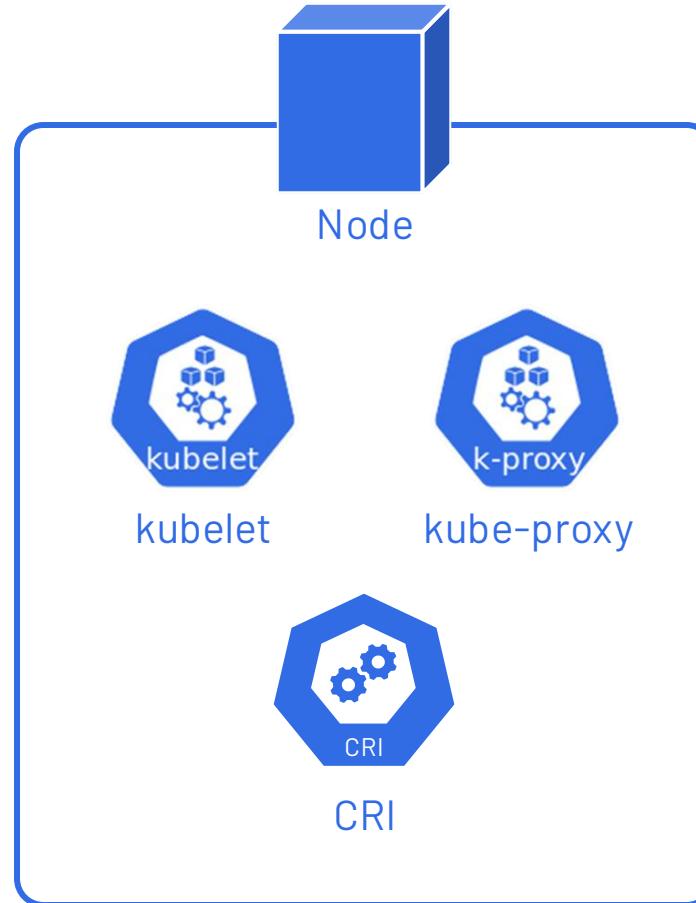
Node Components



- ✓ Act as the engine that powers the containers
- ✓ Popular options - Docker, containerd, and CRI-O
- ✓ Responsible for managing containers operations on the node



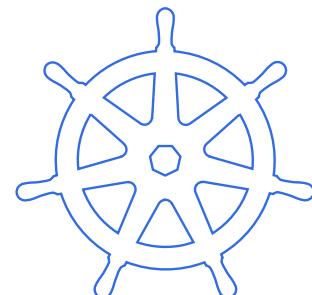
Node Components



Addons



- ✓ Network Plugins
- ✓ Service discovery addons
- ✓ State metrics
- ✓ Kubernetes dashboard



Documentation

Documentation

The screenshot shows a web browser window displaying the Kubernetes Documentation homepage. The URL in the address bar is <https://kubernetes.io/docs/home/>. The page has a dark background with a network-like graphic. At the top, there is a navigation bar with links for Documentation, Kubernetes Blog, Training, Partners, Community, Case Studies, Versions, and English. On the left, there is a sidebar with a search bar and a 'Documentation' section containing links for Available Documentation, Versions, Getting started, Concepts, Tasks, Tutorials, Reference, and Contribute. The main content area features three main sections: 'Understand Kubernetes', 'Try Kubernetes', and 'Set up a K8s cluster'. Each section has associated text and links. On the right side, there are four buttons for editing the page: 'Edit this page', 'Create child page', 'Create an issue', and 'Print entire section'. A blue gear icon is visible in the bottom right corner.

Kubernetes Documentation

Kubernetes Documentation / Documentation

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation ([CNCF](#)).

Understand Kubernetes

Learn about Kubernetes and its fundamental concepts.

[Why Kubernetes?](#)
[Components of a cluster](#)

Try Kubernetes

Follow tutorials to learn how to deploy applications in Kubernetes.

[Hello Minikube](#)
[Walkthrough the](#)

Set up a K8s cluster

Get Kubernetes running based on your resources and needs.

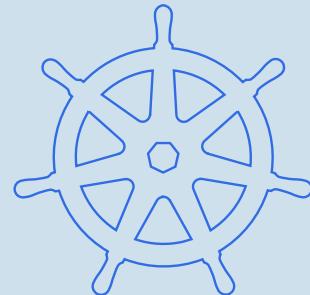
[Learning environment](#)
[Production environment](#)

[Edit this page](#)
[Create child page](#)
[Create an issue](#)
[Print entire section](#)



Demo

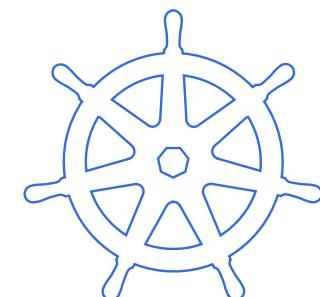
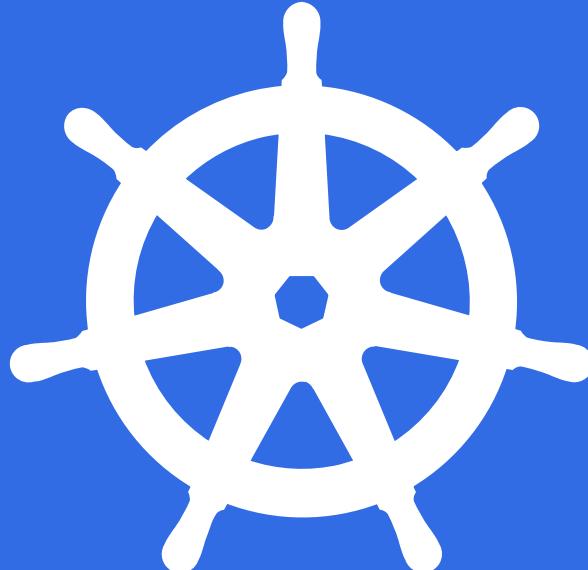
Kubernetes
Documentation



Summary

Summary

- ✓ Overview of Kubernetes components
- ✓ How to navigate the official Kubernetes documentation

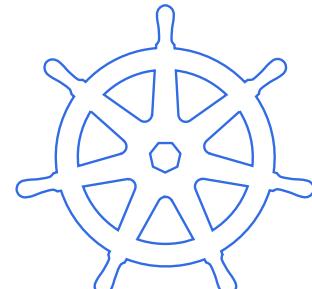


Section: 4

Installation of Kubernetes

Section Overview

- **Setup Kubernetes cluster**
- **Pre-requisites**
- **Demonstration**



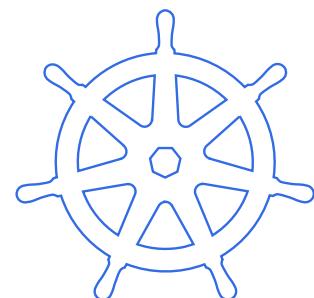
Two Node Cluster



Control Plane/ Master Node



Worker Node



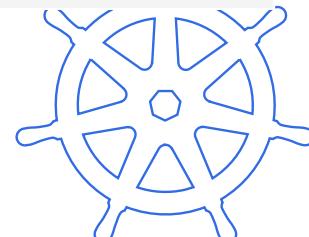
Pre-Requisites (Master Node)

Pre-Requisites (Master Node)



- ✓ Linux Host
- ✓ Memory - 4 GB RAM or more per node
- ✓ CPU - 2 CPUs or more
- ✓ Disable Swap
- ✓ Disable SELinux
- ✓ Network Connectivity
- ✓ Unique Identifiers (hostname or MAC address)
- ✓ Port Requirements

| Protocol | Direction | Port Range | Purpose | Used By |
|----------|-----------|------------|-------------------------|----------------------|
| TCP | Inbound | 6443 | Kubernetes API server | All |
| TCP | Inbound | 2379-2380 | etcd server client API | kube-apiserver, etcd |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10259 | kube-scheduler | Self |
| TCP | Inbound | 10257 | kube-controller-manager | Self |
| TCP | Inbound | 10256 | kube-proxy | Self, Load balancers |



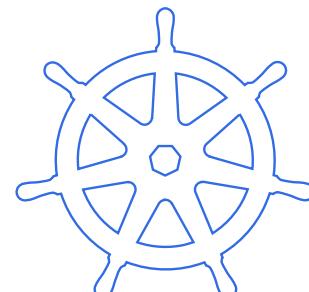
Pre-Requisites (Worker Node)

Pre-Requisites (Worker Node)

- ✓ Linux or Windows Host
- ✓ Memory - 2 GB RAM more per node
- ✓ CPU - 2 CPUs or more
- ✓ Disable Swap
- ✓ Disable SELinux
- ✓ Network Connectivity
- ✓ Unique Identifiers (hostname or MAC address)
- ✓ Port Requirements

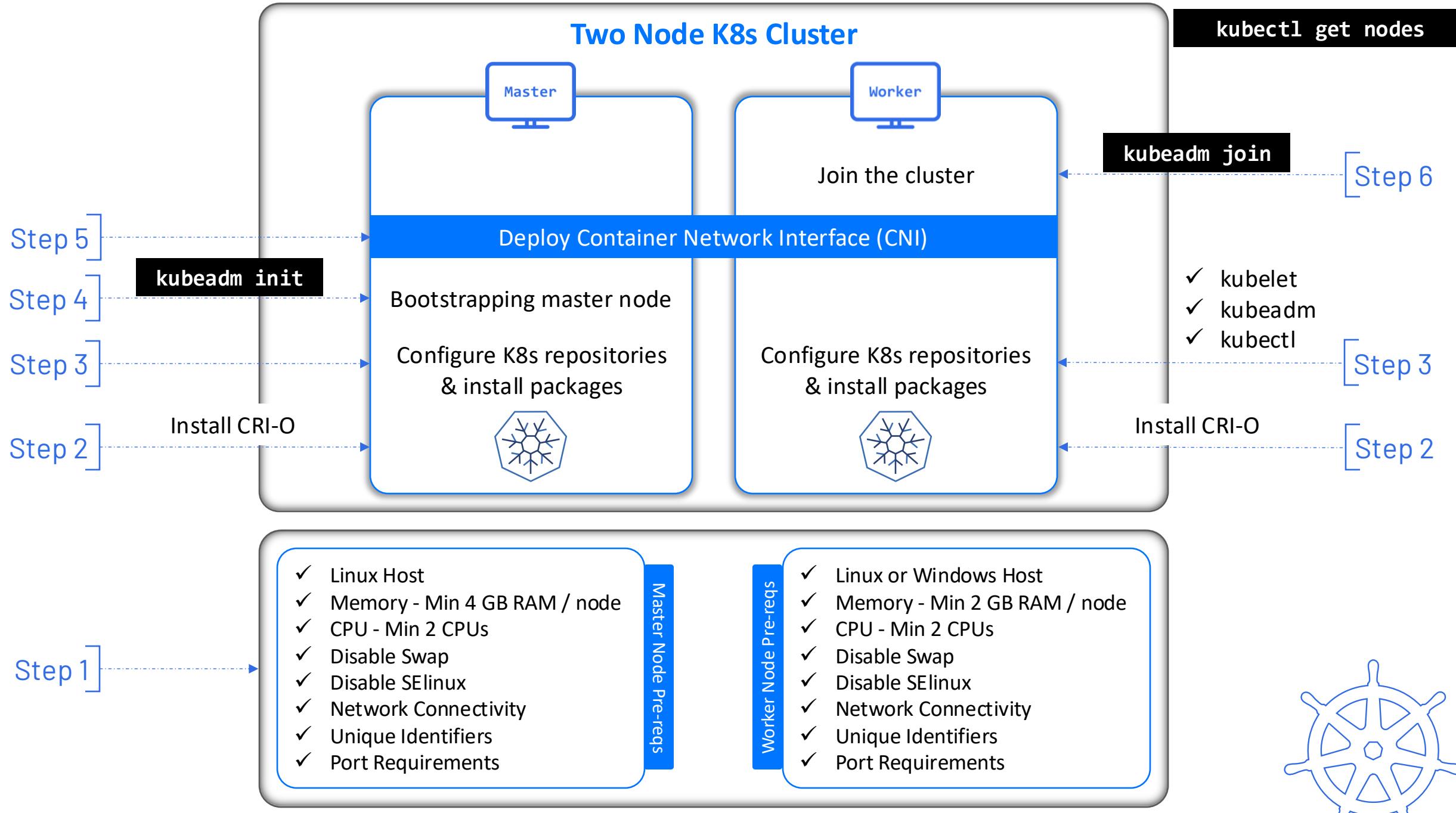


| Protocol | Direction | Port Range | Purpose | Used By |
|----------|-----------|-------------|-------------------|----------------------|
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10256 | kube-proxy | Self, Load balancers |
| TCP | Inbound | 30000-32767 | NodePort Services | All |



Setting up Two-node Cluster

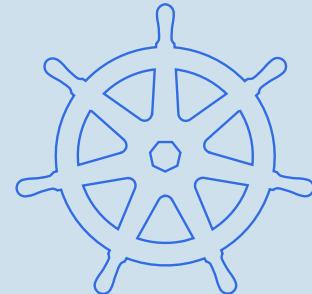
Two Node K8s Cluster





Demo

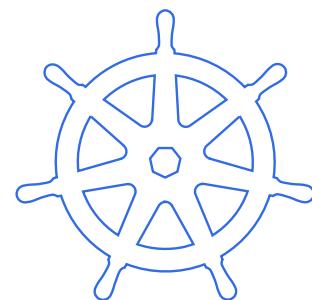
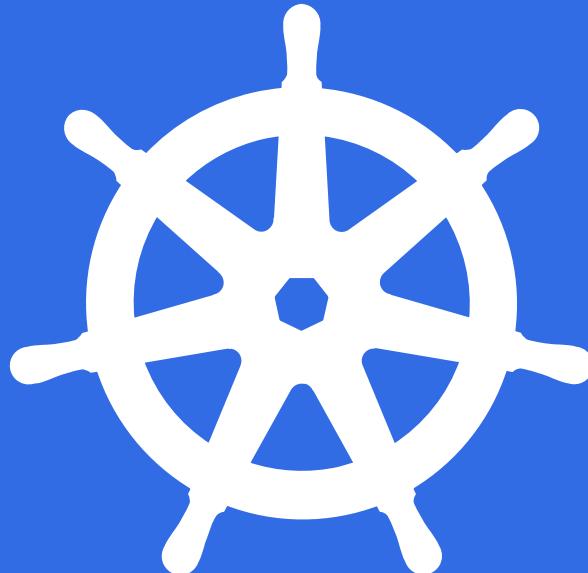
Two Node
Cluster Setup



Summary

Summary

- ✓ Two-node Kubernetes cluster installation

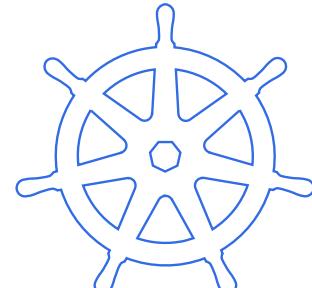


Section: 5

Section Overview

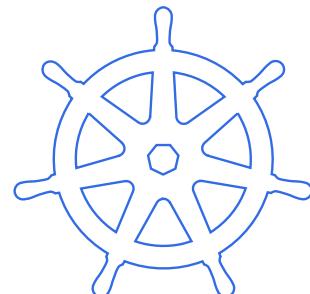
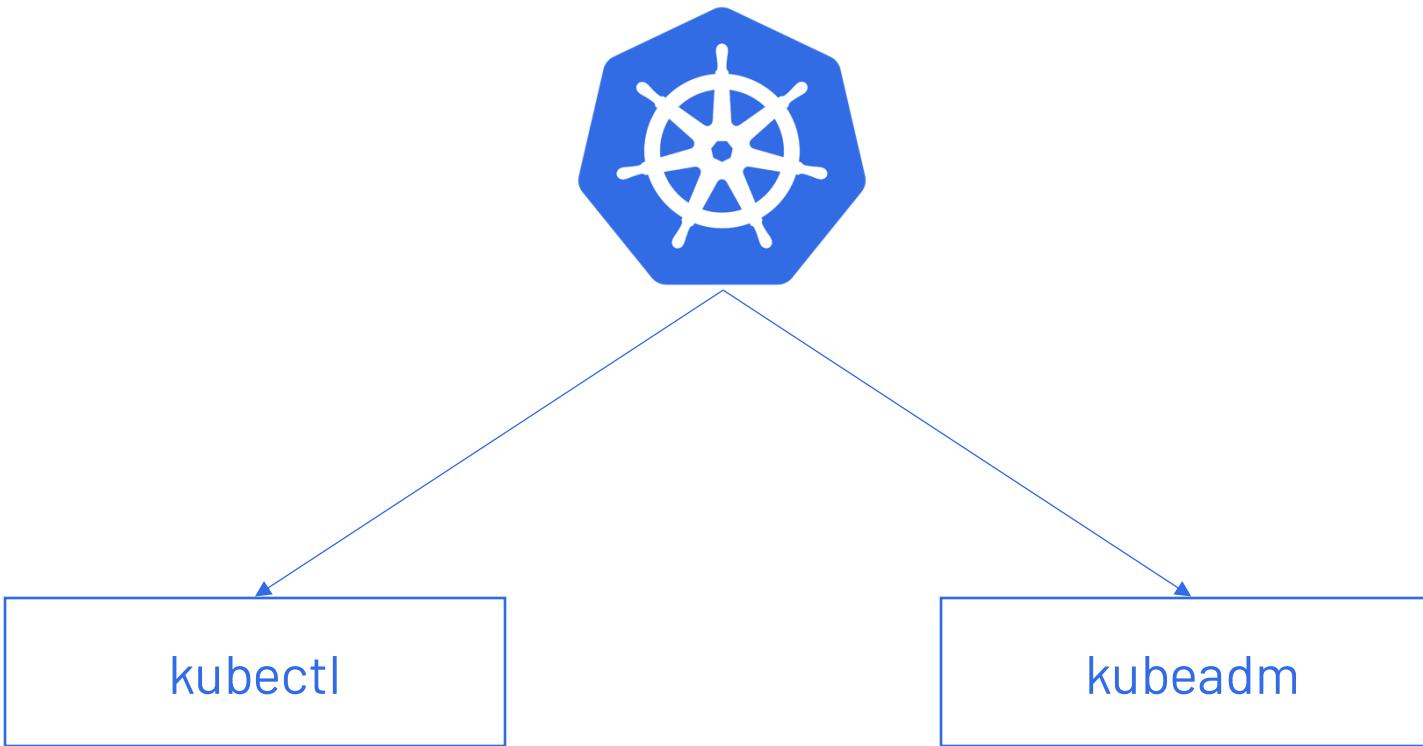
Kubernetes Objects

- ↳ **Namespaces**
- ↳ **Pods**
- ↳ **ReplicaSets**
- ↳ **Deployments**
- ↳ **Labels & Selectors**



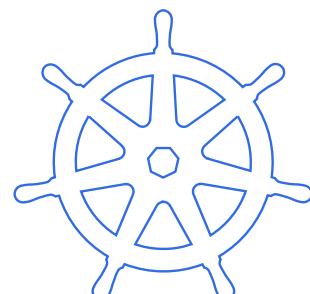
Kubernetes CLI Overview

Kubernetes CLI Overview



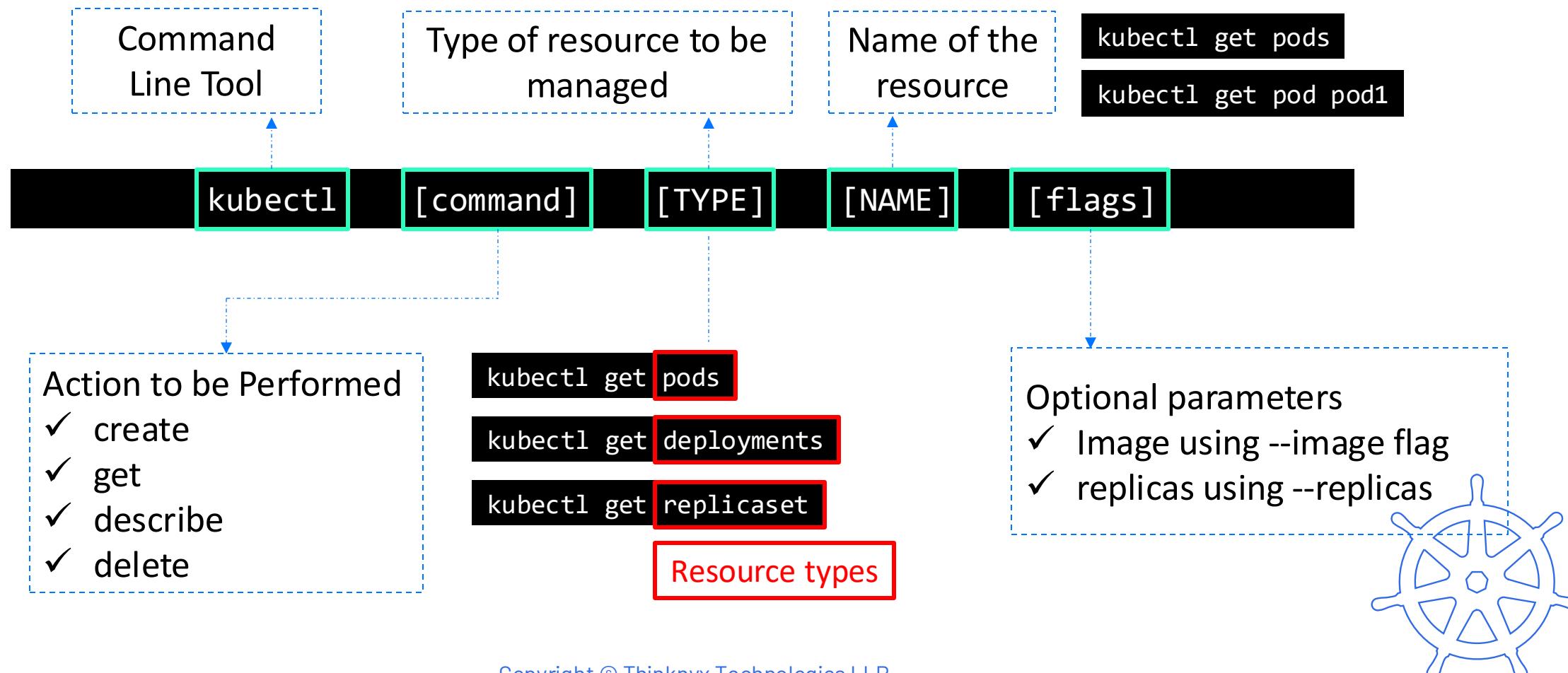
kubectl

- Command-line tool - Communicates with control plane of K8s cluster via K8s API
 - ✓ Create
 - ✓ Update
 - ✓ Delete
 - ✓ Obtain Information

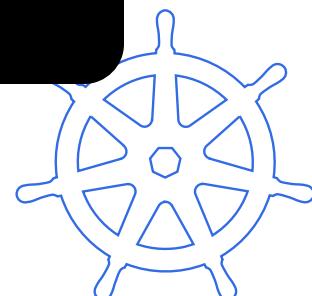
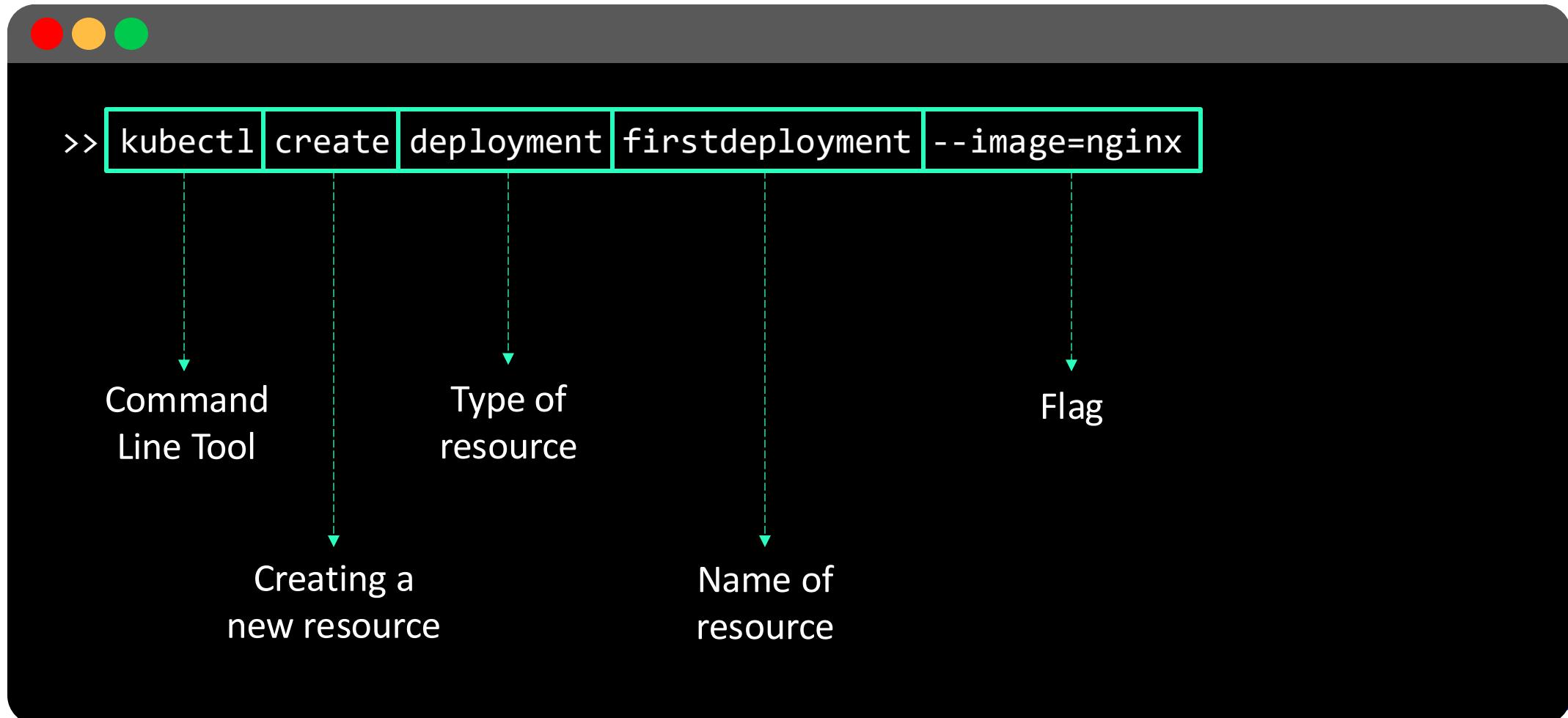


kubectl

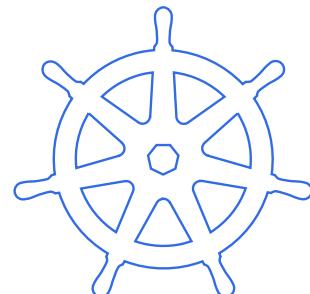
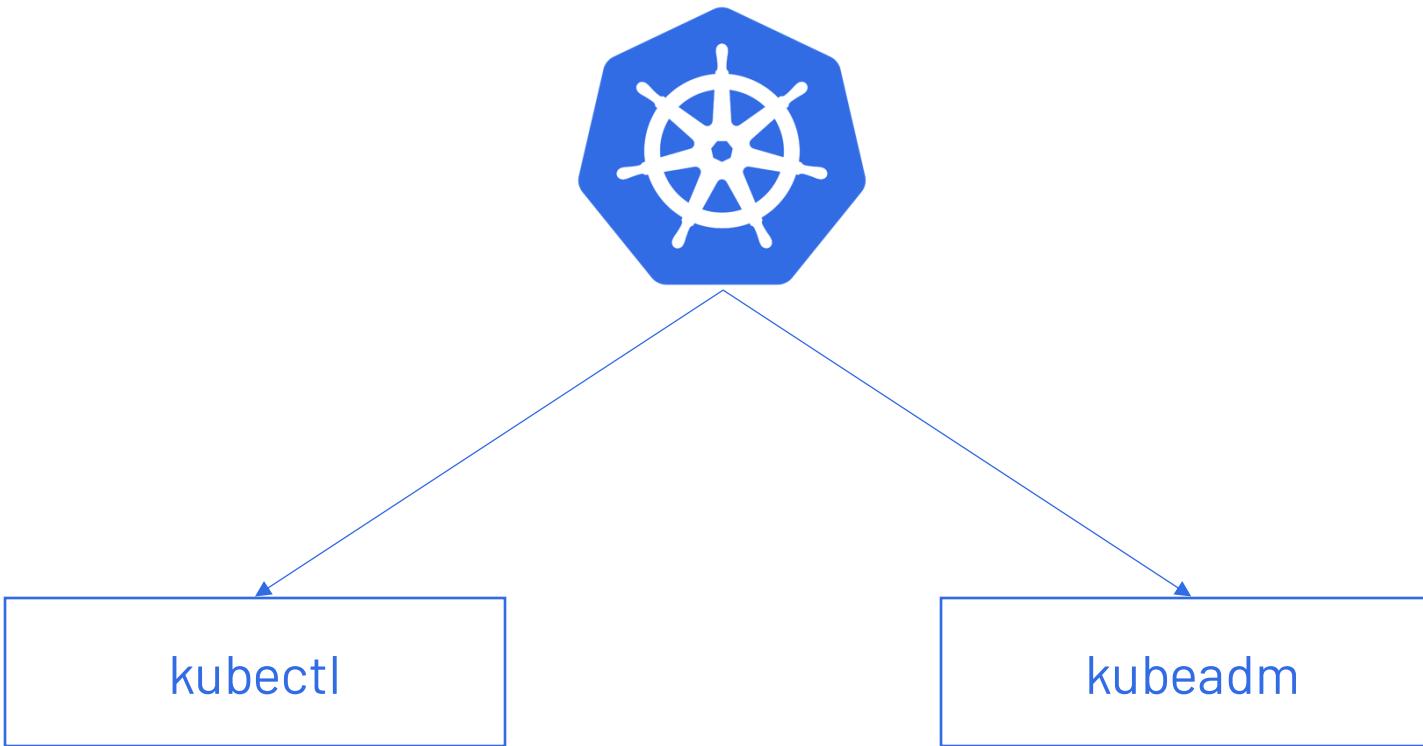
Basic syntax of kubectl command



kubectl

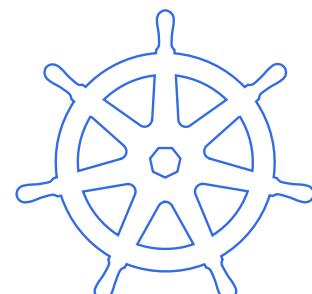


Kubernetes CLI Overview



kubeadm

- Used to build K8s clusters
- Focuses on bootstrapping rather than provisioning machines or installing additional add-ons
 - ✓ Upgrading clusters
 - ✓ Managing tokens
 - ✓ Resetting configurations
 - ✓ Handling certificates
 - ✓ Managing kubeconfig files



Kubernetes Objects

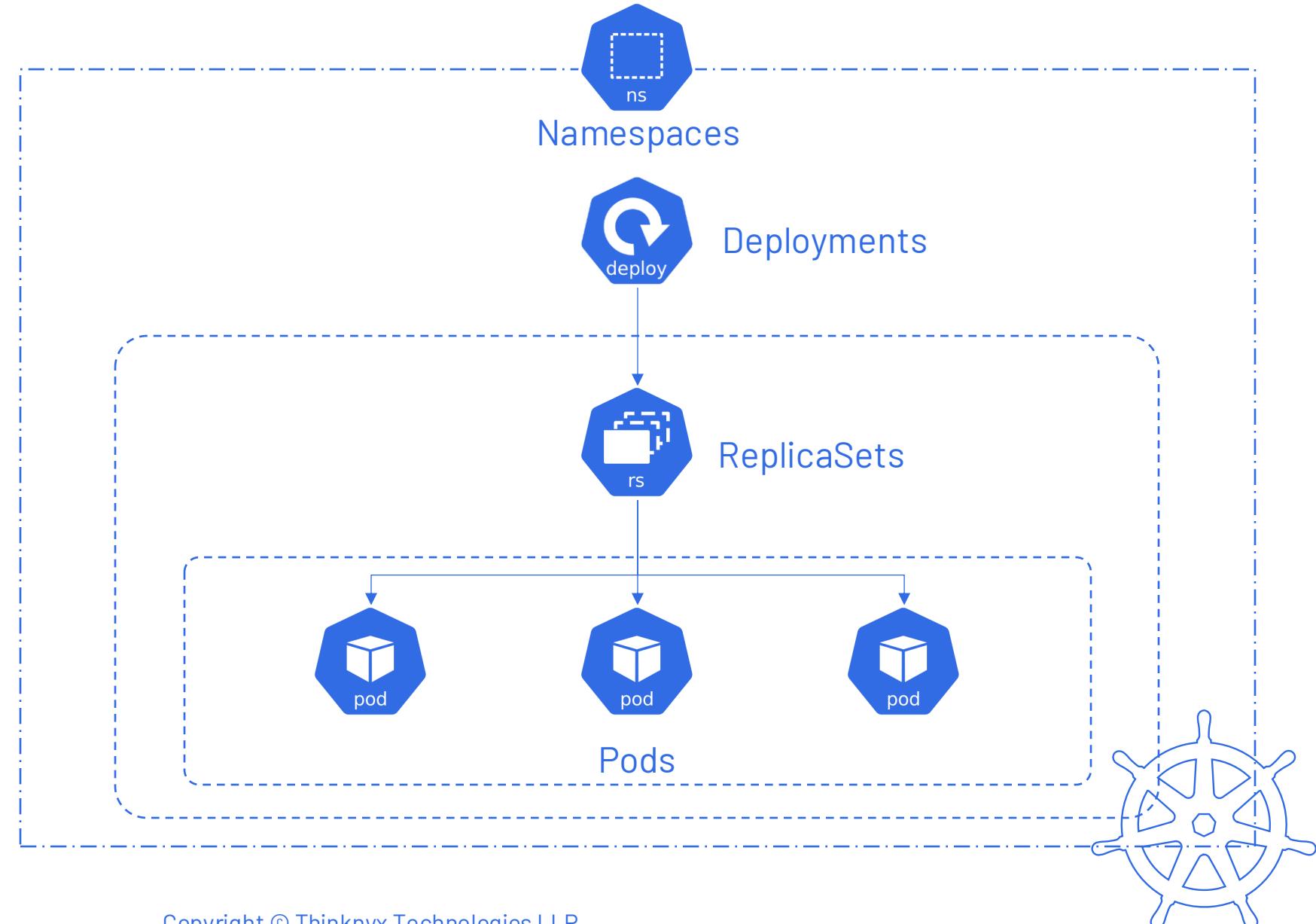
Kubernetes Objects

Provides a way to create logical partitions within a cluster

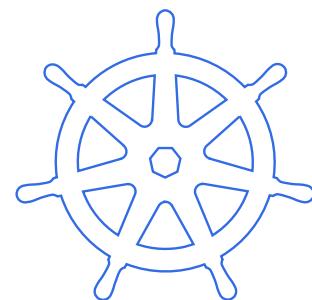
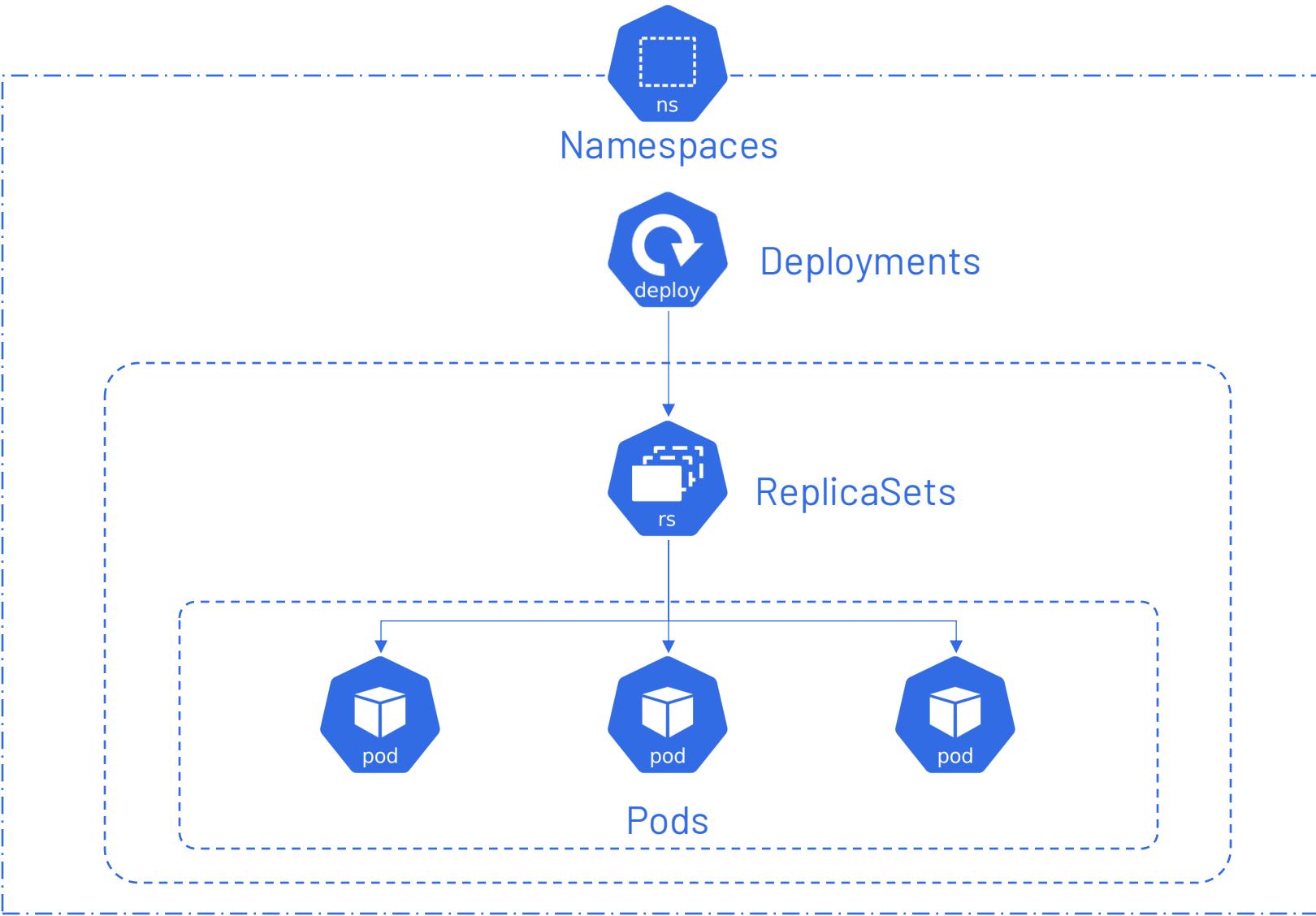
Manage Pod's lifecycle & provide advanced capabilities for application management

Ensure continuous pod availability by automatically replacing failed pods with new ones

Smallest installable units of computing that can be created and managed in K8s



Kubernetes Objects



Namespaces

Namespaces

Default Namespaces

Creates four default namespaces

- default
- kube-node-lease
- kube-public
- kube-system

to manage cluster resources

Resource Isolation

Isolate resources in a cluster, preventing interference between users or teams



Resource Organization

Helps in organizing and managing resources specific to application or environment

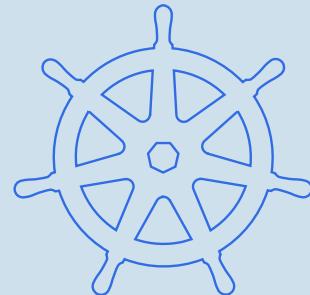
Resource Quotas

Enforce limits on resource usage to avoid excessive consumption by any single team





Demo | Namespaces



Pods

Pods

Ephemeral Nature

Dynamically created, destroyed, or replaced in response to scaling needs, resource limitations, or node failures

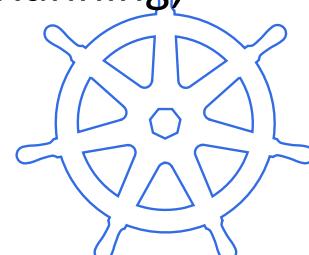


Group of Containers

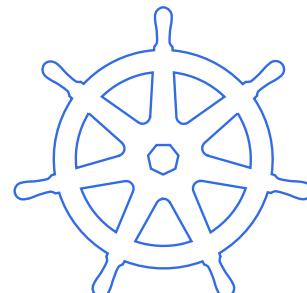
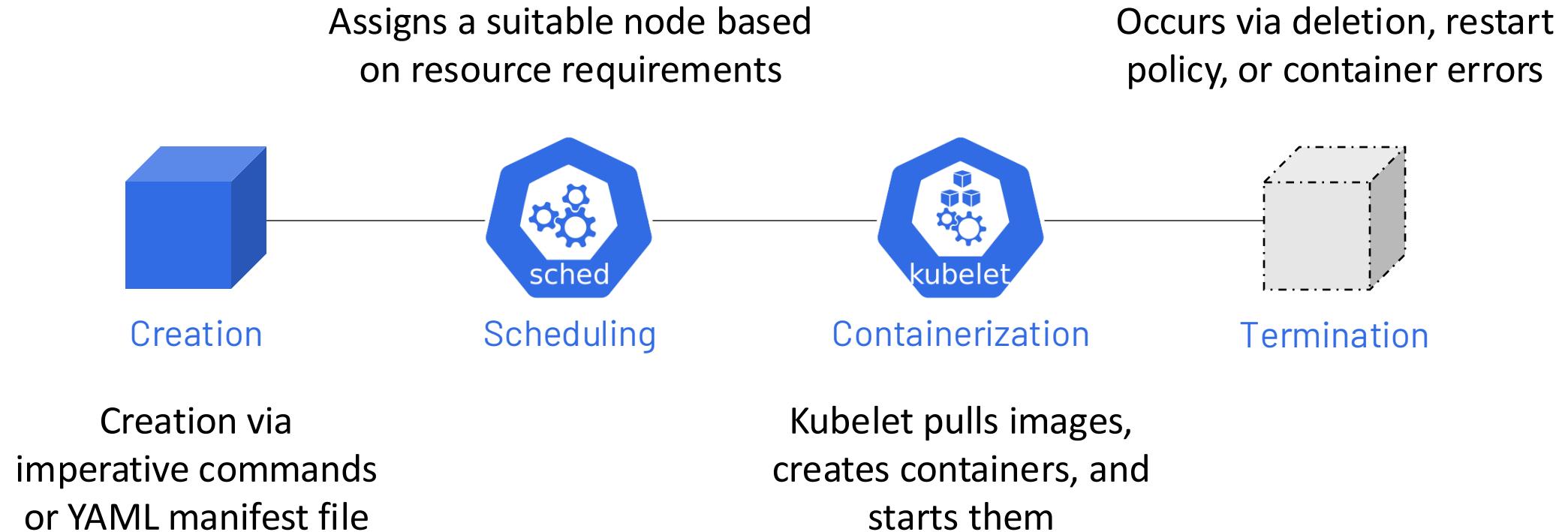
Groups of one or more tightly coupled containers that share the same network namespace and storage

Lifecycle

Go through phases like Pending, Running, Succeeded, or Failed

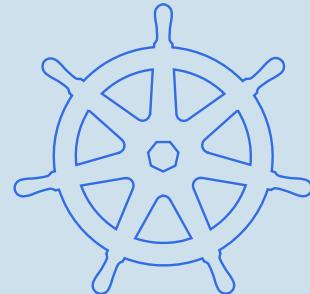


Pods



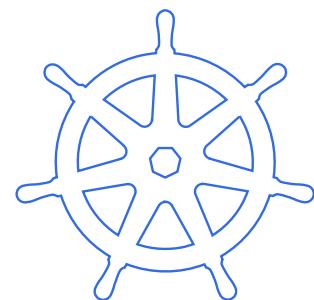
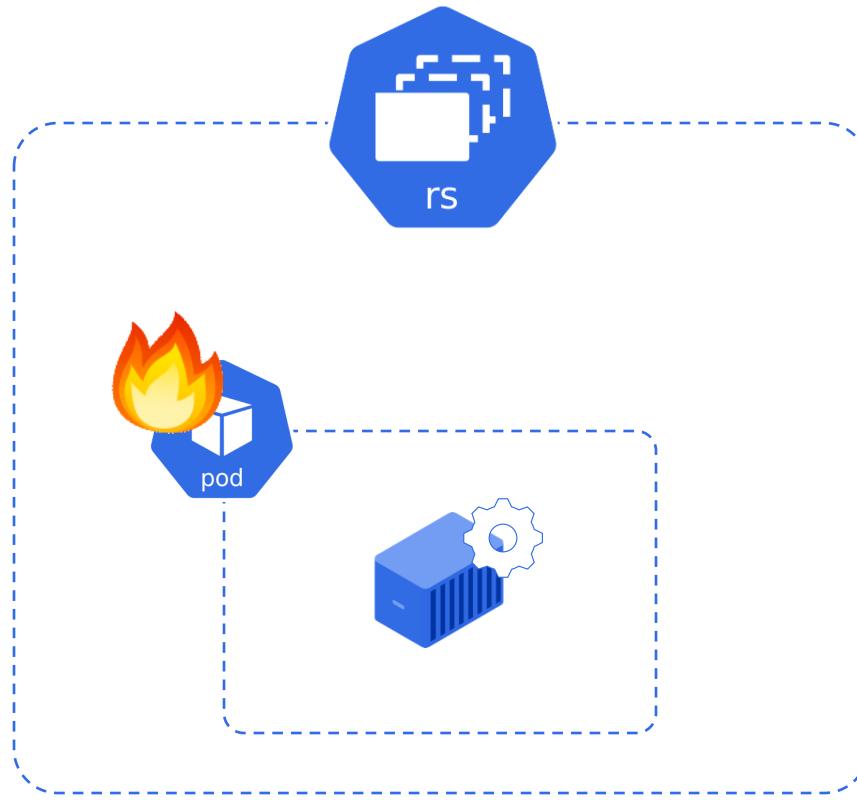


Demo | Pods



ReplicaSets

ReplicaSets

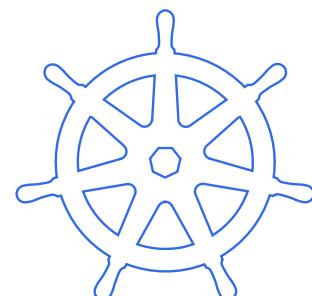


How ReplicaSets Works?

Define
ReplicaSet
Object

Monitor
running Pods

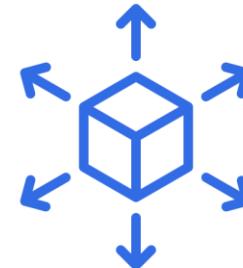
Effortless
Scaling



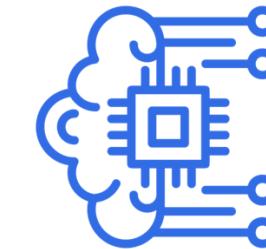
Benefits of ReplicaSets



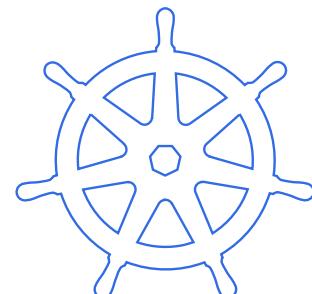
High Availability



Scalability

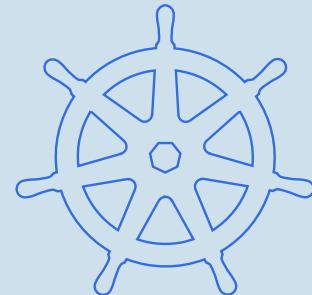


Self-healing





Demo | ReplicaSets



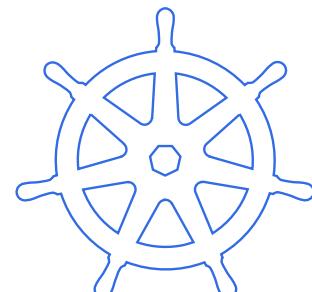
Deployments

Deployments

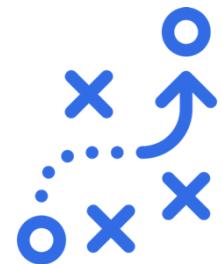
Offers advanced features such as rolling updates & rollbacks, makes it easier to manage the application lifecycle



Provides declarative updates to applications and ensures pods are running at a given time



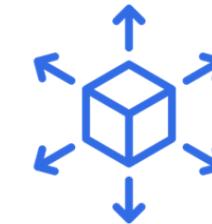
Benefits of Deployments



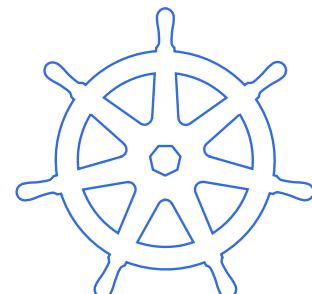
Update Strategies



Rollbacks



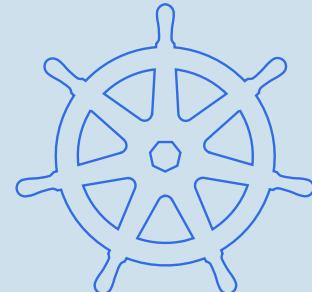
Scaling





Demo

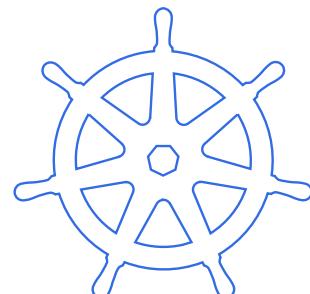
Deployments



Labels & Selectors

Labels & Selectors

- Provides a way to categorize & target resources within a cluster
 - ✓ Labels categorize resources in Kubernetes with identifying tags
 - ✓ Selectors filter resources based on their assigned labels



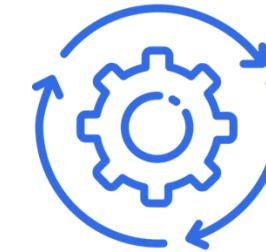
Benefits of Labels & Selectors



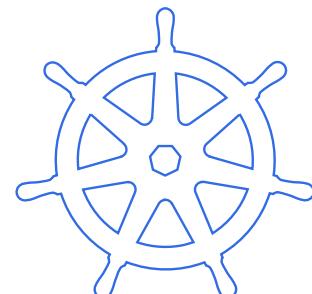
Categorization
/Organization



Selection



Automation



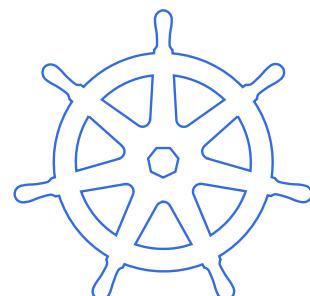
Key Concepts

Key-value pairs for classifying
Kubernetes resources

Labels

Selectors

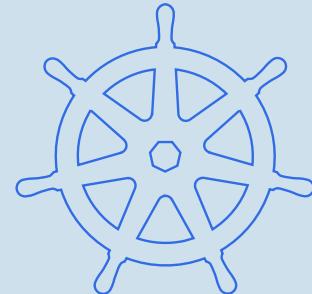
Filter resources based on labels
key-value comparisons





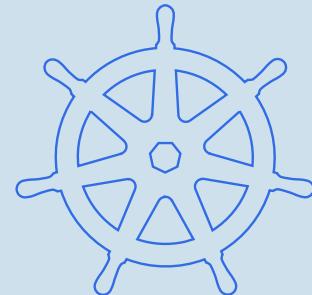
Demo

Labels &
Selectors





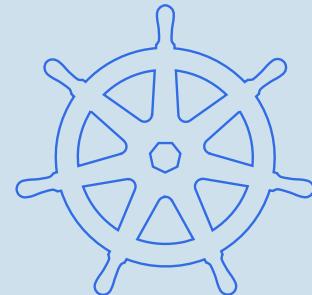
Demo | Inbuilt Labels





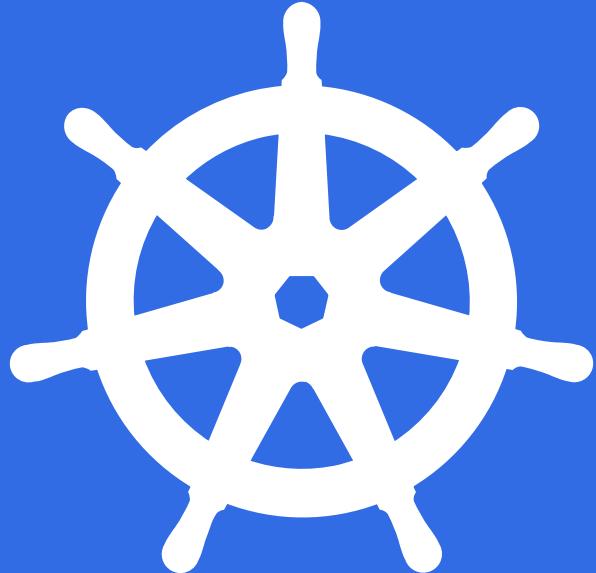
Demo

Useful kubectl Tips

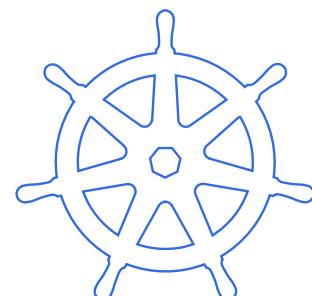


Summary

Summary



- ✓ Kubernetes Objects:
 - Namespaces
 - Pods
 - ReplicaSets
 - Deployments
- ✓ Labels and Selectors
- ✓ Inbuilt Labels
- ✓ Practical kubectl tips

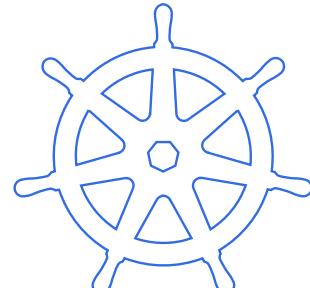


Section: 6

Kubernetes Networking

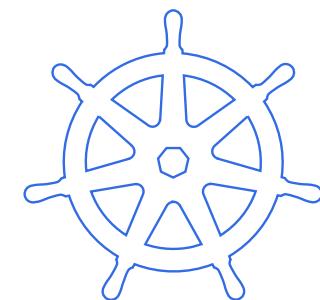
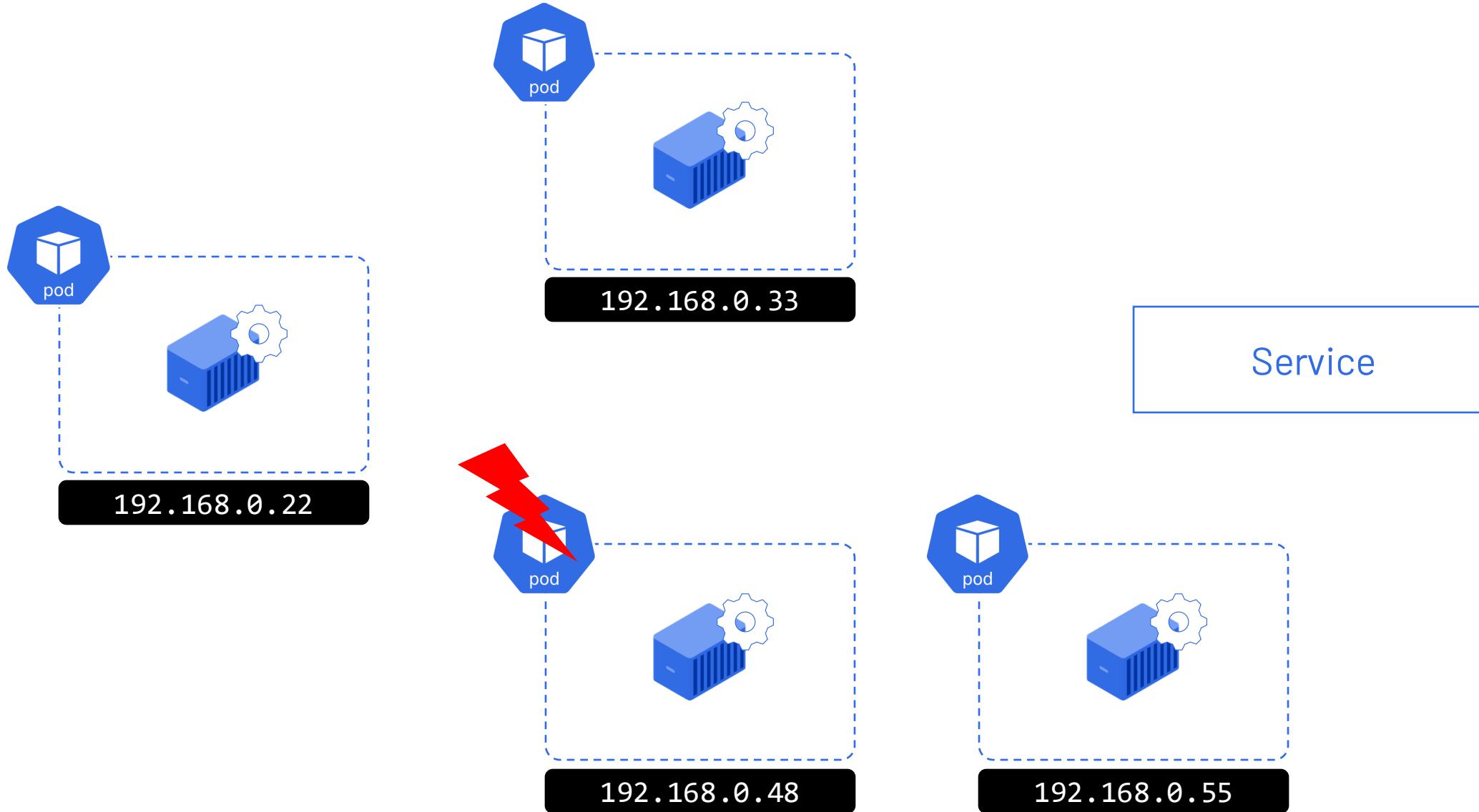
Section Overview

- **Service Object**
- **Service Object Types**
- **Hands-on Demonstrations**

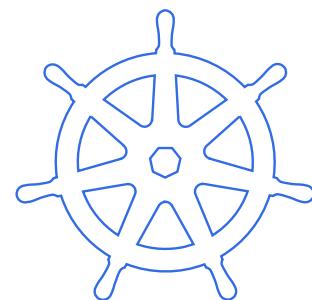
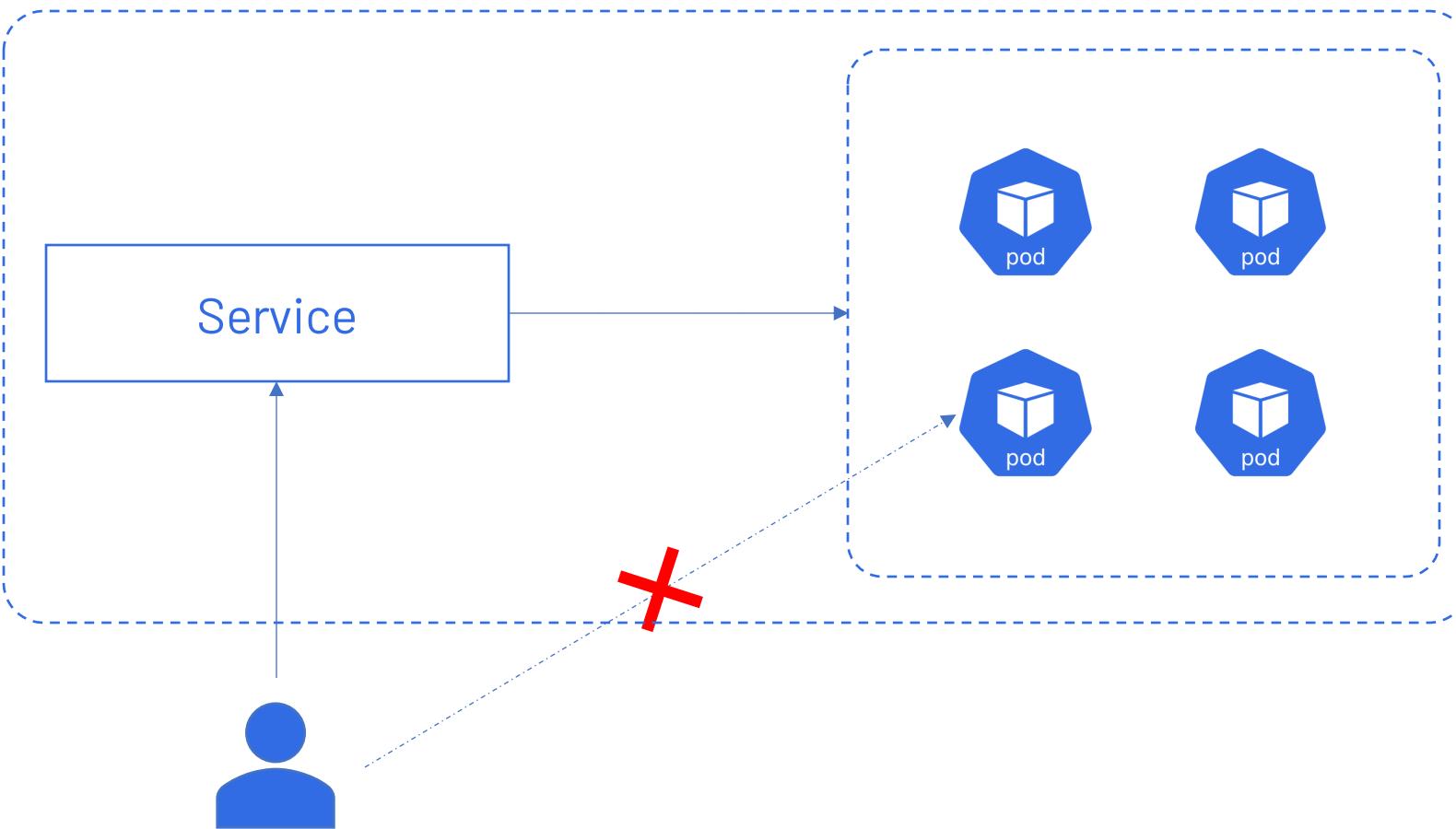


Services

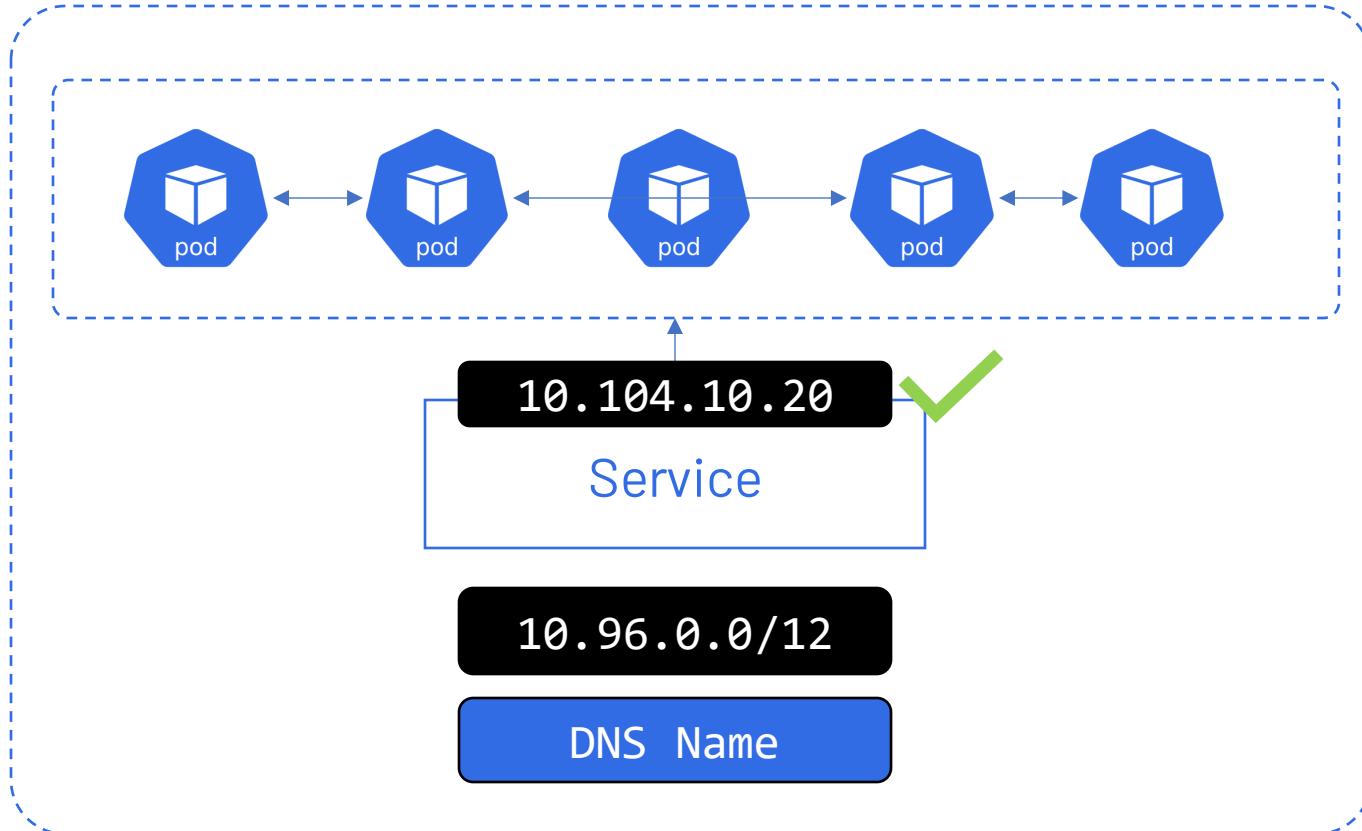
Services



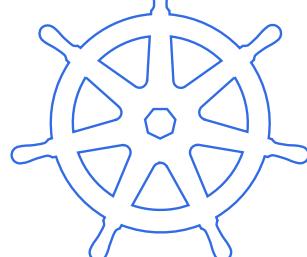
Services



Services

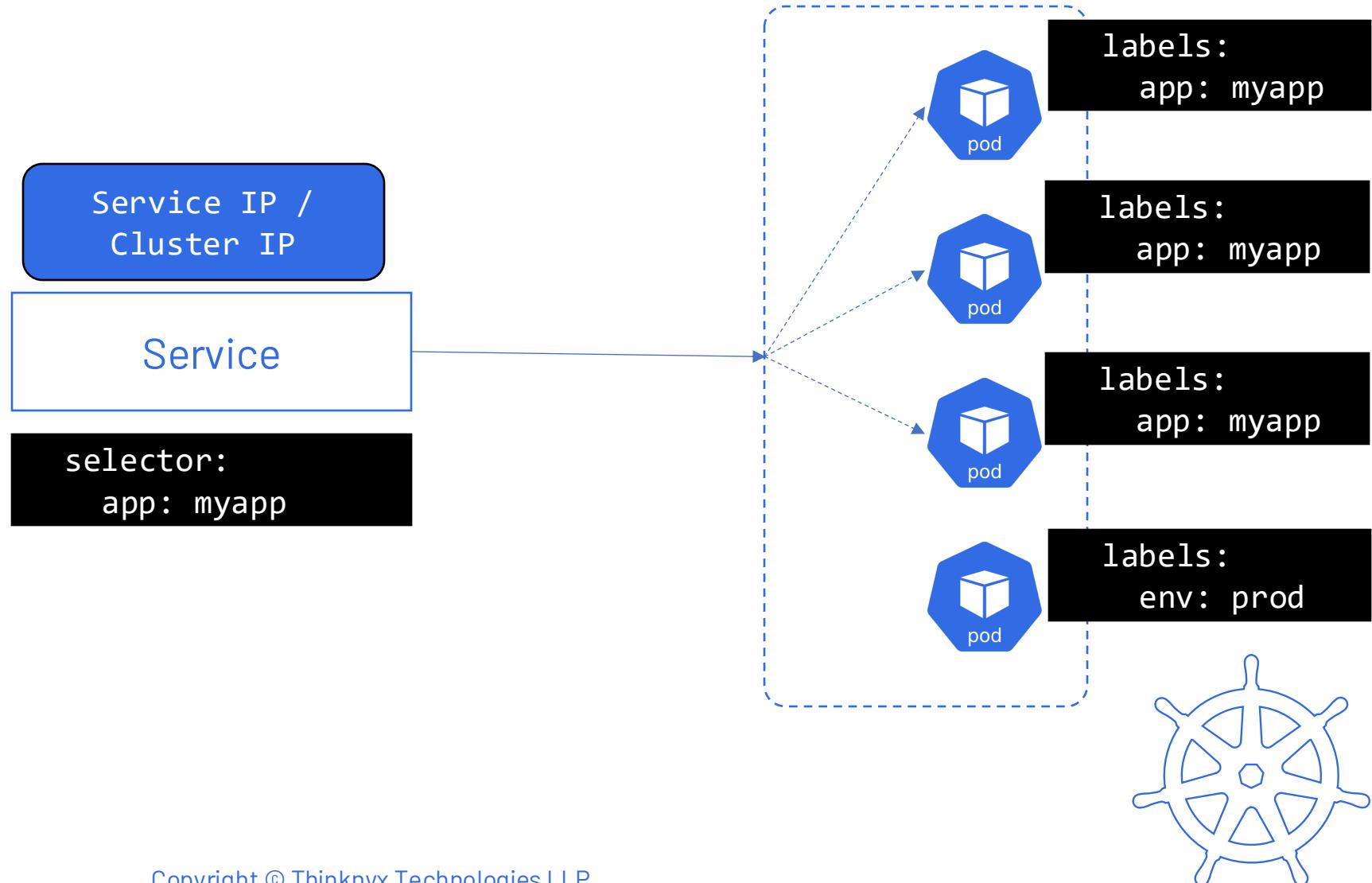


- ✓ ClusterIP
- ✓ NodePort
- ✓ LoadBalancer



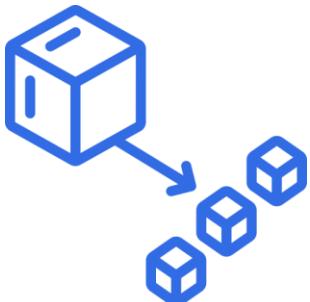
Services

Services can
perform Internal
load balancing

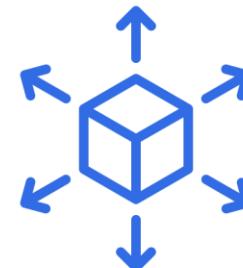


Benefits of Service Objects

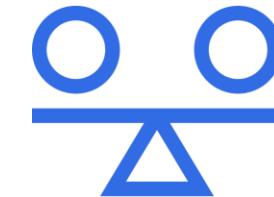
Benefits of Service Objects



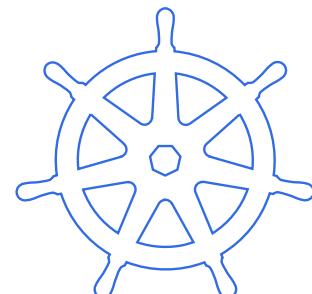
Decoupling



Scalability



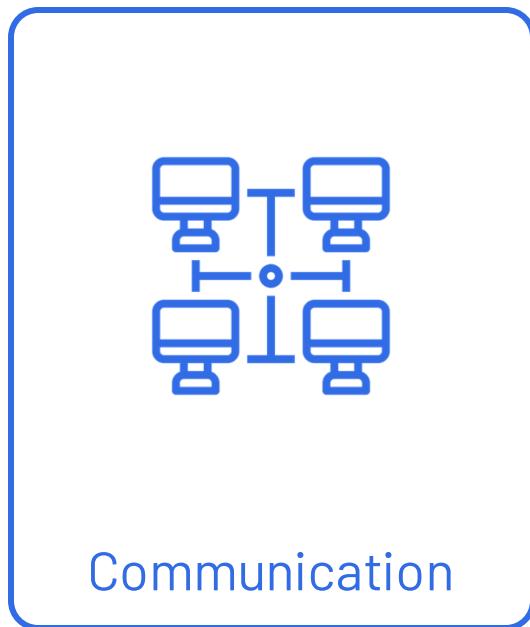
Stability



ClusterIP

ClusterIP

- Expose applications exclusively to other pods within the same cluster for internal communication



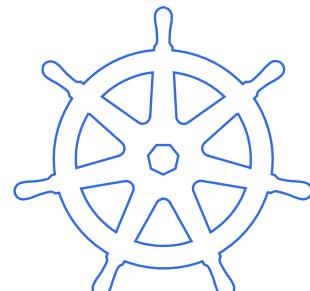
Communication



Security

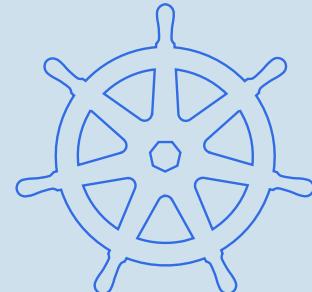


Limitations





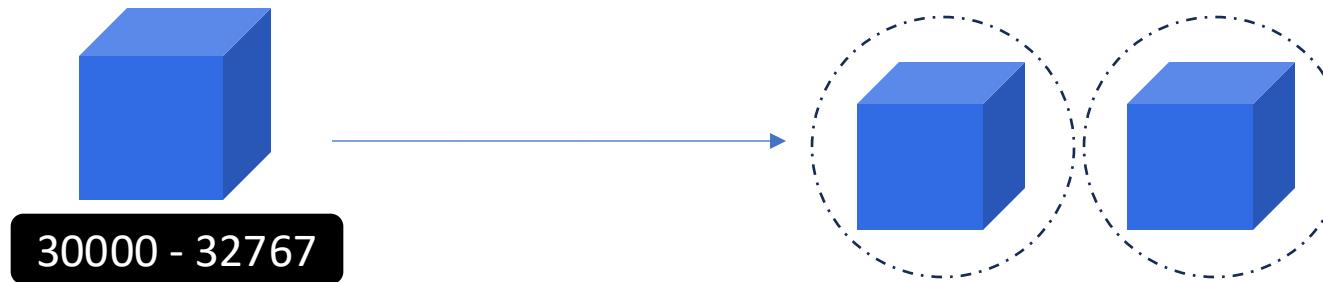
Demo | ClusterIP
Service Type



NodePort

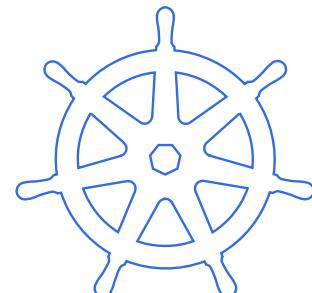
NodePort

- Expose application on a static port across all nodes in the cluster
- Default port range is 30000 – 32767



NodePort Benefits

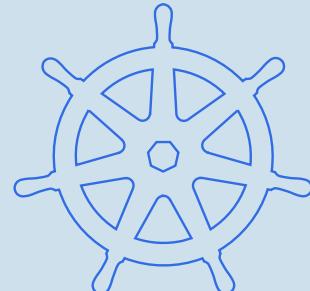
- ✓ Simple External Access
- ✓ No Load Balancer Required





Demo

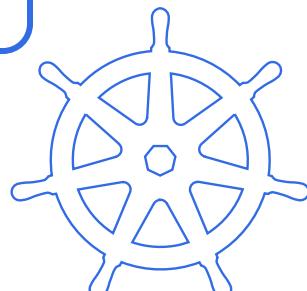
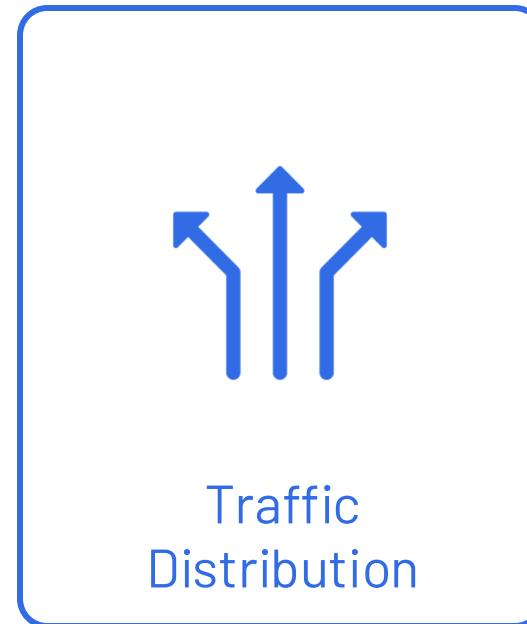
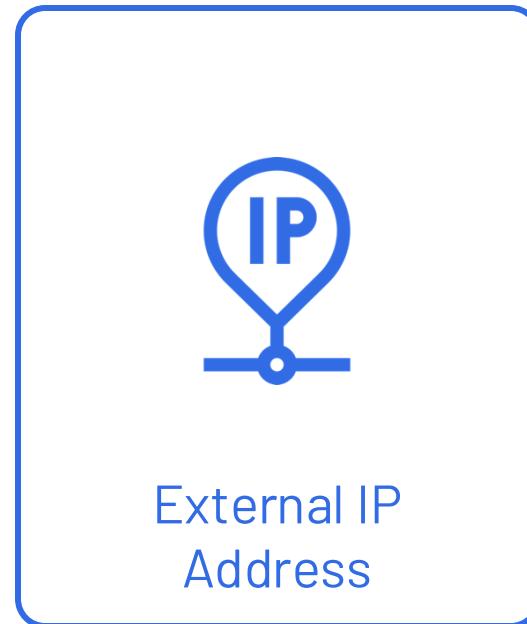
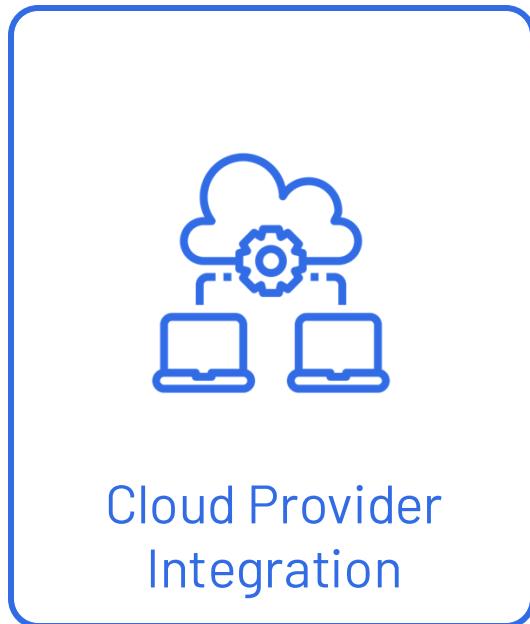
NodePort Service Type



LoadBalancer

LoadBalancer

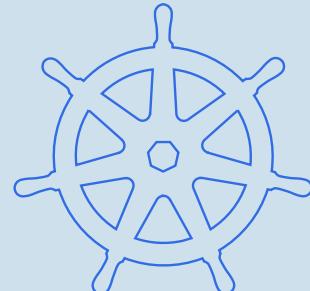
- Leverages external or cloud provider-specific load balancers to distribute incoming traffic across pods
- Highly available & scalable solution for exposing application to the public internet





Demo

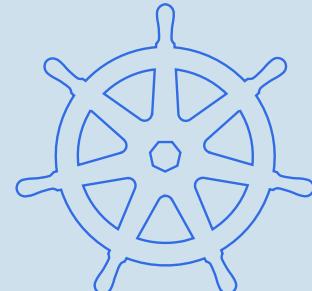
LoadBalancer
Service Type





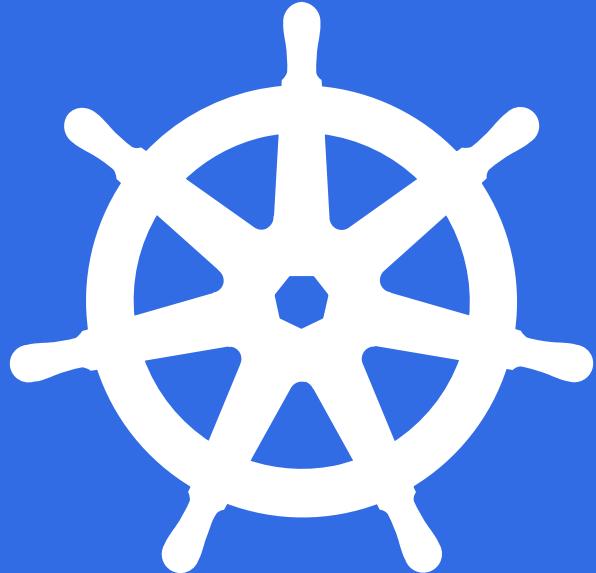
Demo

Troubleshooting
Deployment and Service

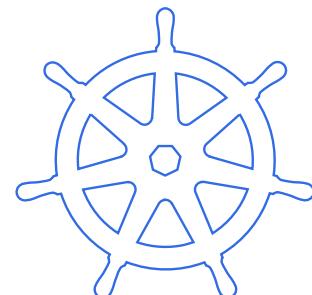


Summary

Summary



- ✓ Service Types:
 - ClusterIP
 - NodePort
 - LoadBalancer
- ✓ Troubleshooting techniques for deployments & services

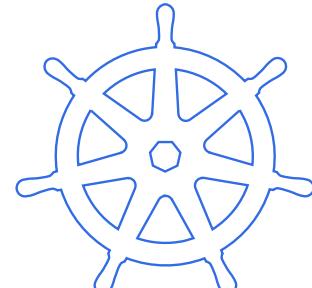


Section: 7

Kubernetes Manifest Files

Section Overview

- ↳ Kubernetes API ecosystem
- ↳ How to develop K8s object manifest files?
- ↳ Smart way to develop K8s manifest files

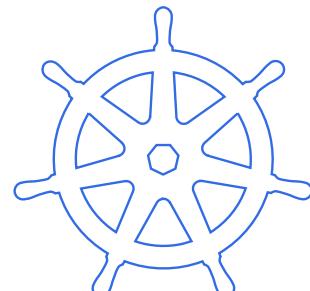
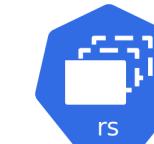


Understanding Kubernetes API Ecosystem

Understanding Kubernetes API Ecosystem

The Kubernetes API Ecosystem: A Structured Playground for Container Orchestration

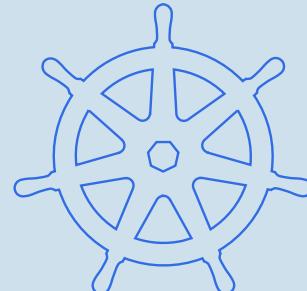
- Serves as central nervous system for any internal & external communication within or outside a cluster
- Offers a programmatic interface for controlling & managing every aspect of a cluster





Demo

Kubernetes API ecosystem & developing object definition files

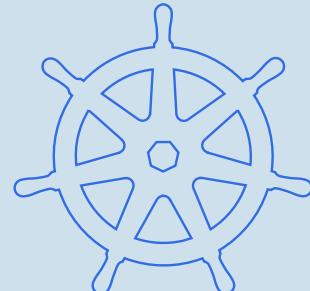


YAML Basics



Demo

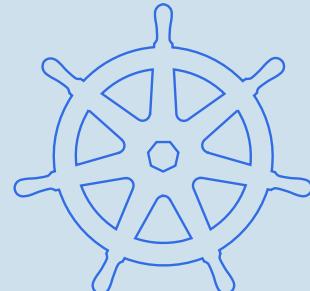
Deployment & Service
Creation





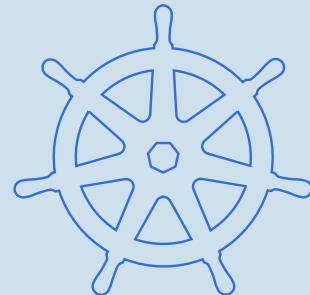
Demo

Smart way to create
Kubernetes objects





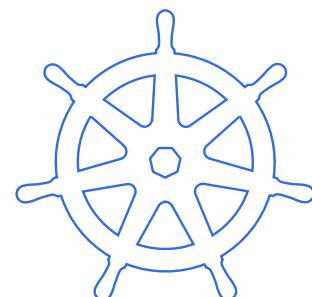
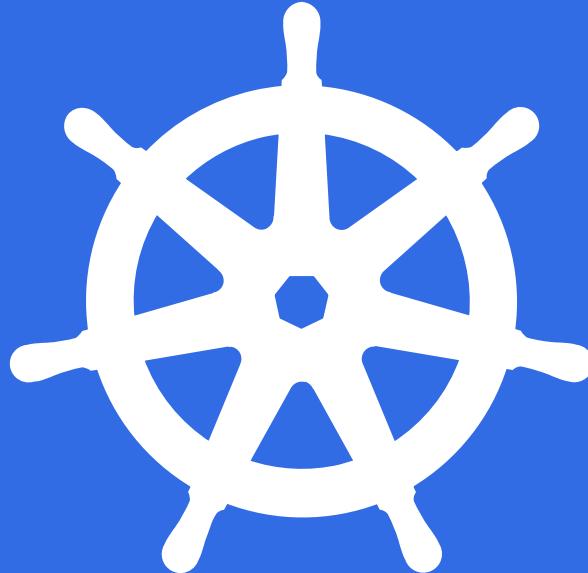
Demo | kubectl



Summary

Summary

- ✓ Create & understand object definition files
- ✓ YAML basics
- ✓ Create Deployment and Service object files
- ✓ Efficient YAML creation methods
- ✓ kubectl explain

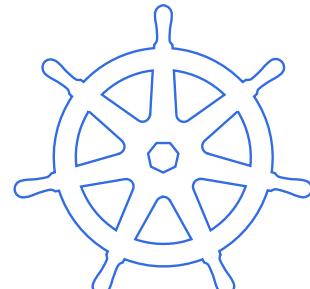


Section: 8

Update Strategies & Rollback

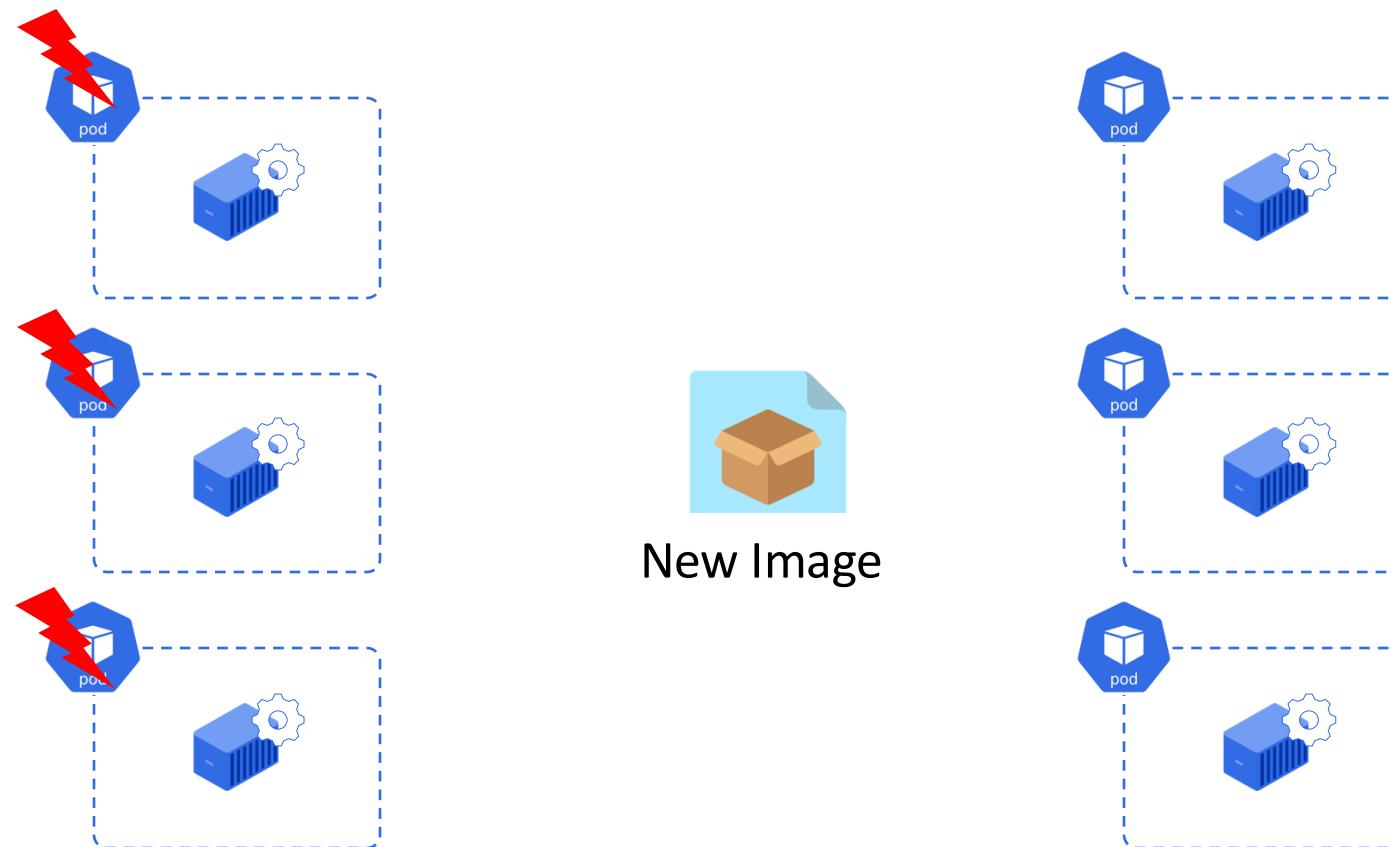
Section Overview

- ↳ Recreate and Rolling Strategies
- ↳ Understanding Rollback in K8s

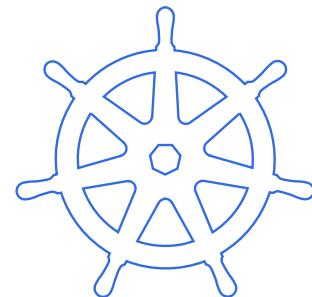


Recreate Strategy

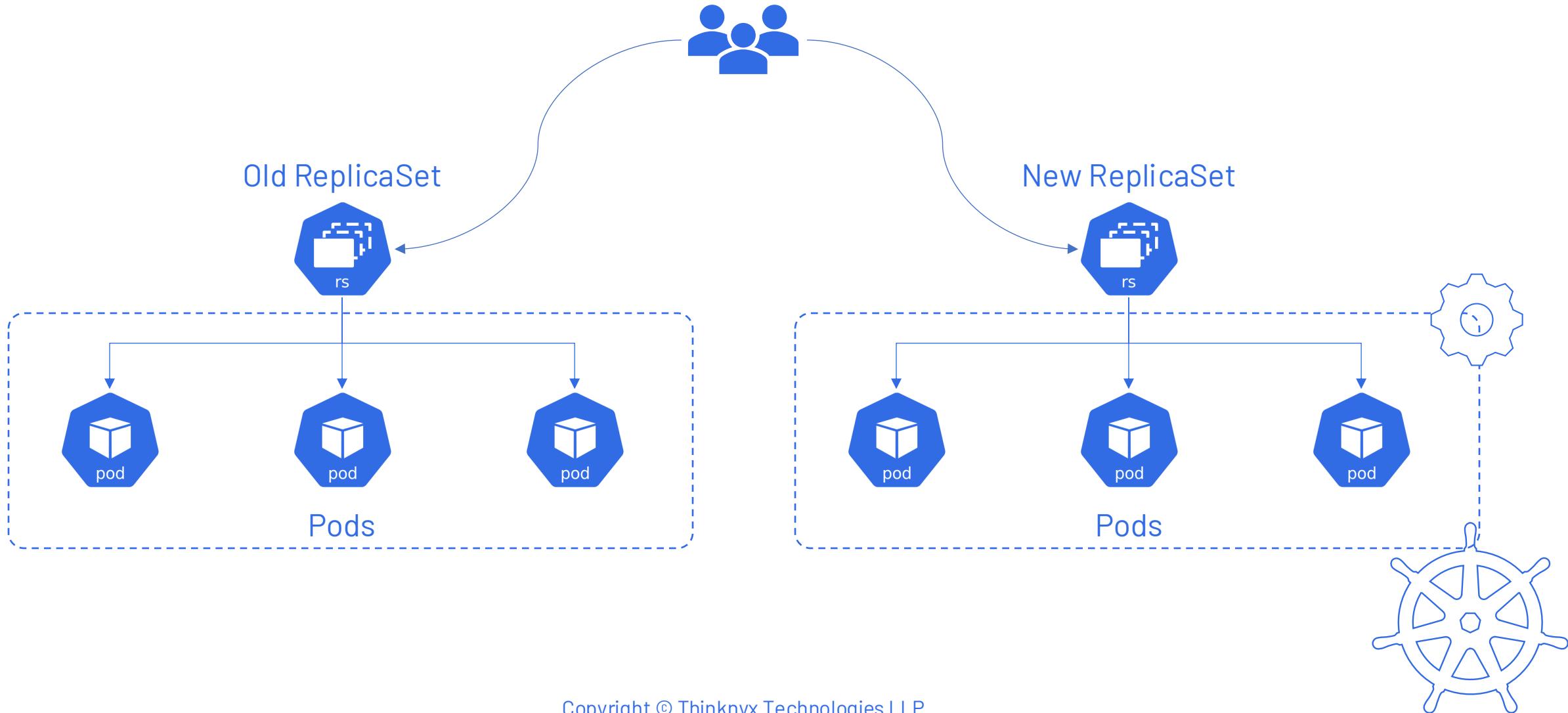
Recreate Strategy



- ✓ Quick Update but lead to downtime



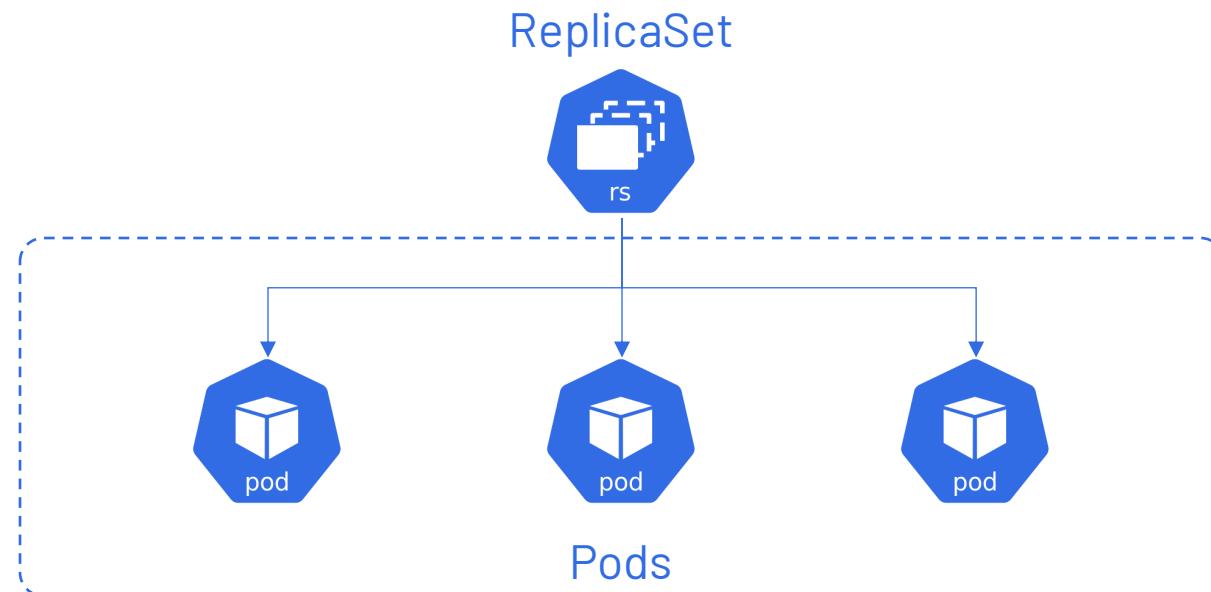
Recreate Strategy



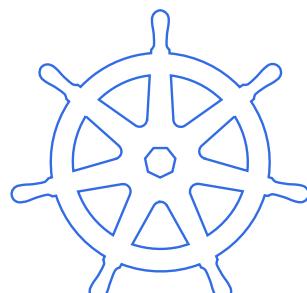
Recreate Strategy

Deployment Configuration

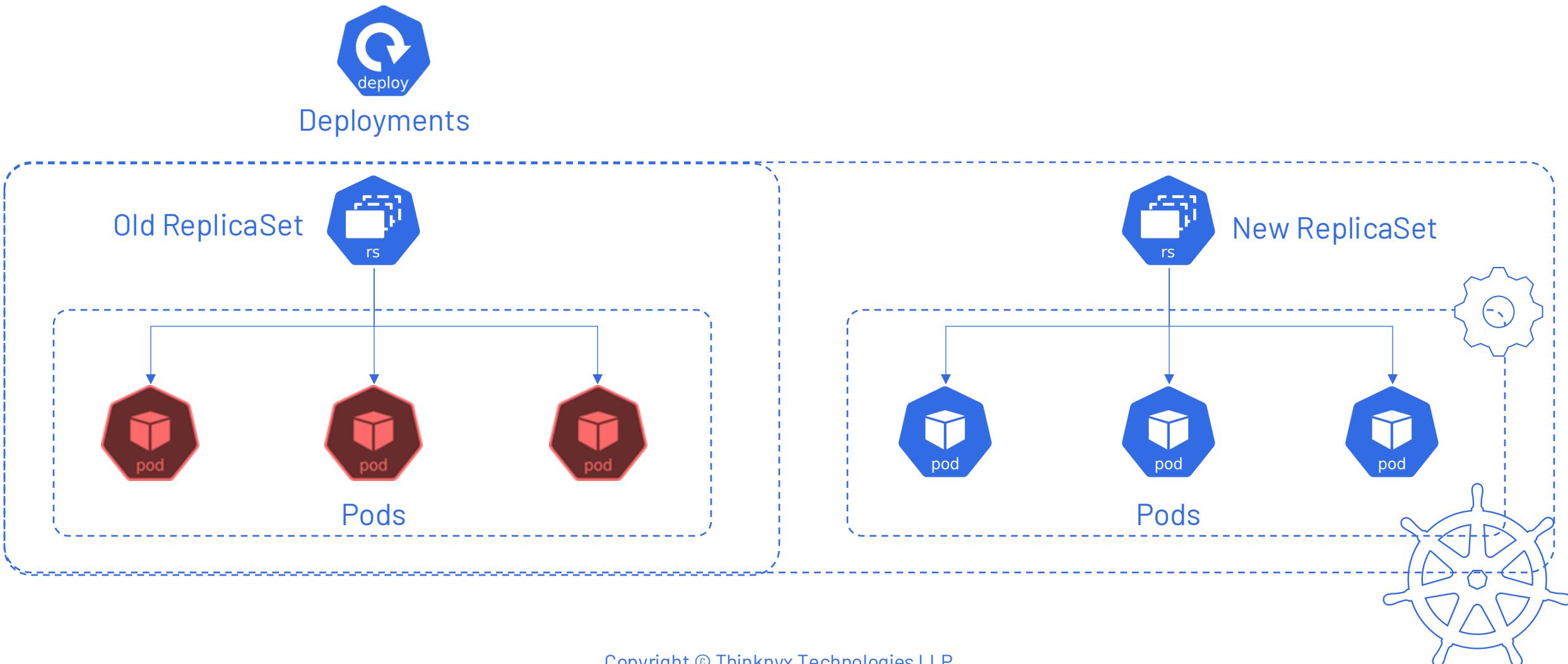
- ✓ Image version
- ✓ Number of replicas
- ✓ Other parameters



- ✓ Imperative
- ✓ Declarative



Recreate Strategy



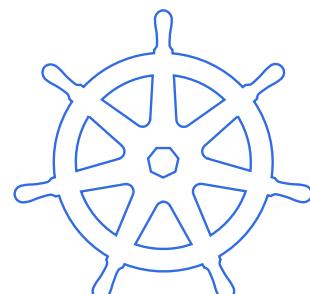
Recreate Strategy

Benefits

- ✓ Deployments are generally faster to implement
- ✓ Update process happens swiftly
- ✓ Rapid delivery of features or fixes

Drawback

- ✓ Downtime

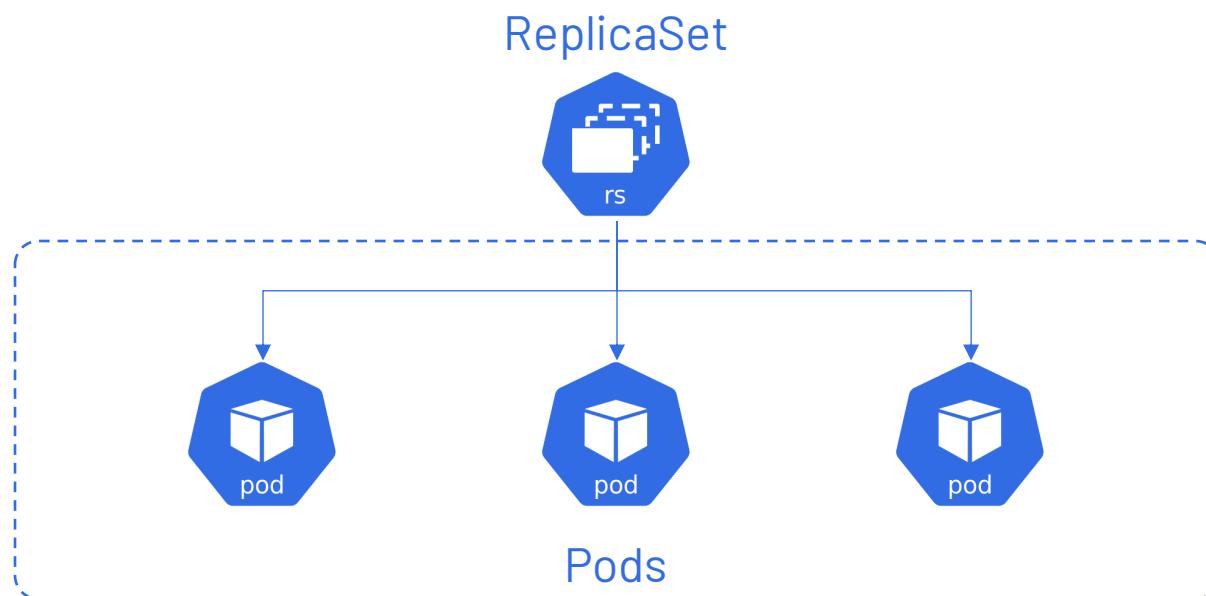


Rolling Update Strategy

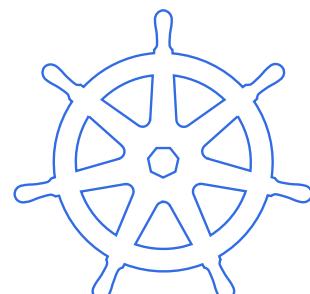
Rolling Update Strategy

Deployment Configuration

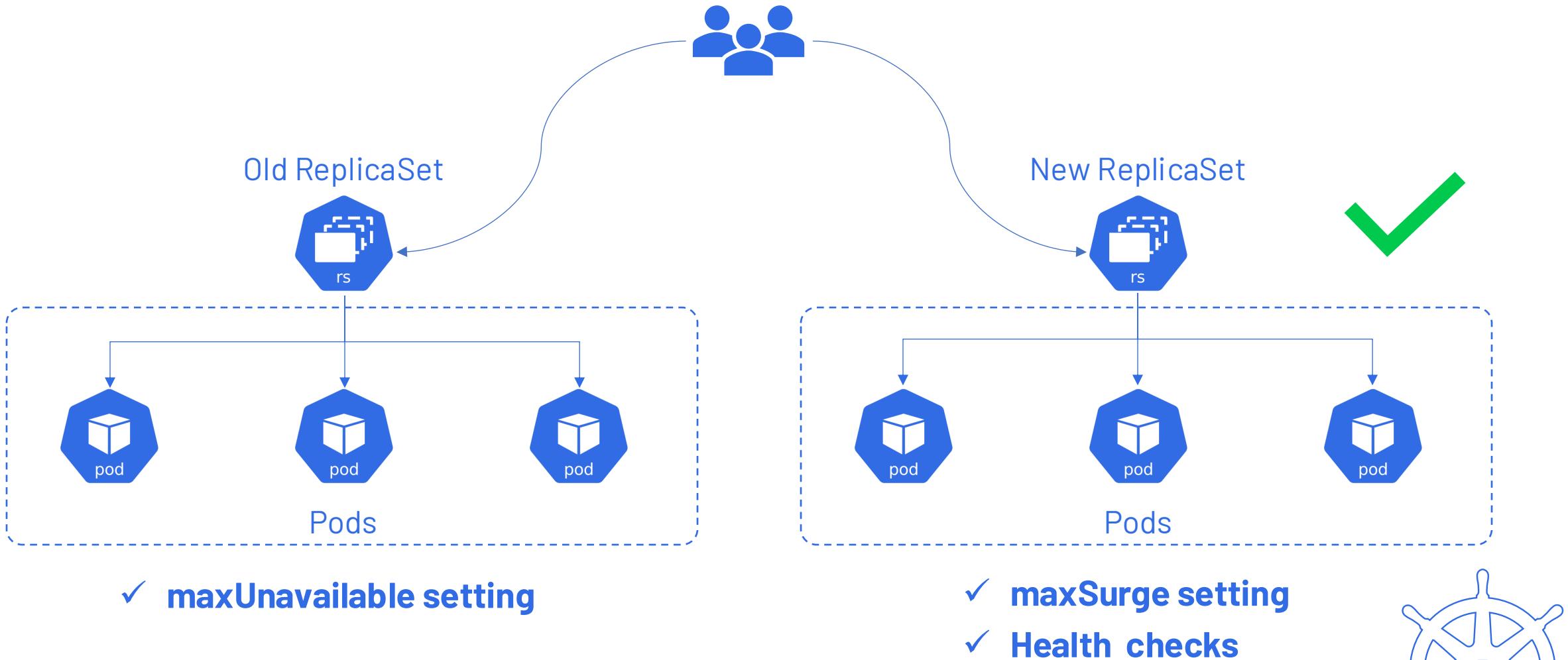
- ✓ Image version
- ✓ Number of replicas
- ✓ Other parameters



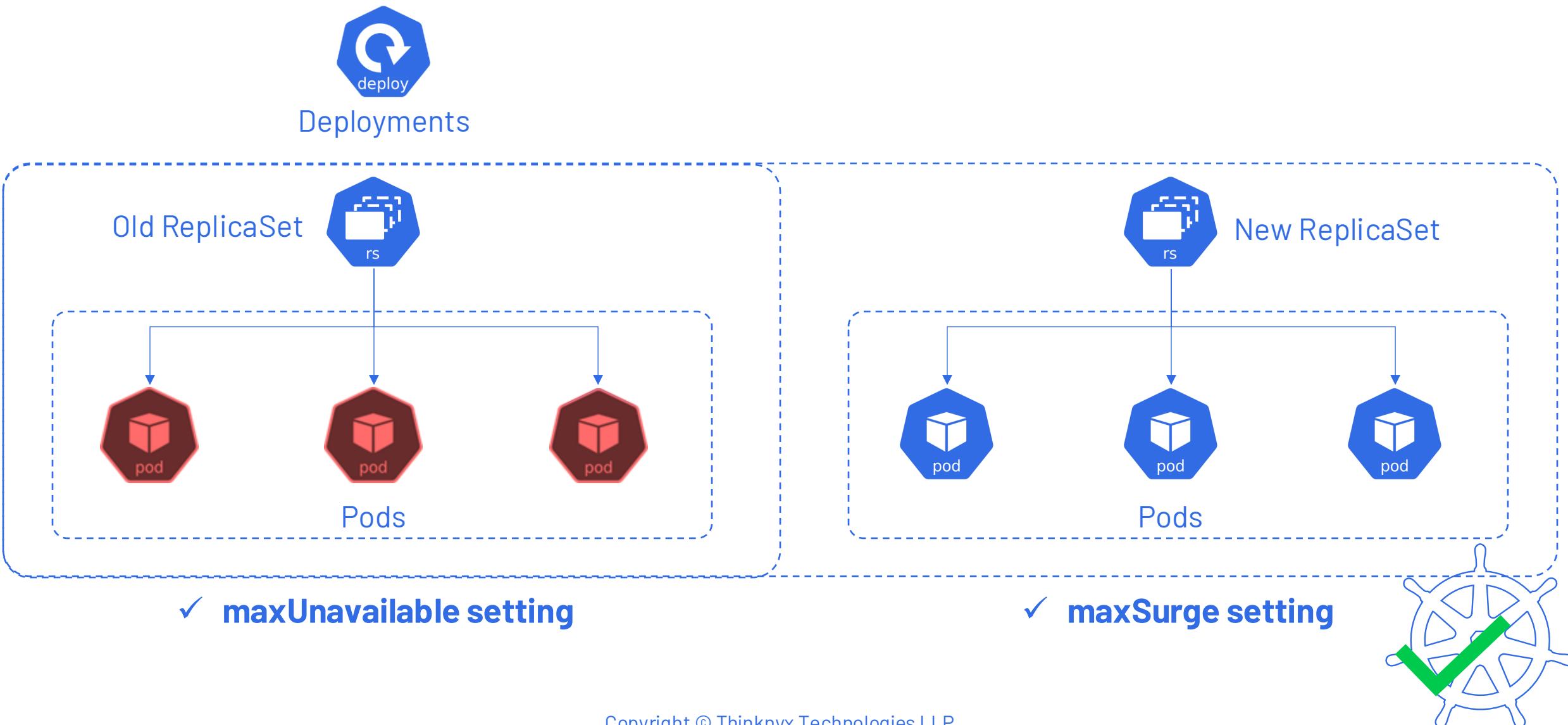
- ✓ Imperative
- ✓ Declarative



Rolling Update Strategy



Rolling Update Strategy



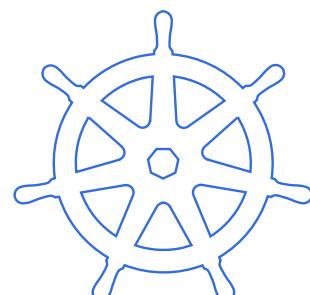
Rolling Update Strategy

Benefits

- ✓ Minimize service disruption

Drawback

- ✓ Gradual Update Process



Choosing the Right Strategy

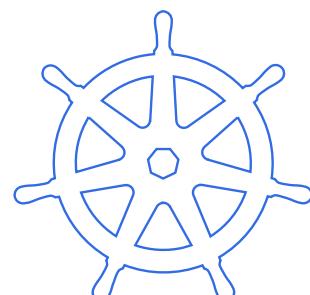
Choosing the Right Strategy

Recreate Strategy

Suitable for applications that can tolerate downtime

Rolling Update Strategy

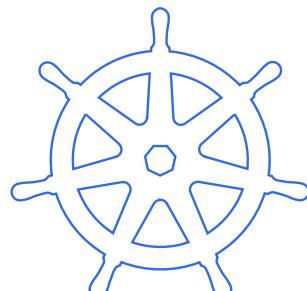
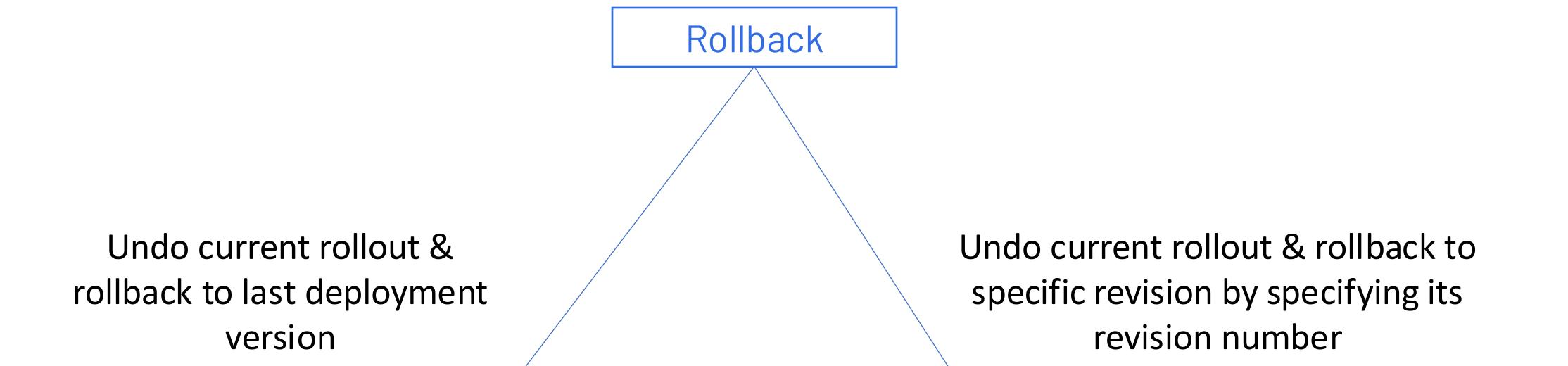
Ideal for applications that need continuous availability during updates



Rollback

Rollback

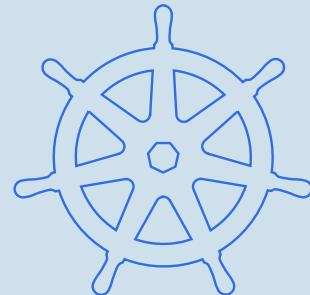
- Process of reverting application deployment to previous version





Demo

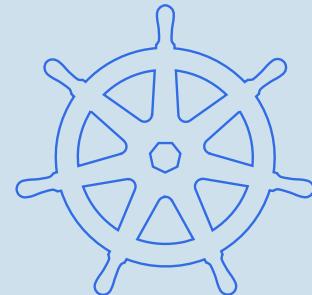
Rolling update Strategy
& Rollback





Demo

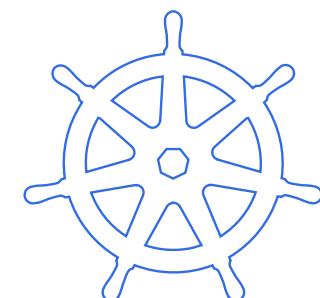
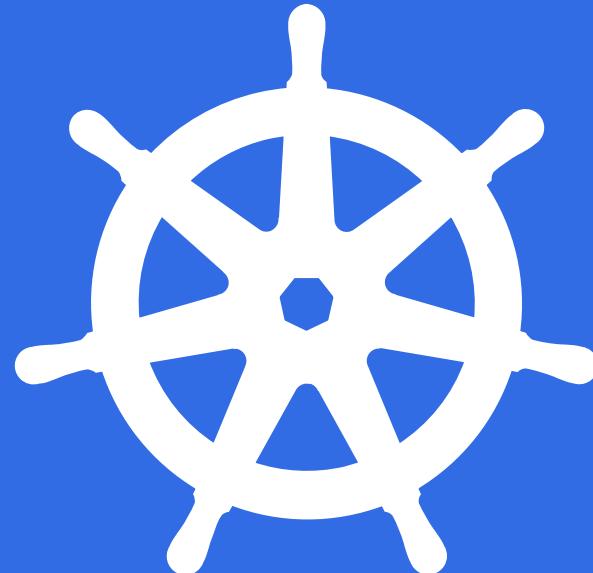
Recreate
Strategy



Summary

Summary

- ✓ Deployment Strategies
- ✓ Rollbacks

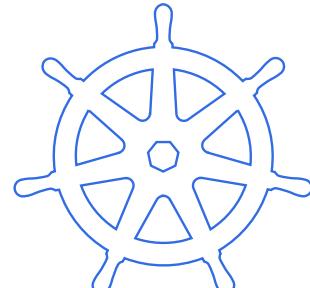


Section: 9

Section Overview

Kubernetes Cluster Maintenance

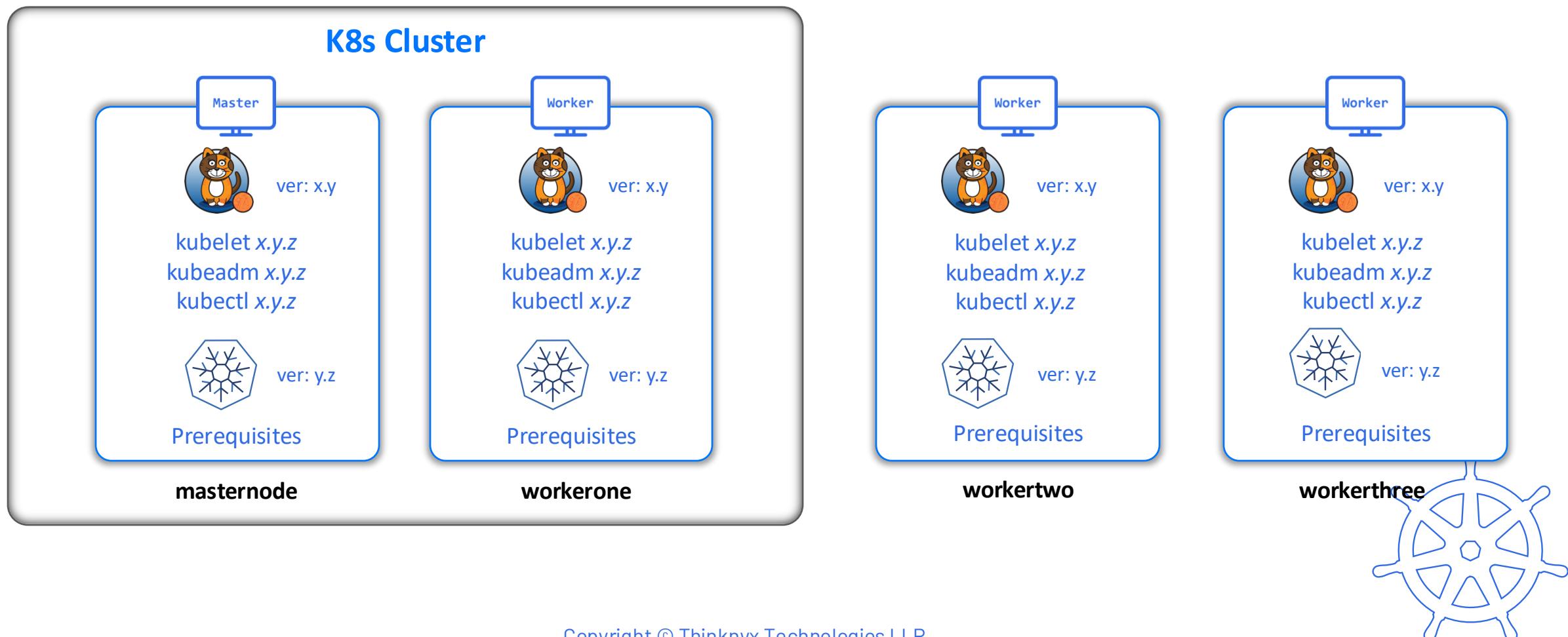
- ↳ **Cluster Scale Out (adding a new node)**
- ↳ **Cordon & Uncordon in K8s**
- ↳ **Draining in K8s**
- ↳ **etcd Backup & Restoration**
- ↳ **Cluster Scale In (removing a node)**



Scaling out a Cluster

Scaling out a Cluster

Step 1: Provision New Worker Nodes



Scaling out a Cluster

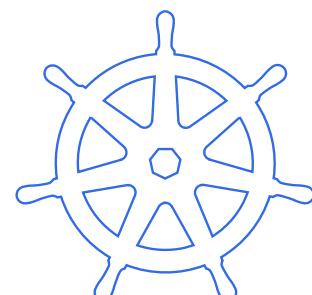
Step 2: Generate a New Join Token

- K8s token expires after 24 hours
- To add a new worker node, we need to generate a new token

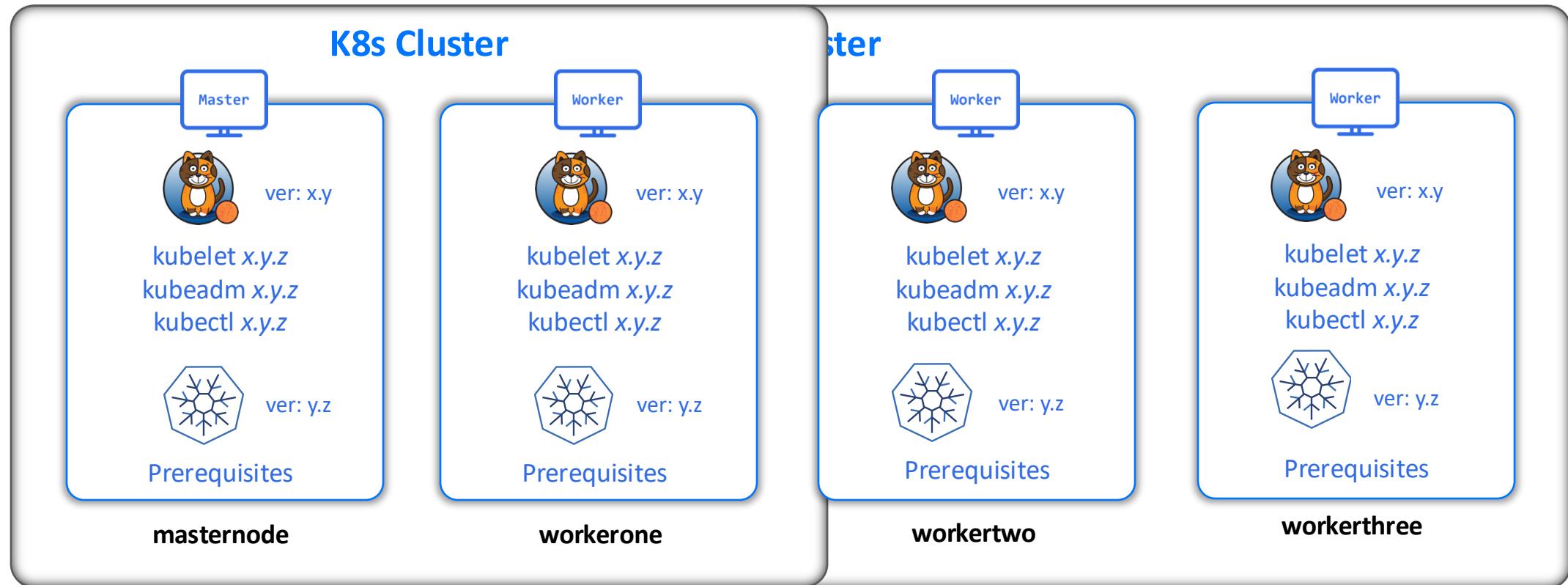


```
kubeadm token create --print-join-command
```

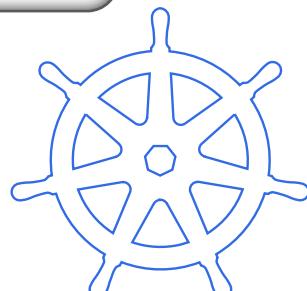
```
kubeadm join <master-node-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```



Scaling out a Cluster



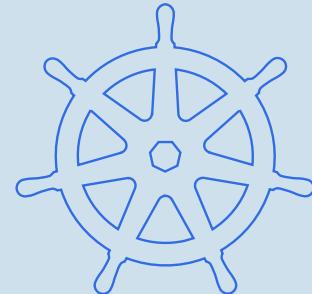
`kubectl get nodes`





Demo

Scaling out K8s
Cluster



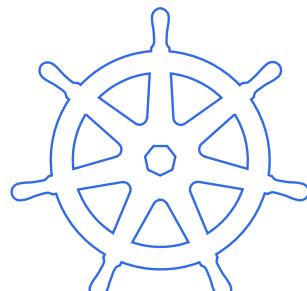
Cordon & Uncordon

Cordon & Uncordon

- Managing node availability in a Kubernetes cluster is key for planned maintenance or upgrades

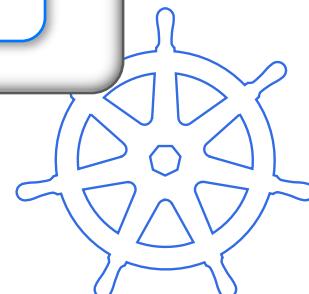
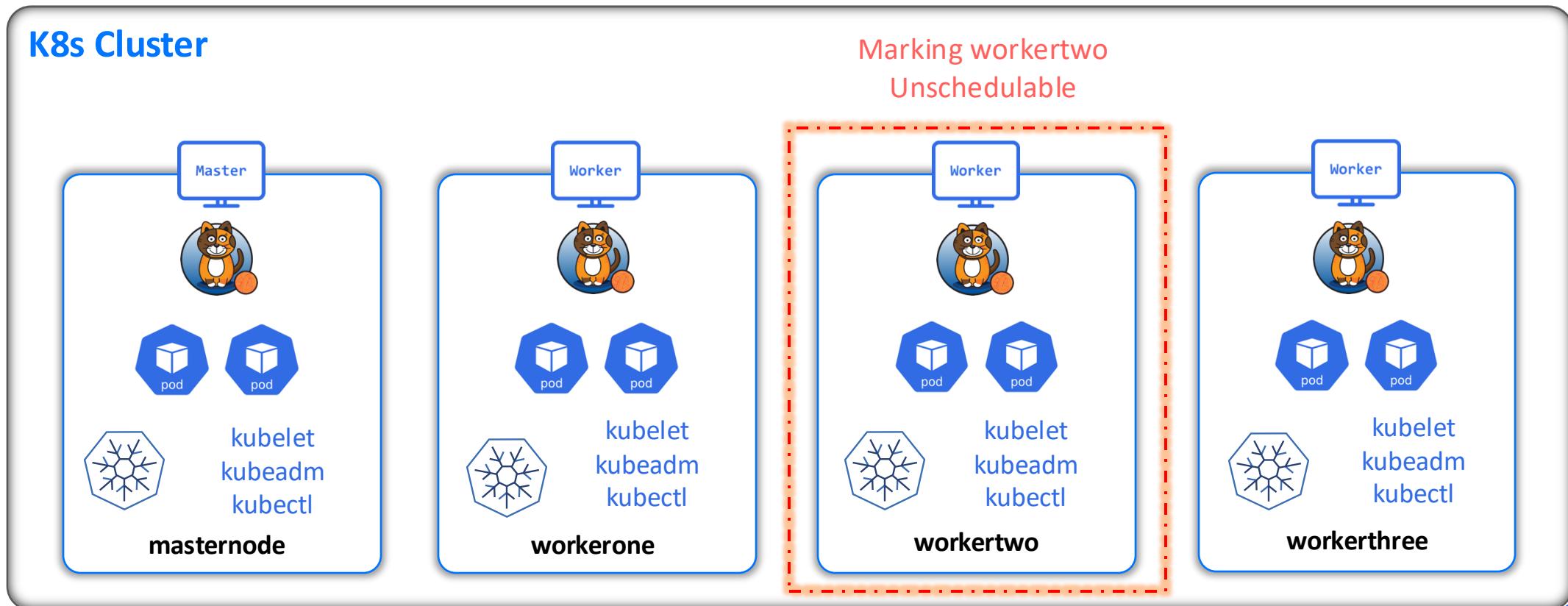
Cordon

Uncordon



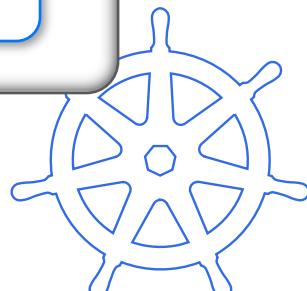
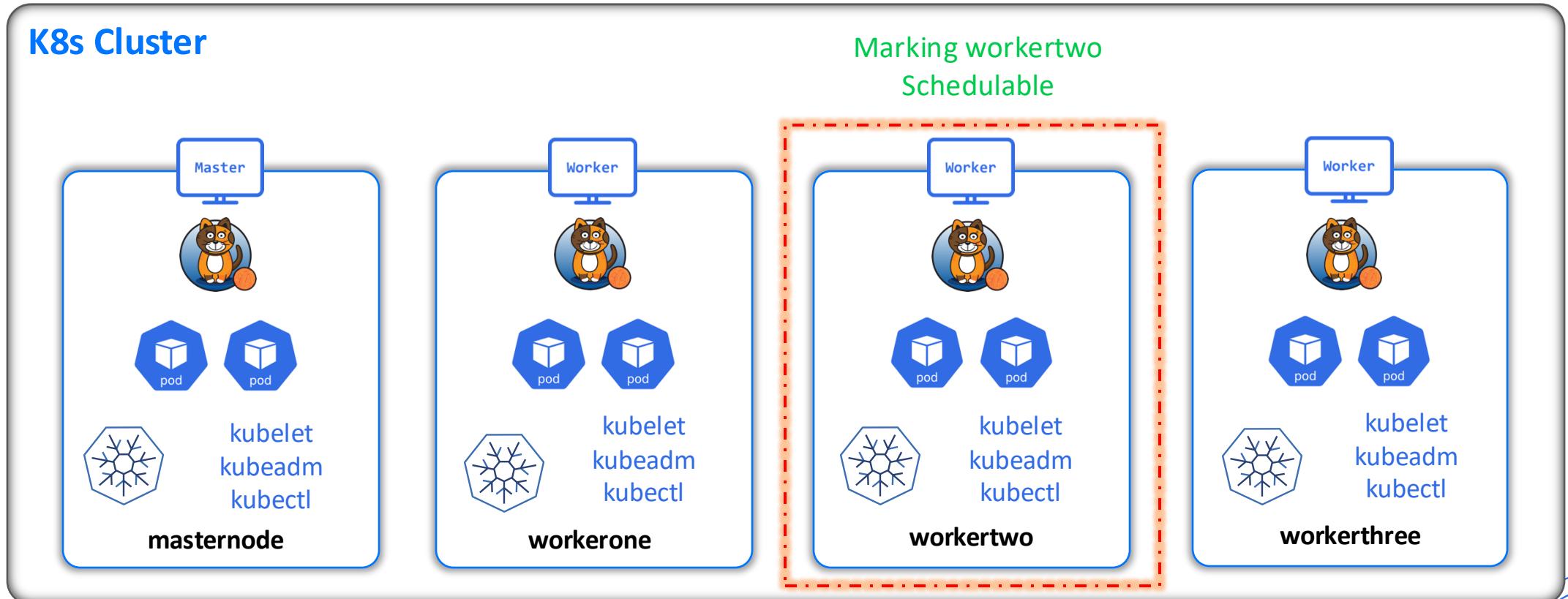
Cordon

- Cordon a Kubernetes node stops new pods from being scheduled, allowing maintenance without disrupting current workloads



Uncordon

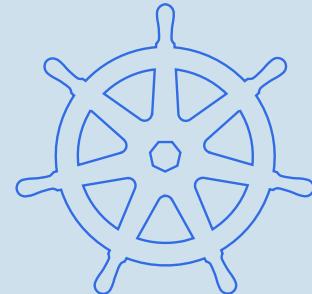
- After maintenance, uncordoning the node makes it schedulable again, allowing new pods to be scheduled





Demo

Cordon &
Uncordon



Draining in Kubernetes

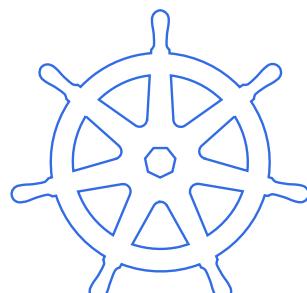
Draining in Kubernetes

- Cordon stops new pods from being scheduled but it doesn't evict existing ones
- Draining offers a mechanism to gracefully evict existing pods for smooth maintenance or upgrades

```
kubectl drain <node-name>
```

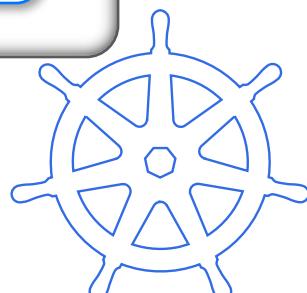
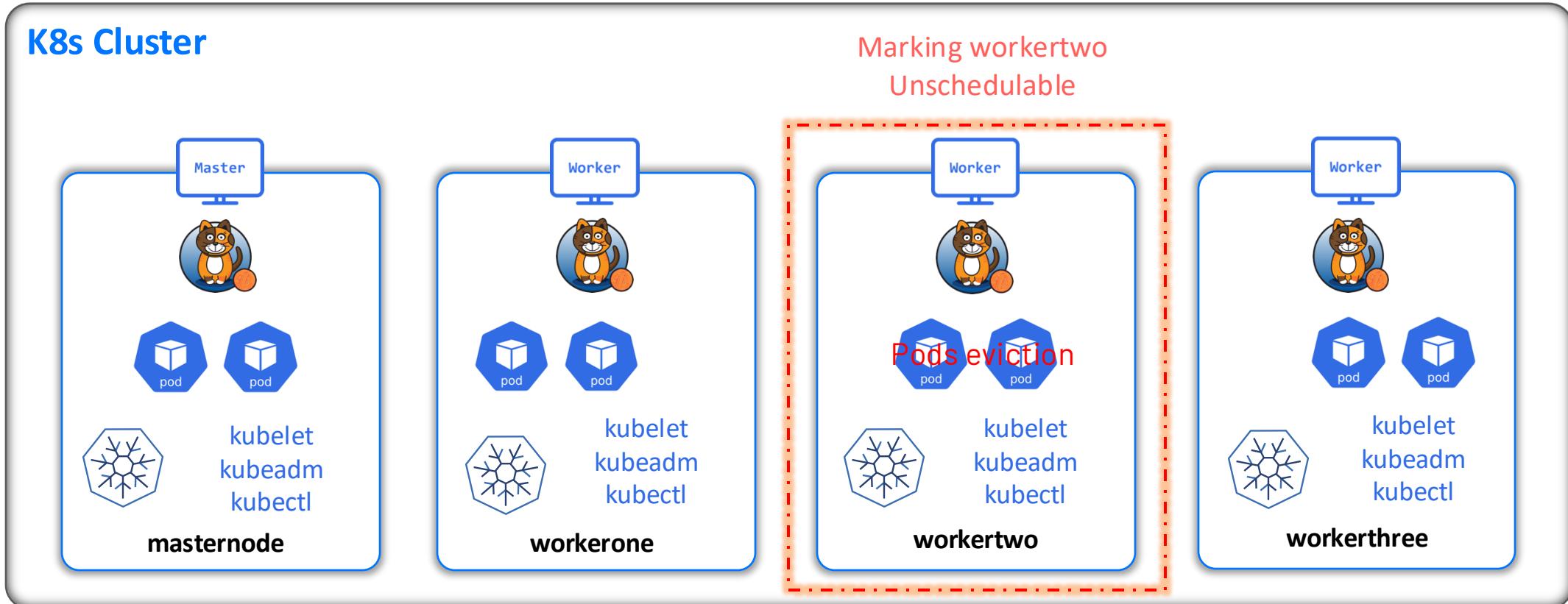
- **kubectl drain** cannot evict pods managed by DaemonSets

```
--ignore-daemonsets=true
```



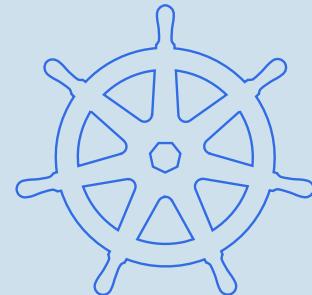
Draining in Kubernetes

```
kubectl drain workertwo --ignore-daemonsets=true
```



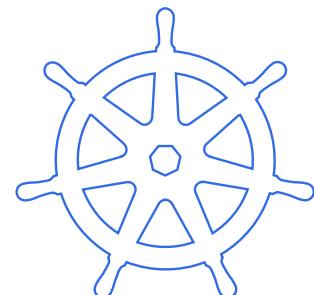


Demo | Draining in K8s



etcd Backup & Restoration

etcd Backup & Restoration

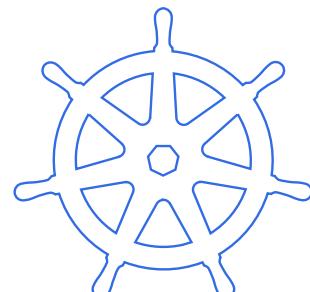


etcd Backup & Restoration



Importance of etcd Backups

- ✓ etcd stores all critical Kubernetes data, including deployments, services, and secrets
- ✓ Regular etcd backups are essential to recover from data corruption, deletions, or failures
- ✓ Backups help restore the cluster quickly, reducing downtime and data loss

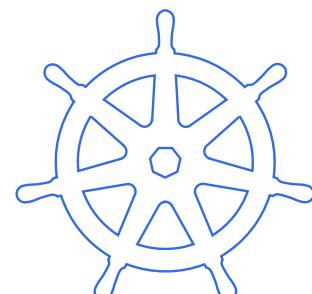


etcd Backup & Restoration



Steps to perform etcd backup and restoration

- ✓ You only need to install the **etcdctl** binary on the master node, available at etcd.io



etcd Backup & Restoration

CA Certificate
Path

/etc/kubernetes/pki/etcd/ca.crt

Server
Certificate Path

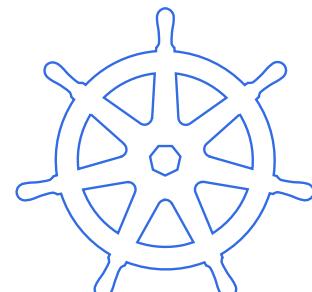
/etc/kubernetes/pki/etcd/server.crt

Server Key Path

/etc/kubernetes/pki/etcd/server.key

API Server
Endpoint

[https://\[127.0.0.1\]:2379](https://[127.0.0.1]:2379)



etcd Backup & Restoration



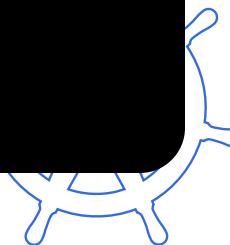
```
root@thinknyx:~$ ETCDCCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \  
--cacert=/etc/kubernetes/pki/etcd/ca.crt \  
--cert=/etc/kubernetes/pki/etcd/server.crt \  
--key=/etc/kubernetes/pki/etcd/server.key \  
snapshot save etcdbackup.db
```

--help

```
root@thinknyx:~$
```

```
root@thinknyx:~$ ETCDCCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \  
--cacert=/etc/kubernetes/pki/etcd/ca.crt \  
--cert=/etc/kubernetes/pki/etcd/server.crt \  
--key=/etc/kubernetes/pki/etcd/server.key \  
snapshot status etcdbackup.db
```

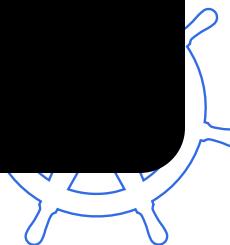
```
root@thinknyx:~$
```



etcd Backup & Restoration



```
root@thinknyx:~$ mkdir /var/lib/etcd-from-backup
root@thinknyx:~$
root@thinknyx:~$ ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
--data-dir /var/lib/etcd-from-backup snapshot restore /root/etcdbackup.db
```



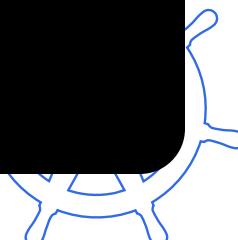
etcd Backup & Restoration



```
root@thinknyx:~$ vi /etc/kubernetes/manifests/etcd.yaml
```

```
volumeMounts:  
- mountPath: /var/lib/etcd  
  name: etcd-data  
  ...  
volumes:  
- hostPath:  
  path:  
  type: DirectoryOrCreate  
  name: etcd-data
```

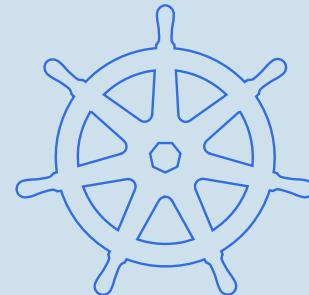
```
root@thinknyx:~$
```





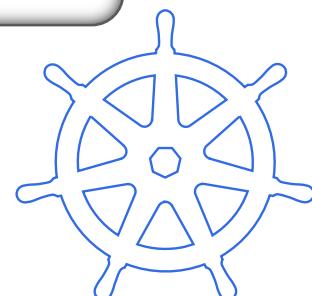
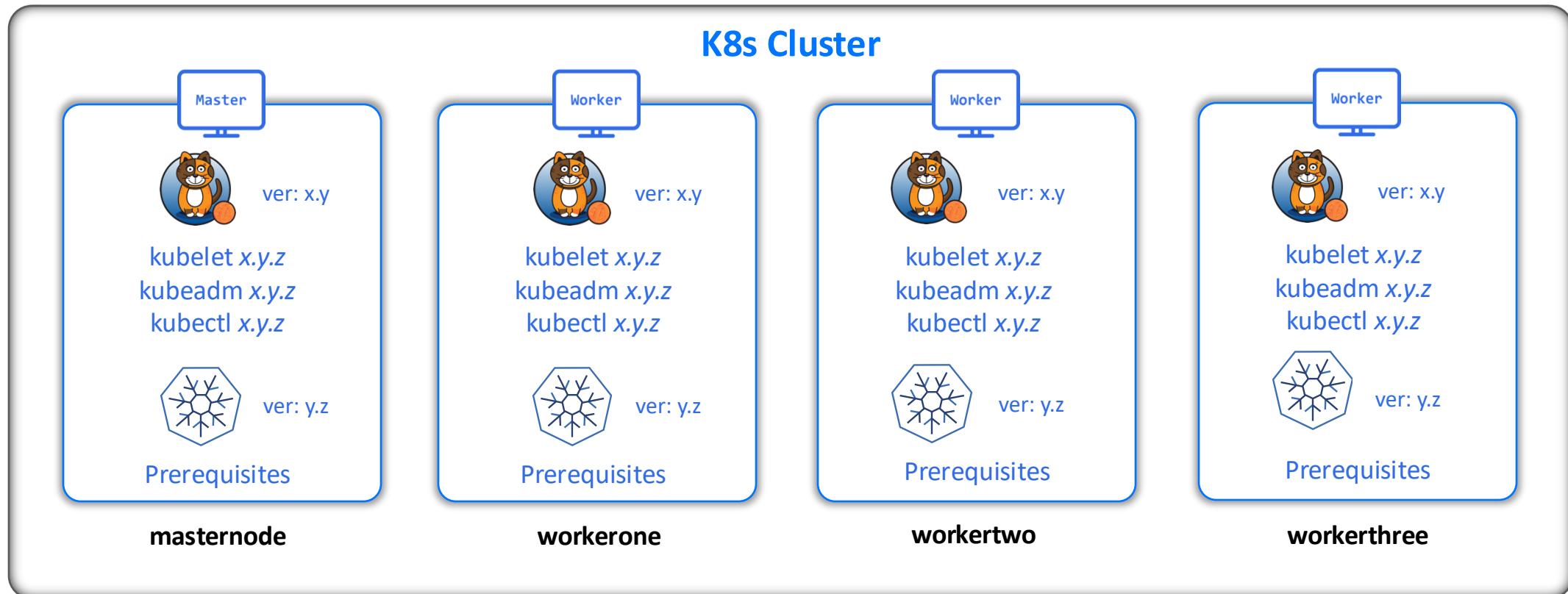
Demo

etcd Backup &
Restoration

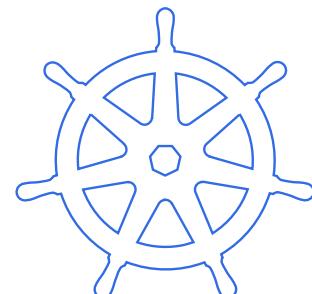
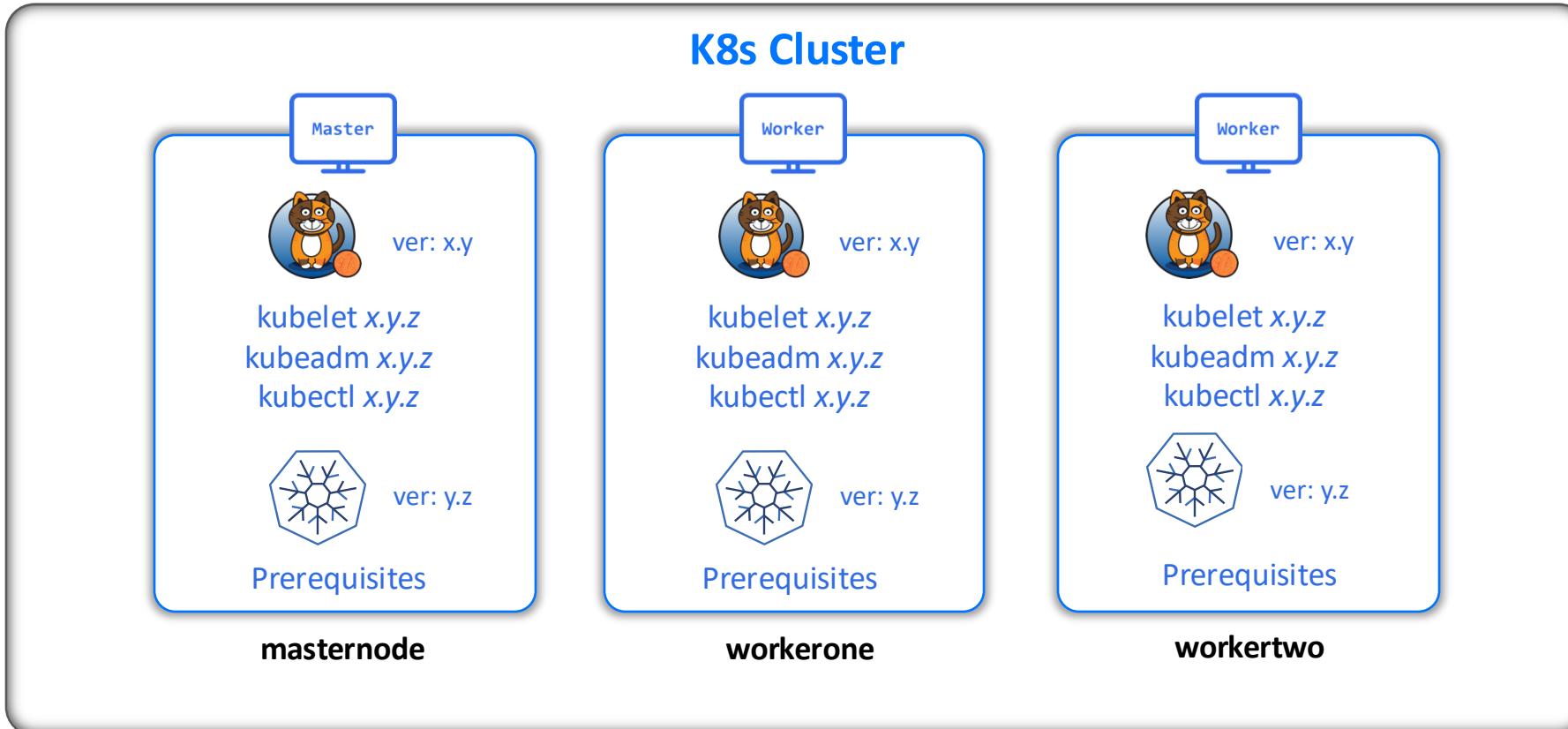


Scaling in a Cluster

Scaling in a Cluster



Scaling in a Cluster



Scaling in a Cluster

Step 1: Drain a node (run below command from your master node)

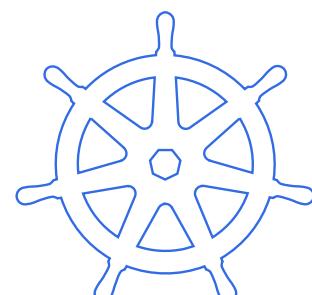
```
kubectl drain <node-name> --ignore-daemonsets=true
```

Step 2: Reset a node (run below command from your worker node)

```
kubeadm reset
```

Step 3: Delete a node (run below command from your master node)

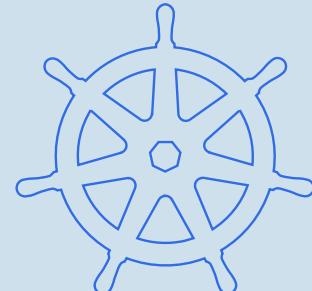
```
kubectl delete node <nodename>
```





Demo

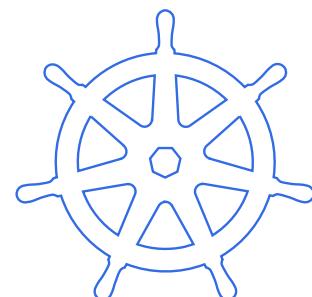
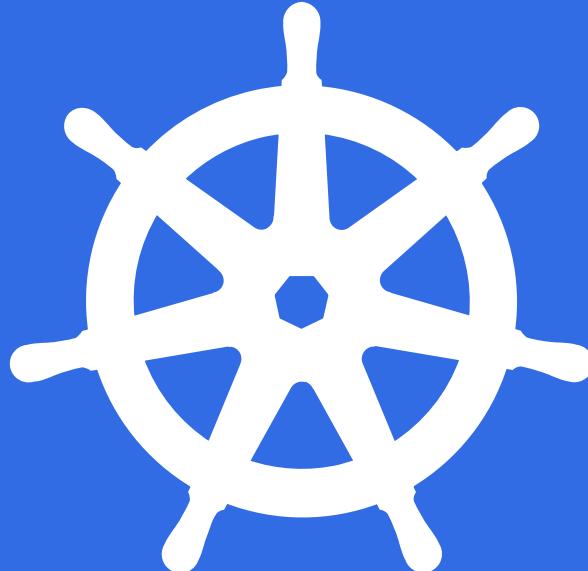
Scaling in K8s Cluster



Summary

Summary

- ✓ Cluster Scale Out (handle increased workloads)
- ✓ Cordon & Uncordon
- ✓ Draining Nodes
- ✓ etcd Backup & Restoration
- ✓ Cluster Scale In (optimize resource usage)

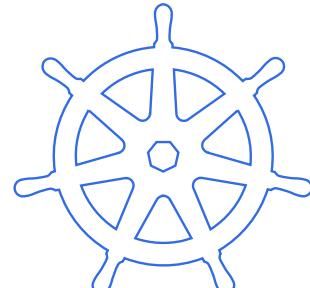


Section: 10

Section Overview

Useful Kubernetes Objects

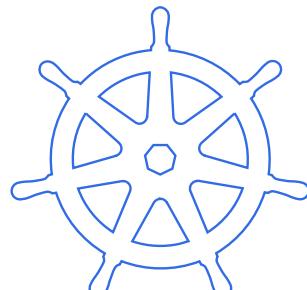
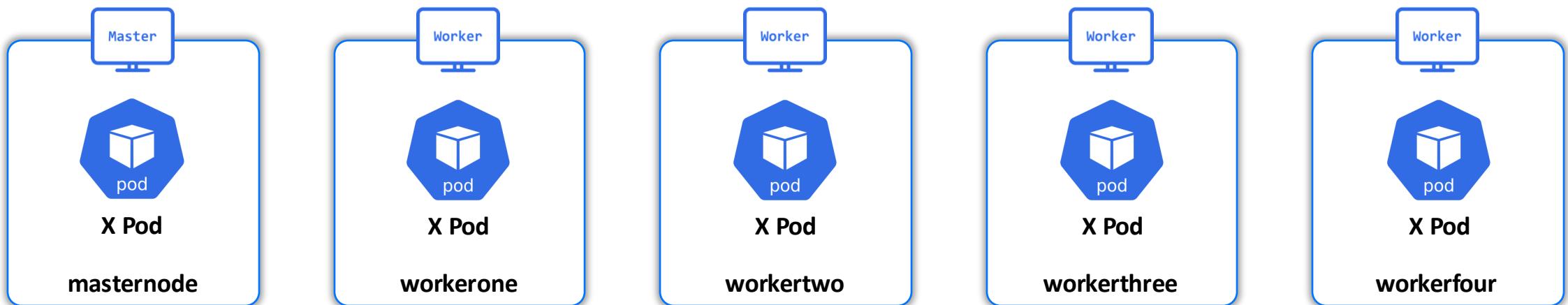
- ↳ **DaemonSet**
- ↳ **Jobs & CronJob**
- ↳ **Static Pods**
- ↳ **ConfigMaps & Secrets**



DaemonSet

DaemonSet

- DaemonSet in Kubernetes ensures a specified pod runs on every node in the cluster, making it ideal for system-level tasks like log collection, monitoring, or networking



DaemonSet

- DaemonSet in Kubernetes ensures a specified pod runs on every node in the cluster, making it ideal for system-level tasks like log collection, monitoring, or networking

Key Features

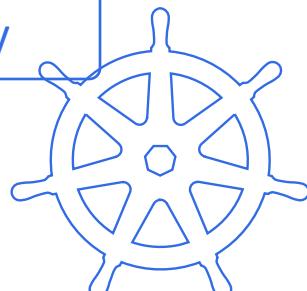
Uniform Deployment

Automatic Updates

Node Affinity>Selectors

Scalability

Resource Efficiency



DaemonSet



Logging



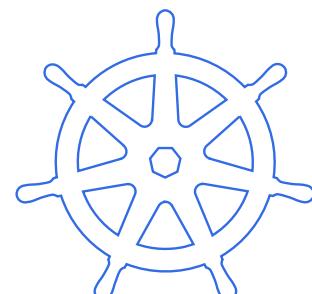
Monitoring



Networking

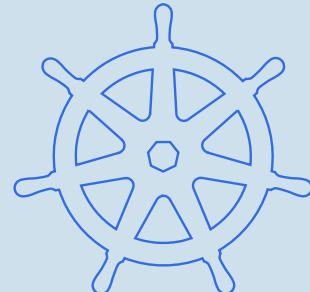


Security





Demo | DaemonSet

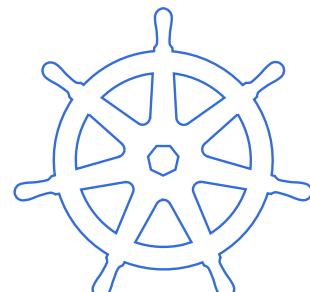
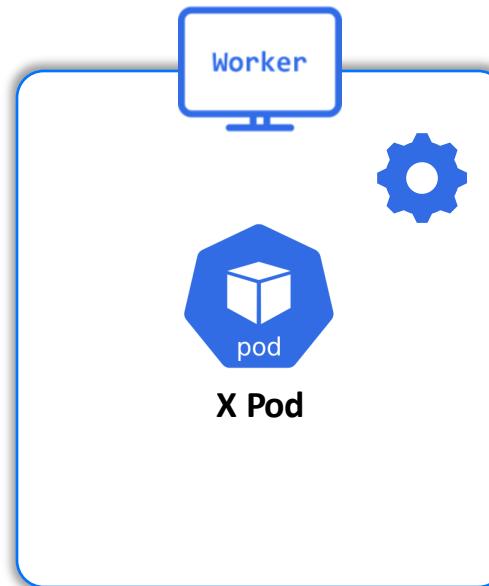


Jobs

Jobs

- In Kubernetes, Jobs manage batch processing tasks by ensuring a set number of pods complete their work, ideal for one-time or finite tasks
- Typically used for tasks like data processing, backups, or sending emails

Remove “X Pod” after the task is completed



Jobs

- In Kubernetes, Jobs manage batch processing tasks by ensuring a set number of pods complete their work, ideal for one-time or finite tasks
- Typically used for tasks like data processing, backups, or sending emails

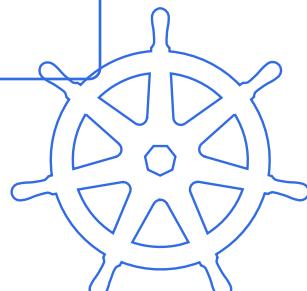
Key Features

Finite
Execution

Pod
Management

Parallelism

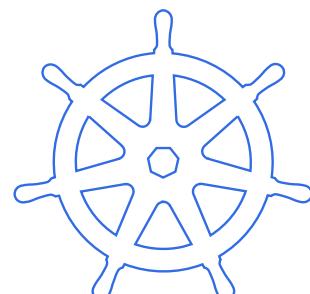
Completion
Tracking



Jobs

Types of Jobs

- ✓ Simple Jobs
- ✓ Parallel Jobs with a Fixed Completion Count
- ✓ Parallel Jobs with a Work Queue (Non-Parallel Jobs)



Jobs

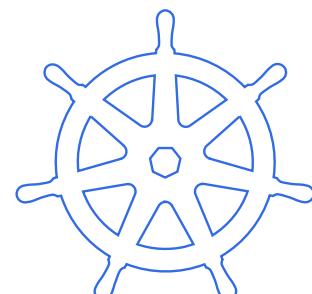
Use Cases

Data Processing

Database
Migrations

Scheduled
Maintenance

Event-Driven
Tasks



Jobs

Resource
Management

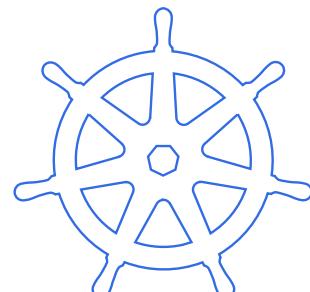
Job Cleanup

Best
Practices

Retries &
Backoff Limit

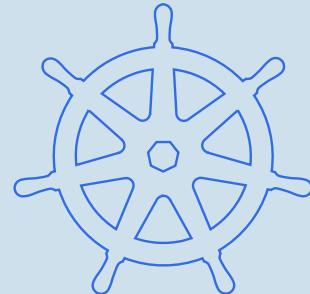
Idempotency

Monitoring and
Logging





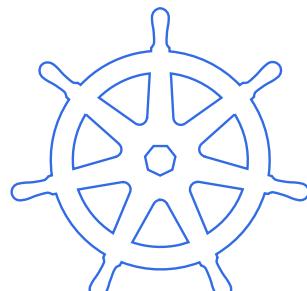
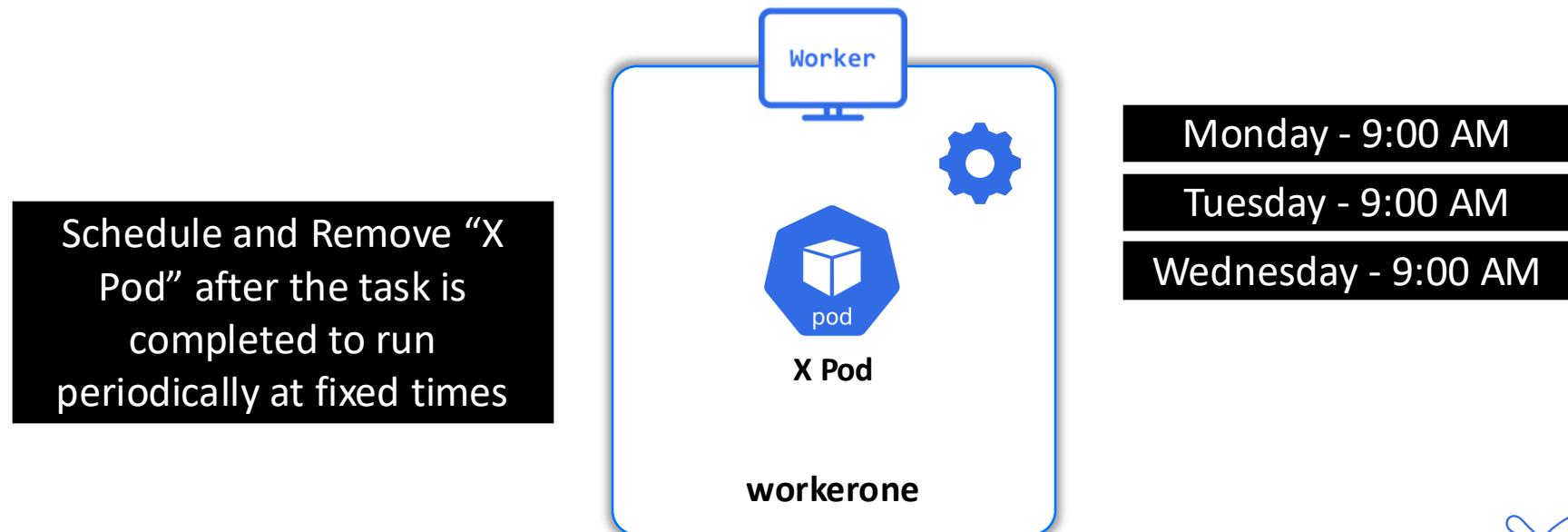
Demo | Jobs



CronJob

CronJob

- CronJob in Kubernetes schedules recurring tasks at specified intervals, similar to Unix cron, for regular operations like backups and report generation
- Ideal for regularly scheduled tasks like backups, report generation, or periodic data processing



CronJob

- CronJob in Kubernetes schedules recurring tasks at specified intervals, similar to Unix cron, for regular operations like backups and report generation
- Ideal for regularly scheduled tasks like backups, report generation, or periodic data processing

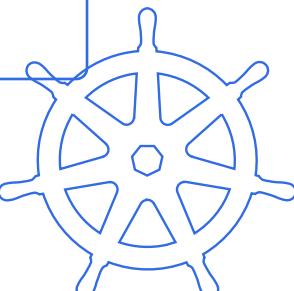
Key Features

Scheduled
Execution

Job
Management

Time-Based
Triggers

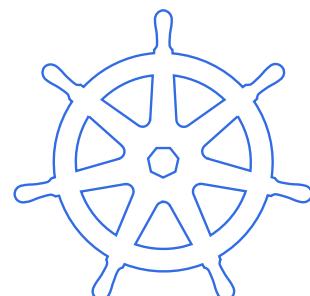
Concurrency
Policy



CronJob

Types of CronJob

- ✓ Simple CronJobs
- ✓ Complex Schedules
- ✓ Timezone Awareness



CronJob

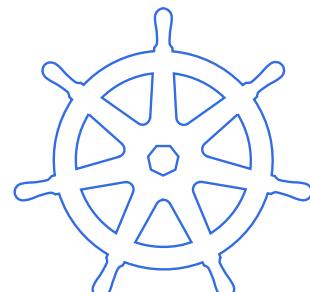
Use Cases

Regular Backups

System
Maintenance

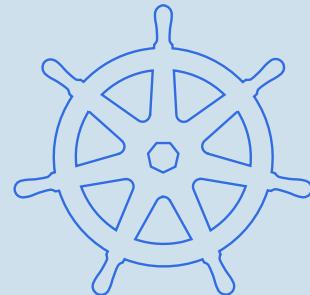
Report
Generation

Periodic Data
Processing





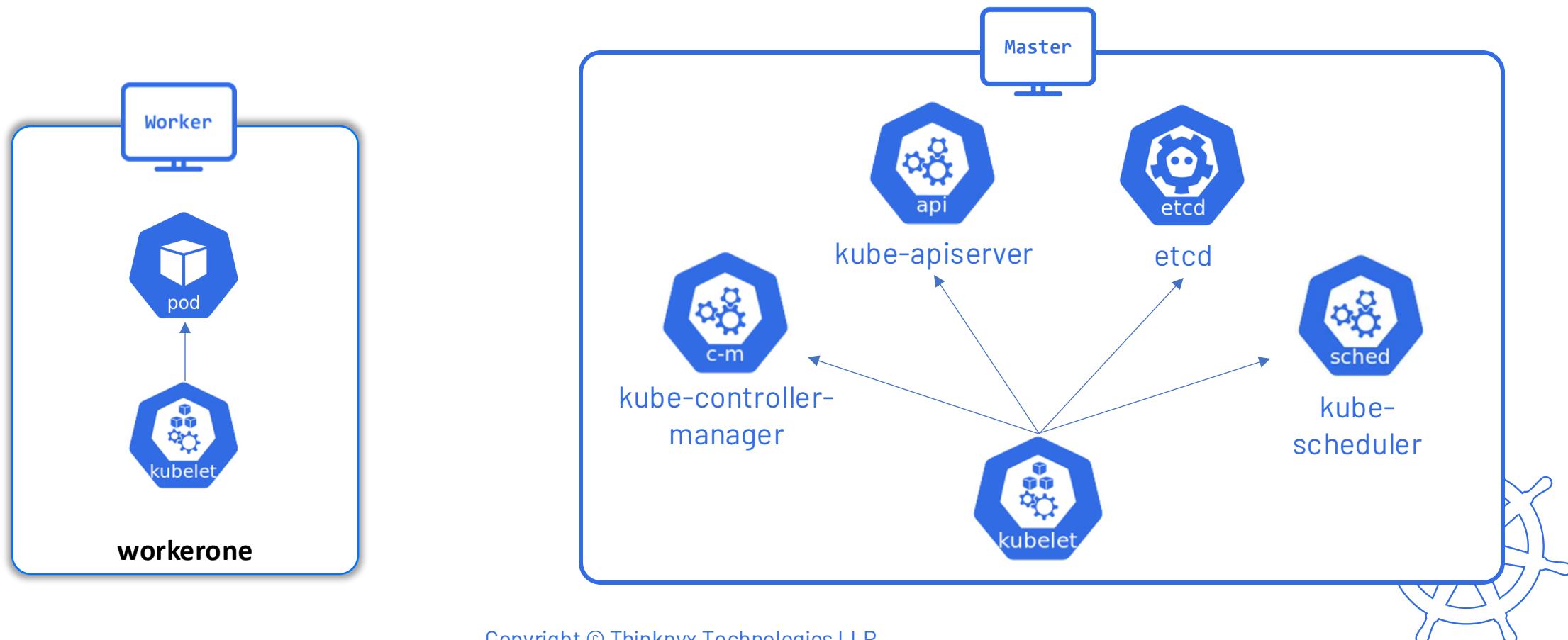
Demo | CronJob



Static Pods

Static Pods

- Static Pods are managed directly by the kubelet on individual nodes
- Provide a way to run containers on nodes without relying on the Kubernetes control plane for management



Static Pods

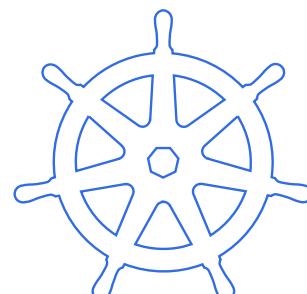
- Static Pods are managed directly by the kubelet on individual nodes
- Provide a way to run containers on nodes without relying on the Kubernetes control plane for management

Key Features

Node-Specific
Management

Configuration
Through Files

Self-Healing

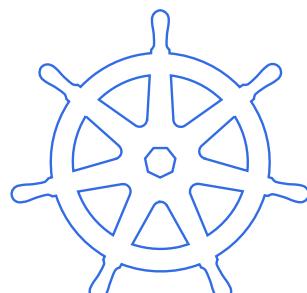


Static Pods

Use Case

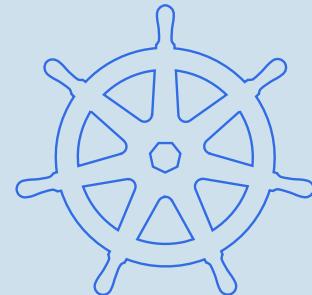
Critical Cluster Components

Static Pods ensure critical applications and components are always running on specific nodes, providing a stable environment for essential cluster operations





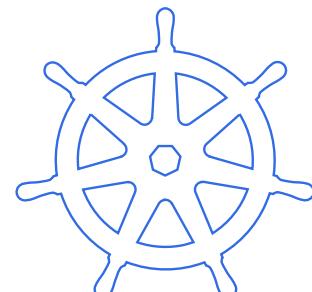
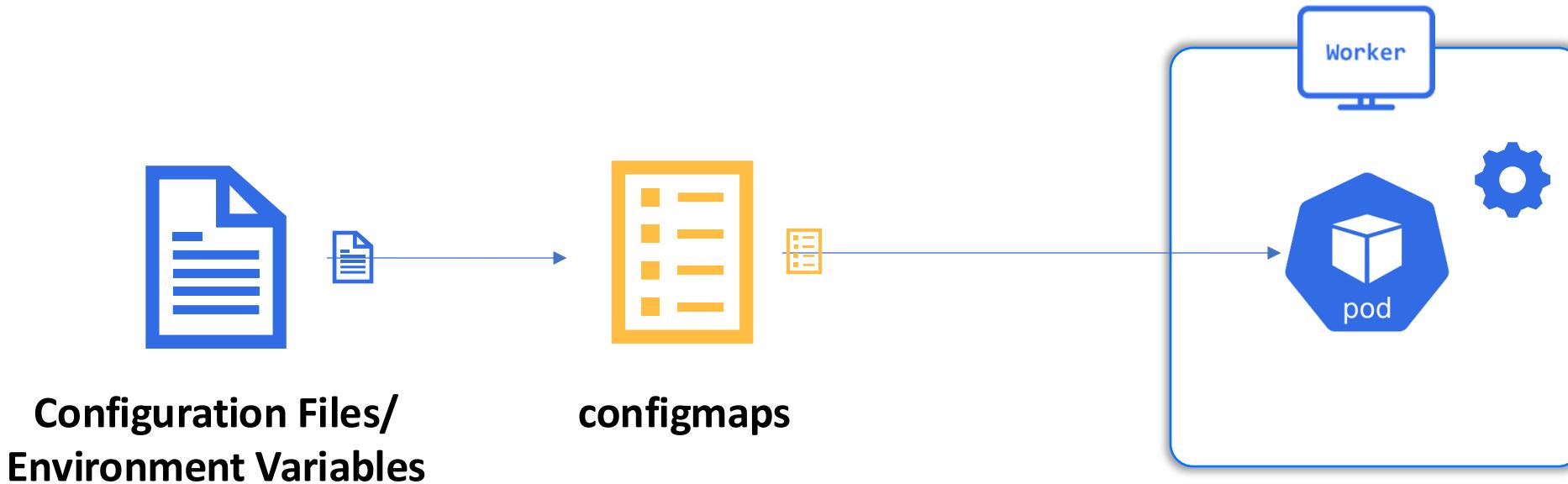
Demo | Static Pods



ConfigMaps

ConfigMaps

- ConfigMaps in Kubernetes store configuration data as key-value pairs, allowing you to manage and update configurations independently from container images



ConfigMaps

- ConfigMaps in Kubernetes store configuration data as key-value pairs, allowing you to manage and update configurations independently from container images

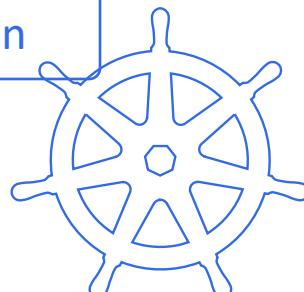
Key Features

Centralized
Management

Dynamic
Updates

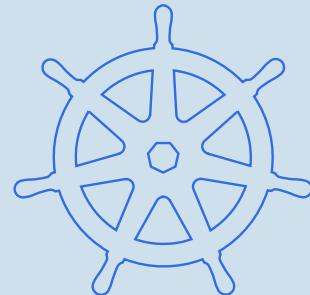
Versatile Data
Sources

Seamless
Integration





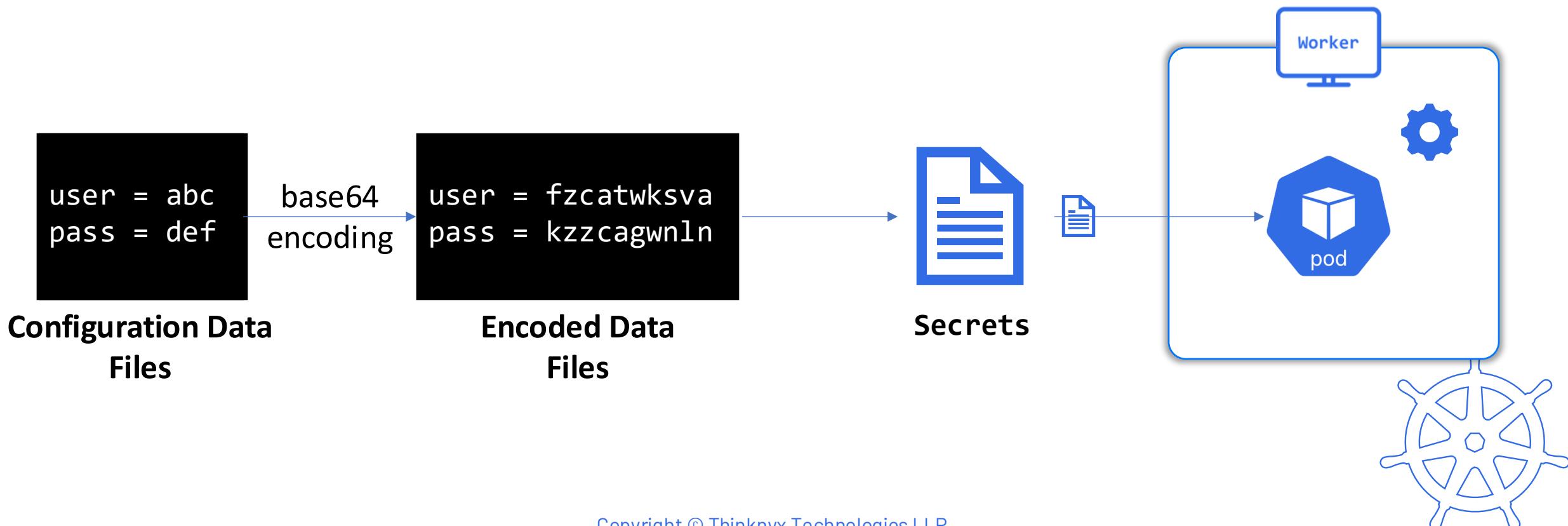
Demo | ConfigMaps



Secrets

Secrets

- Secrets in Kubernetes securely stores sensitive information like passwords and tokens, offering better security than including them directly in pod files

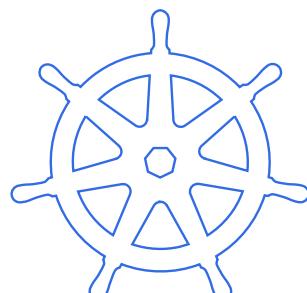


Secrets

- Secrets in Kubernetes securely stores sensitive information like passwords and tokens, offering better security than including them directly in pod files

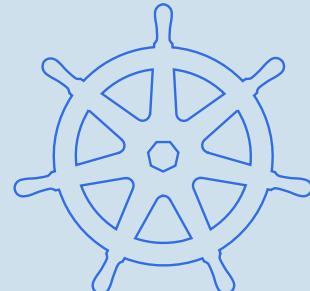
Types of Secrets

- ✓ Opaque Secrets
- ✓ Service Account Token Secrets
- ✓ Docker Config Secrets
- ✓ TLS Secrets





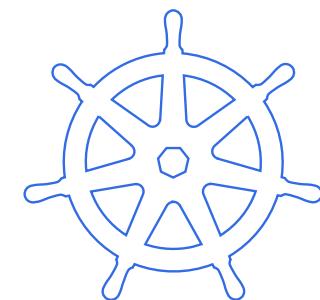
Demo | Secrets



Summary

Summary

- ✓ DaemonSet
- ✓ Jobs & CronJob
- ✓ Static Pods
- ✓ ConfigMaps & Secrets

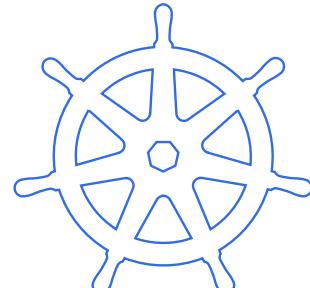


Section: 11

Section Overview

Advanced Pod Tasks

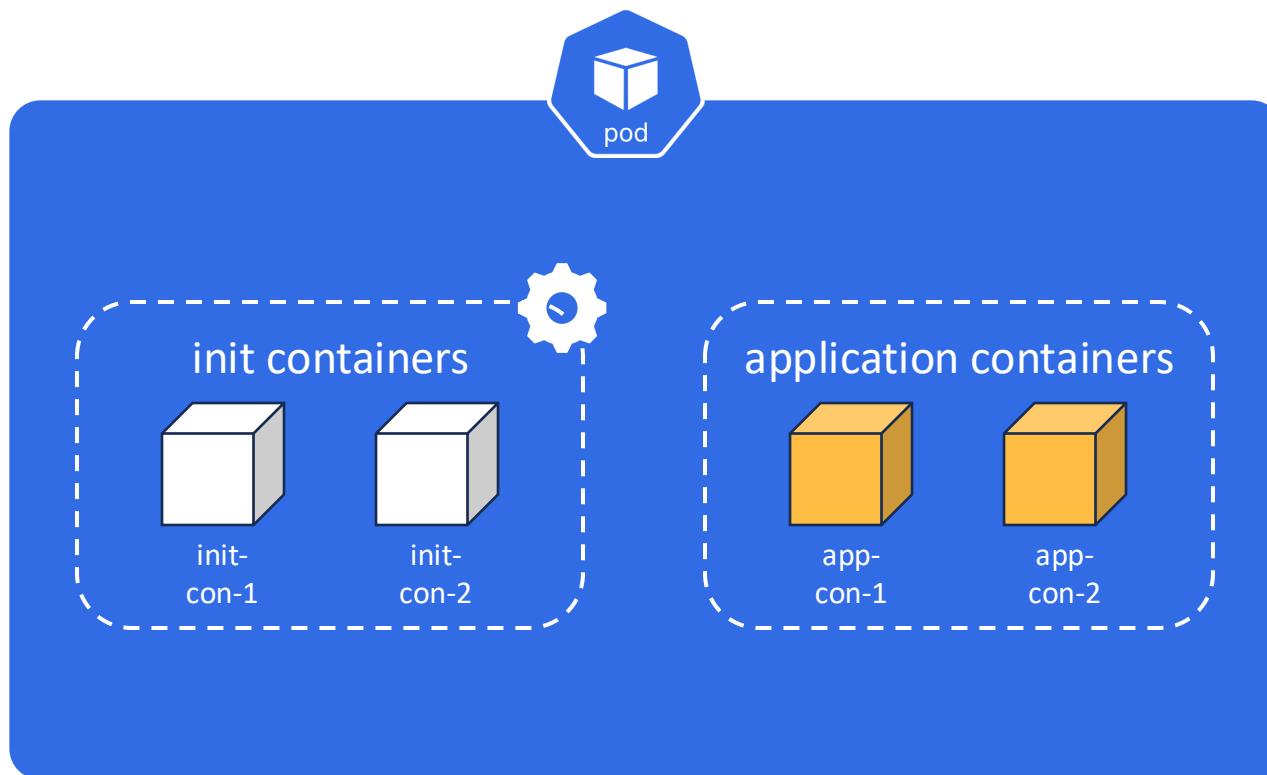
- ↳ **Init containers**
- ↳ **Multicontainer pods**
- ↳ **Resource Limits and Requests**
- ↳ **Liveness, Readiness, & Startup probes**



Init Containers

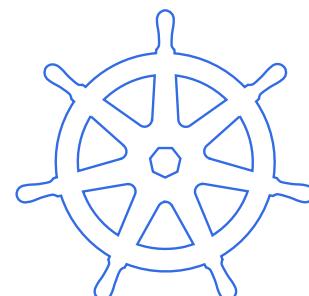
Init Containers

- Init containers are specialized containers that run before application containers in a Pod to perform setup scripts or utilities not included in the application image

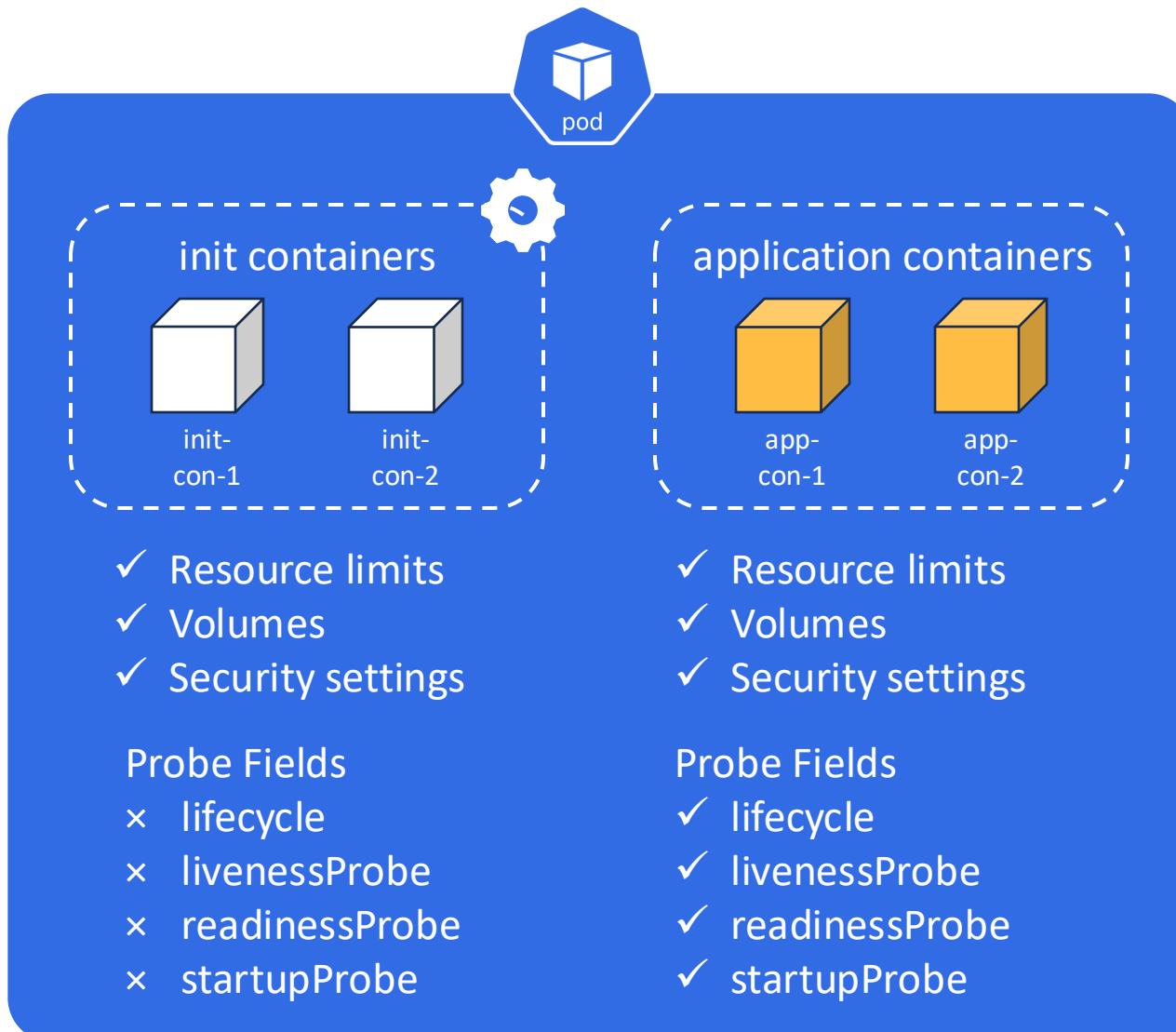


Key characteristics:

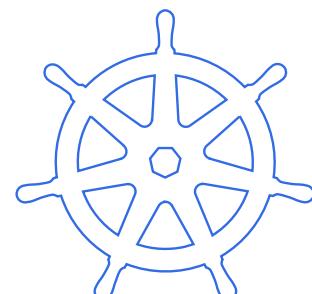
- ✓ Run to Completion
- ✓ Sequential Execution
- ✓ Failure Handling



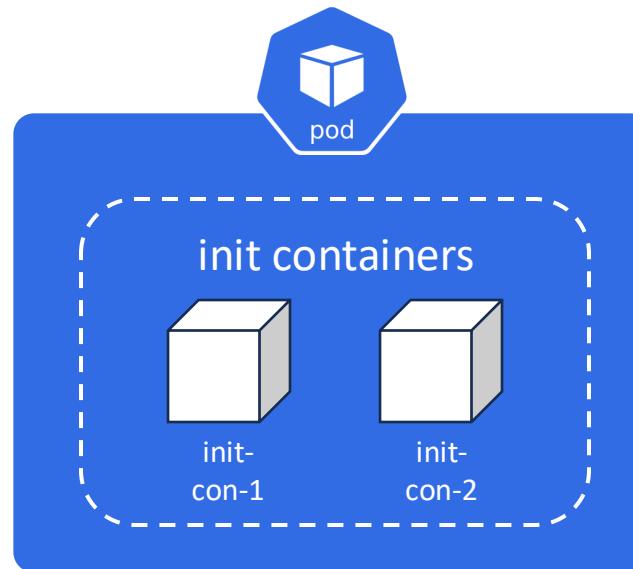
Init Containers



The kubelet runs multiple init containers sequentially, ensuring that the main application containers start only after all init containers have successfully completed

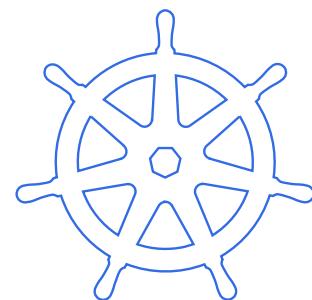


Init Containers



Separate
Images

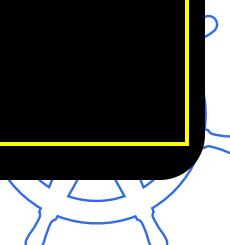
Startup
Control



Init Containers



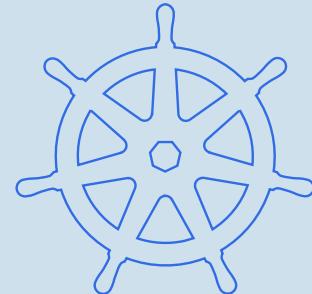
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    name: myapp
spec:
  containers:
    - name: myapp-container
      image: nginx
  initContainers:
    - name: inittest
      image: busybox:latest
      command: ['sh', '-c', "until nslookup testservice.default.svc.cluster.local; do echo waiting for testservice.default.svc.cluster.local; sleep 10; date; done"]
```





Demo

| Init Containers



Multicontainer Pod

Multicontainer Pod

- Multicontainer Pod is a useful way to create modular, reusable, and easy-to-maintain applications



Speed Application
Development

Why Use
Multicontainer
Pods?



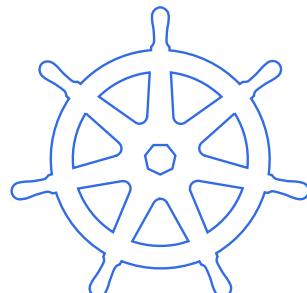
Separation of
Concerns



Codify Expert
Knowledge



Enable Agile
Teams

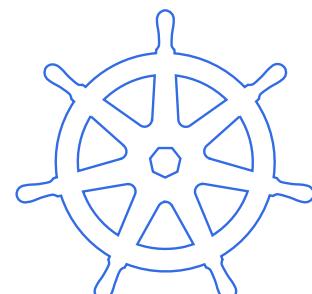
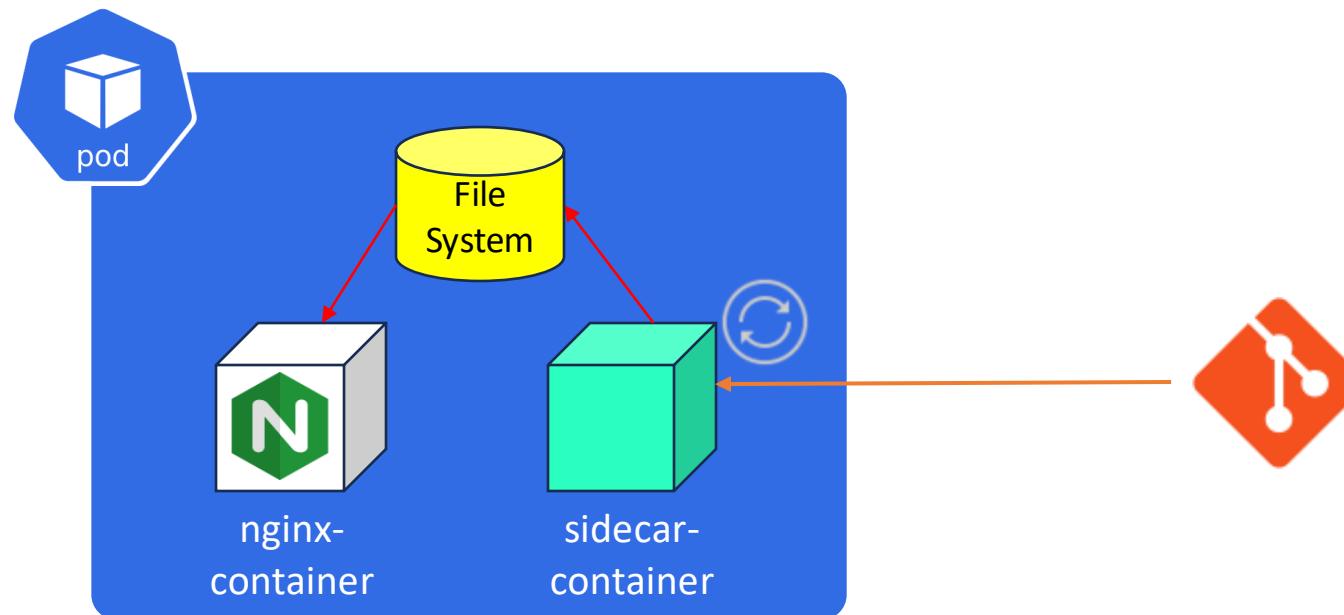


Composite Container Patterns

Composite Container Patterns

Sidecar Containers

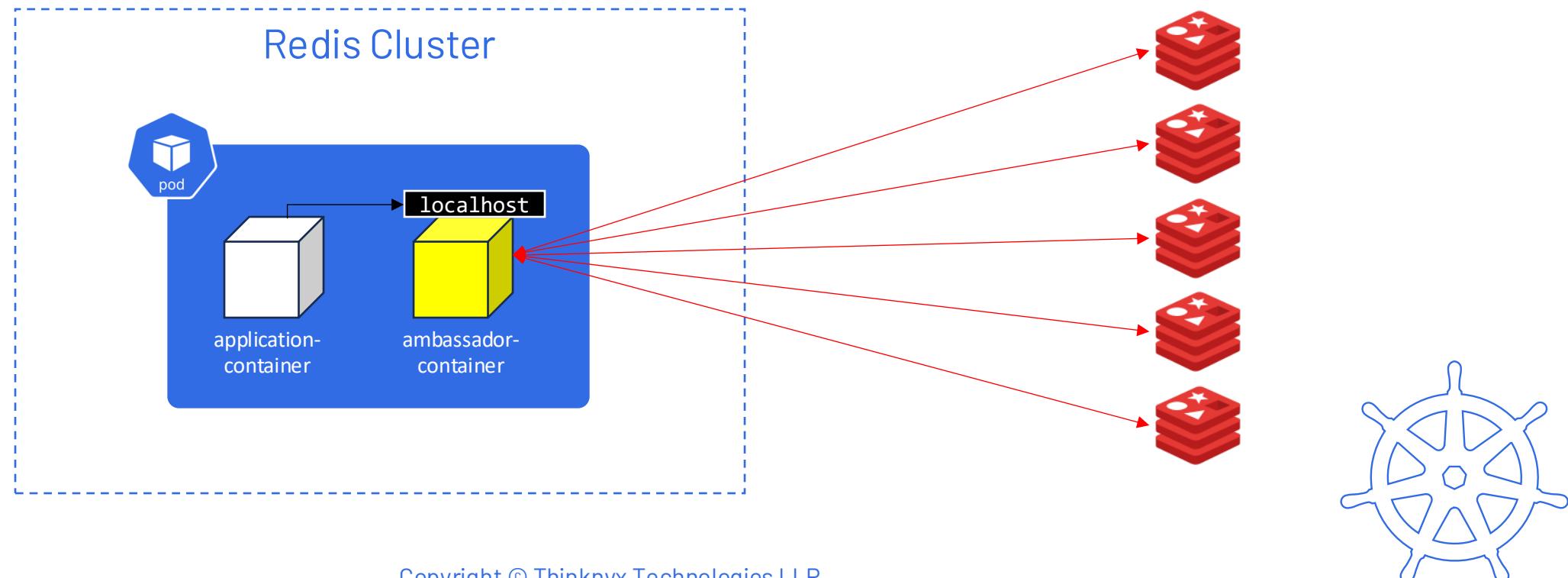
- Enhance the main container's capabilities
- Handle auxiliary functions separate from the main application logic
- This approach simplifies management and maintenance within a Pod



Composite Container Patterns

Ambassador Containers

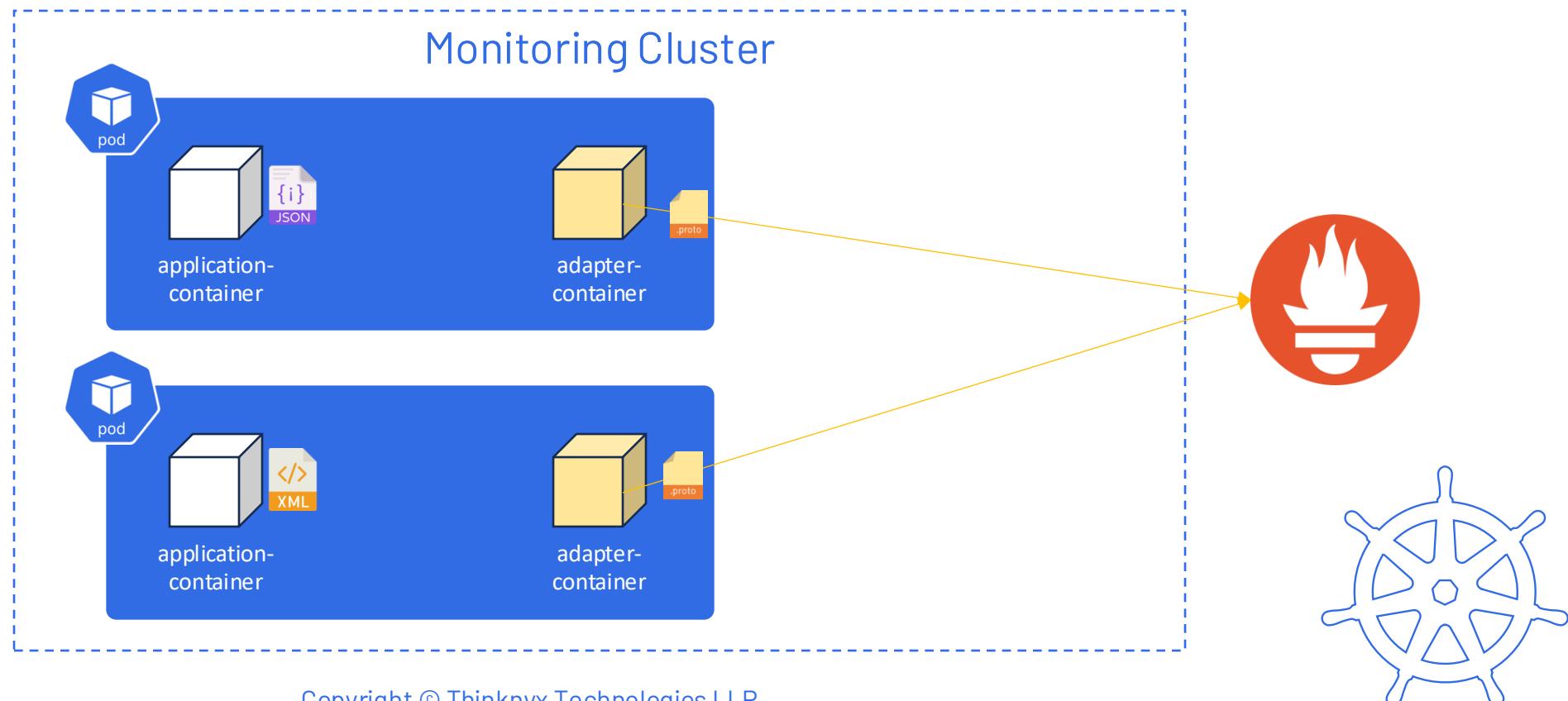
- Act as proxies for communication between the main application and external services
- Handle network connections, protocols, and routing
- This approach abstracts communication details, leading to a more focused application codebase



Composite Container Patterns

Adapter Containers

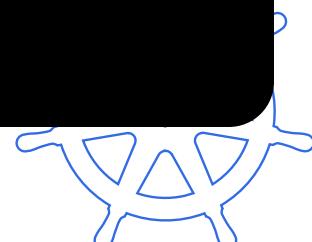
- Standardize outputs from various applications, ensuring they conform to a common format
- By translating and converting data, adapter containers help maintain consistency across diverse components



Multicontainer Pod



```
apiVersion: v1
kind: Pod
metadata:
  name: multicontainer
  labels:
    run: multicontainer
spec:
  containers:
    - image: nginx
      name: nginxcon
    - image: redis
      name: rediscon
    - image: memcached
      name: memcachedcon
```



Multicontainer Pod



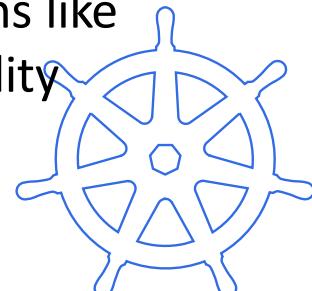
```
apiVersion: v1
kind: Pod
metadata:
  name: myapppod
  labels:
    name: myApp
spec:
  containers:
    - name: myappcontainer
      image: busybox:1.28
      command: ['sh', '-c', 'echo The app is running! && sleep 3600']
    - name: logsidecar
      image: busybox:1.28
      command: ['sh', '-c', 'while true; do cat /var/log/app.log; sleep 10; done']
  volumeMounts:
    - name: app-logs
      mountPath: /var/log
volumes:
  - name: app-logs
    emptyDir: {}
```

Multicontainer Pod



```
root@thinknyx:~$ kubectl logs myapp-pod -c myapp-container  
root@thinknyx:~$ kubectl logs myapp-pod -c log-sidecar
```

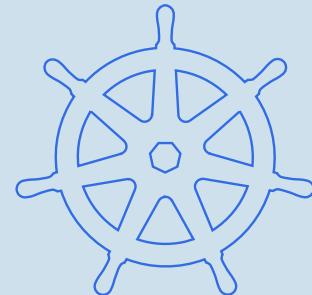
Multicontainer Pods in Kubernetes help build modular, scalable applications by using patterns like sidecar, ambassador, and adapter containers, boosting both productivity and maintainability





Demo

Multicontainer
Pod



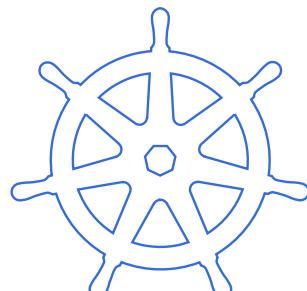
Resource Limits & Requests

Resource Limits & Requests

- Specifying resource requests and limits for containers in Kubernetes optimizes resource usage and prevents overconsumption in the cluster
- Helps to optimize resource usage and maintain stability across the cluster

Resource Requests

Resource Limits



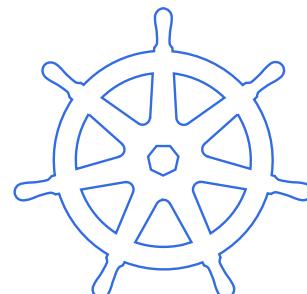
Resource Limits & Requests

Set the minimum amount of CPU or memory a container needs

Resource Requests

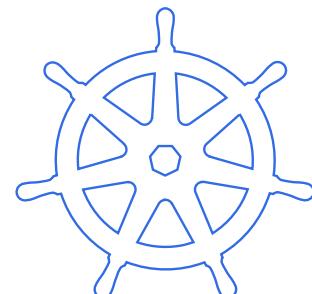
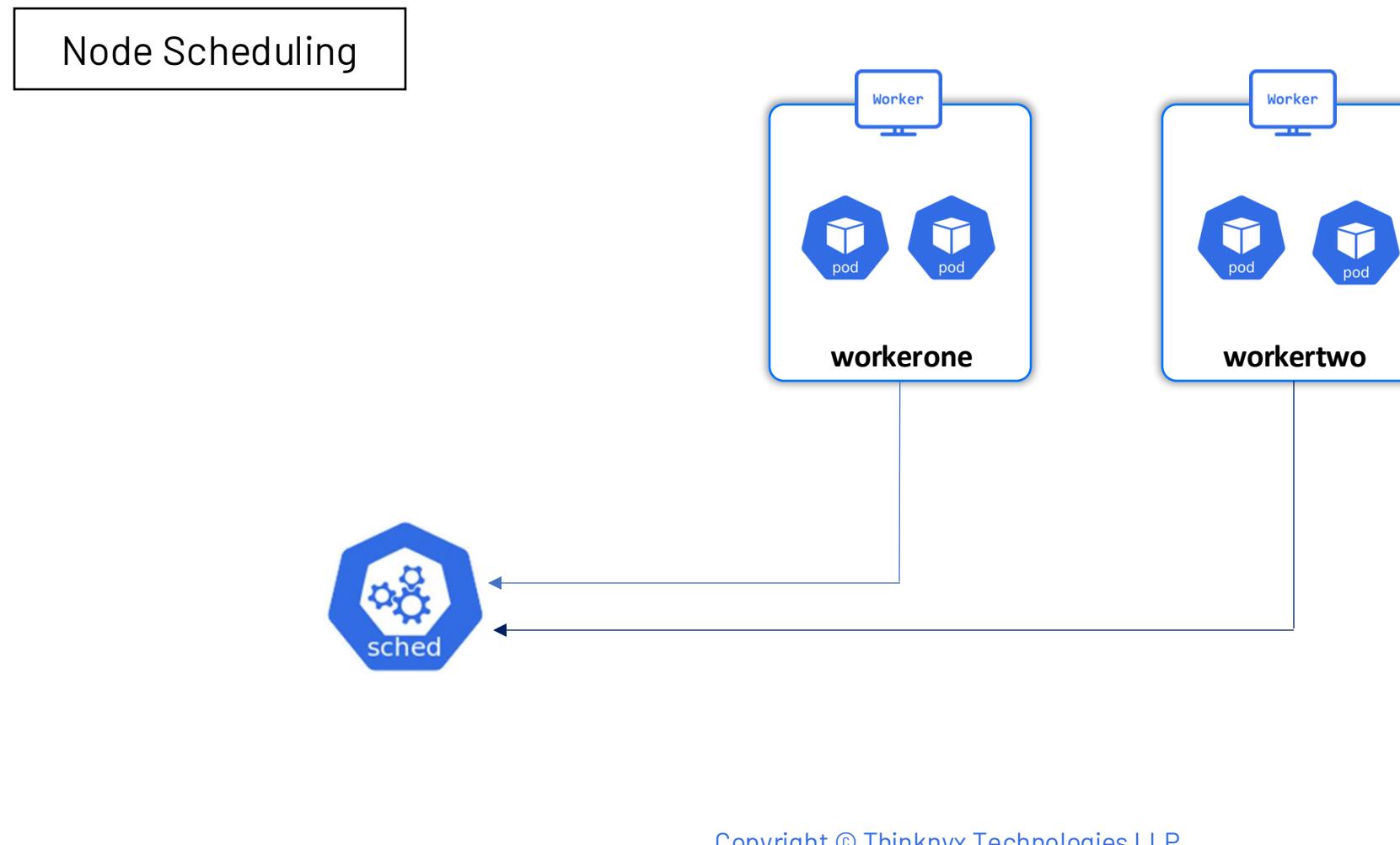
Resource Limits

Set the maximum amount of CPU or memory a container can use



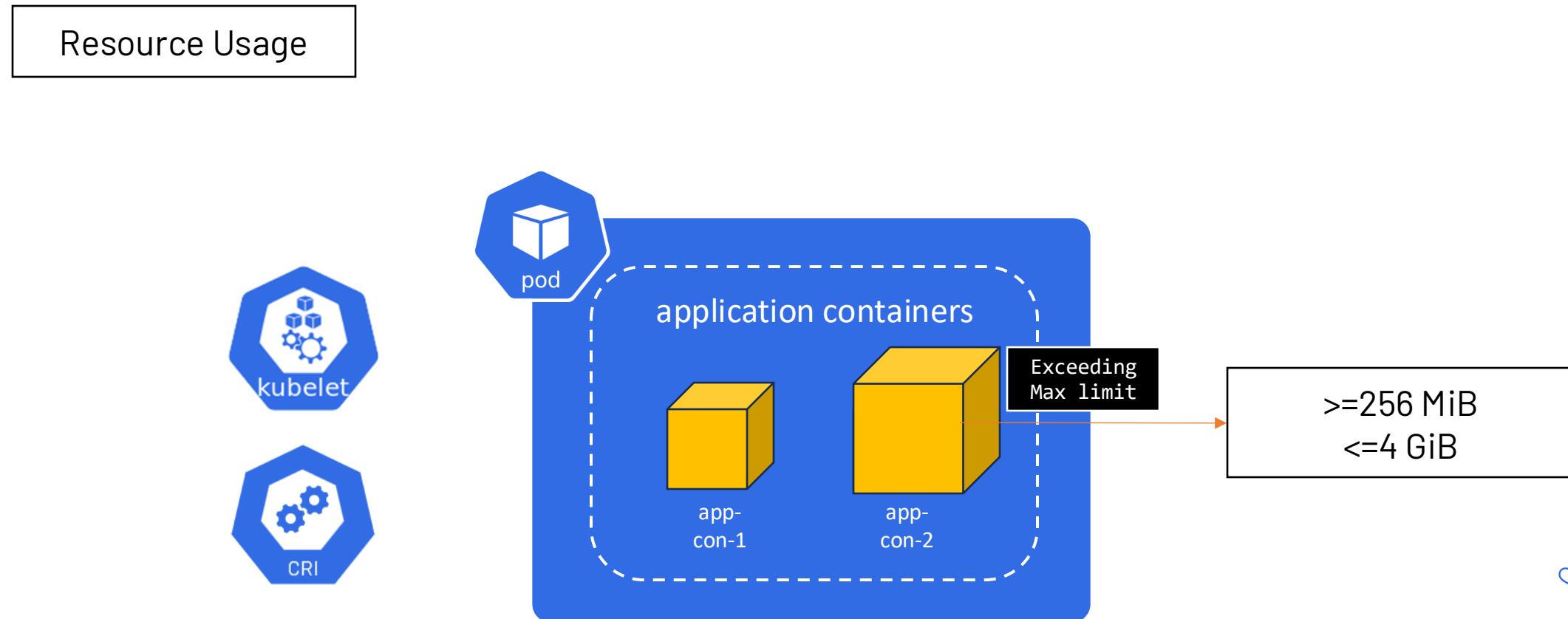
Resource Limits & Requests

How requests & limits work?



Resource Limits & Requests

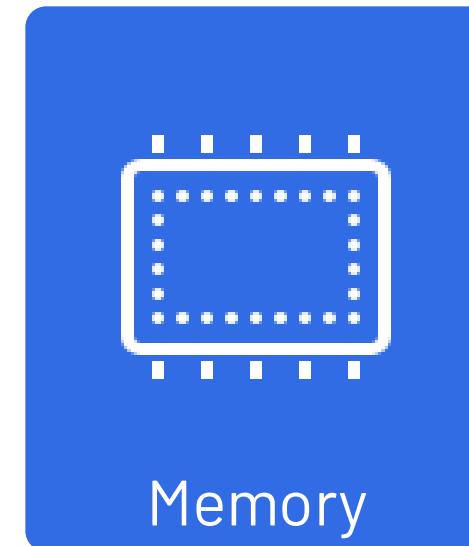
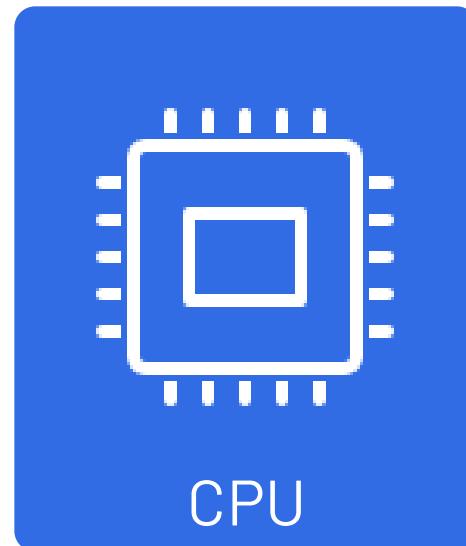
How requests & limits work?



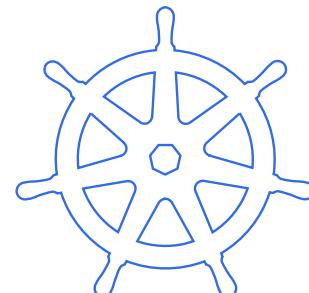
Resource Limits & Requests

Resource Types

- One CPU unit equals one physical or virtual core
- Fractional requests are allowed (0.5 CPU or 500m)



- Measured in bytes
- Suffixes: Ki (kilobytes), Mi (megabytes), and Gi (gigabytes).





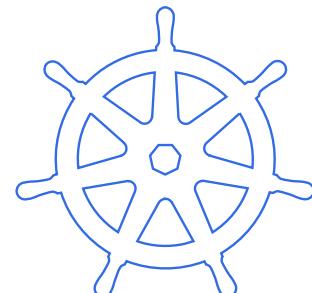
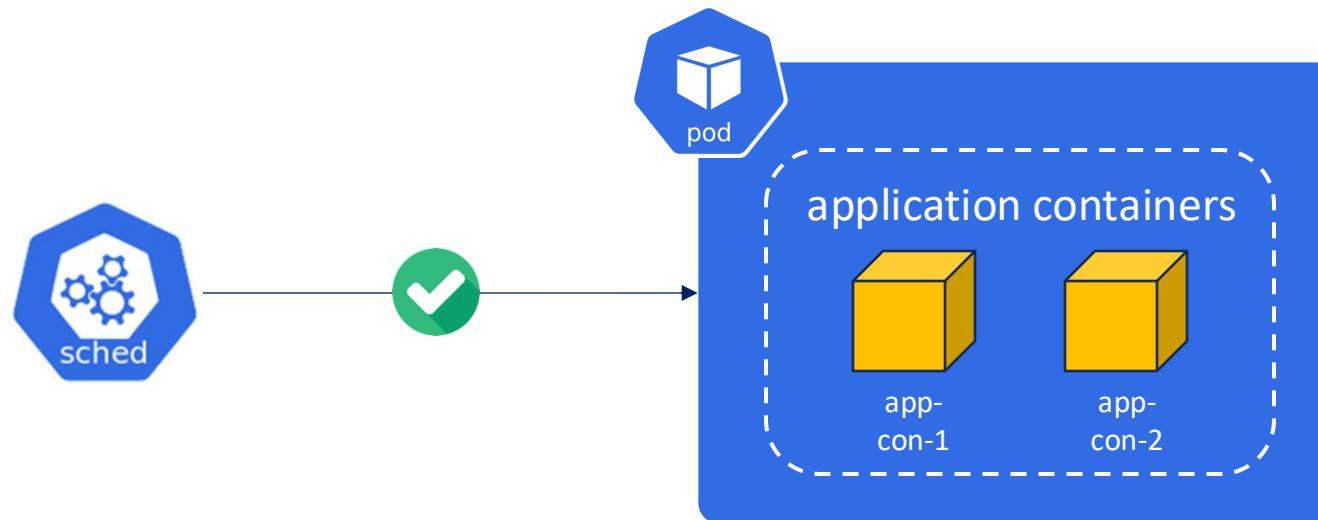
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: app
      image: httpd:v1
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
    - name: redis
      image: redis:v1
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

Total Request: 0.5 CPU & 128 MiB
Total Limit: 1 CPU & 256 MiB

Resource Limits & Requests

How Kubernetes Applies Requests and Limits

During Scheduling



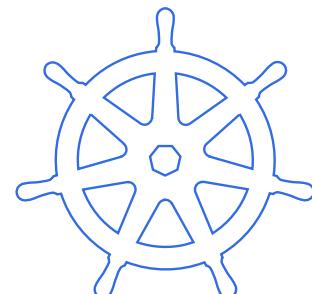
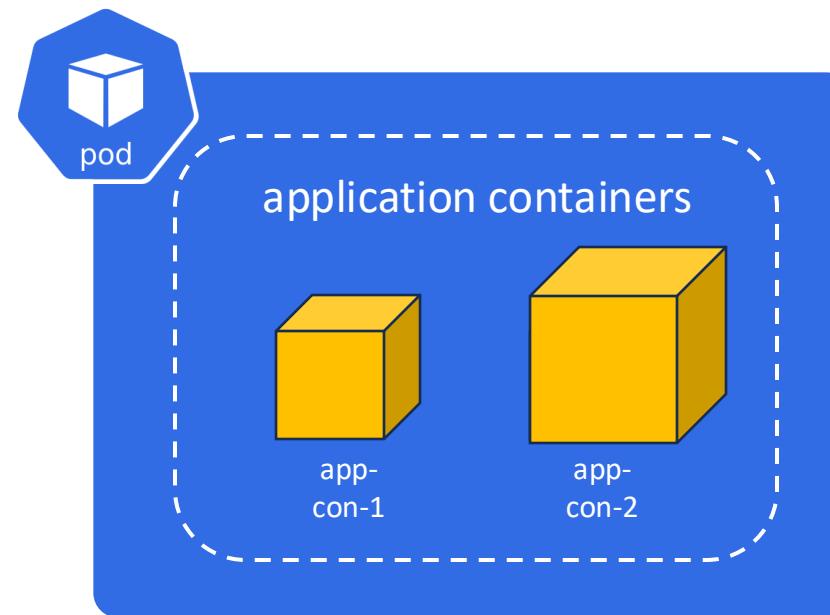
Resource Limits & Requests

How Kubernetes Applies Requests and Limits

During Execution



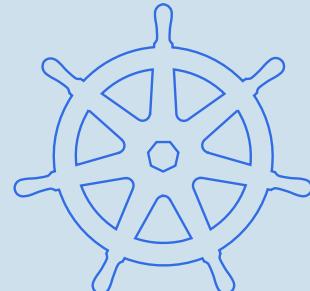
cgroups





Demo

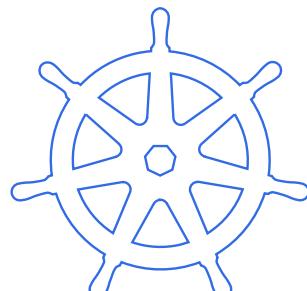
Resource Limits &
Requests



Liveness Probe

Liveness Probe

- Liveness probes check if containers are healthy and functioning, with the kubelet managing their status
- If a liveness probe fails, the kubelet kills the container, which is then restarted based on the Pod's restart policy

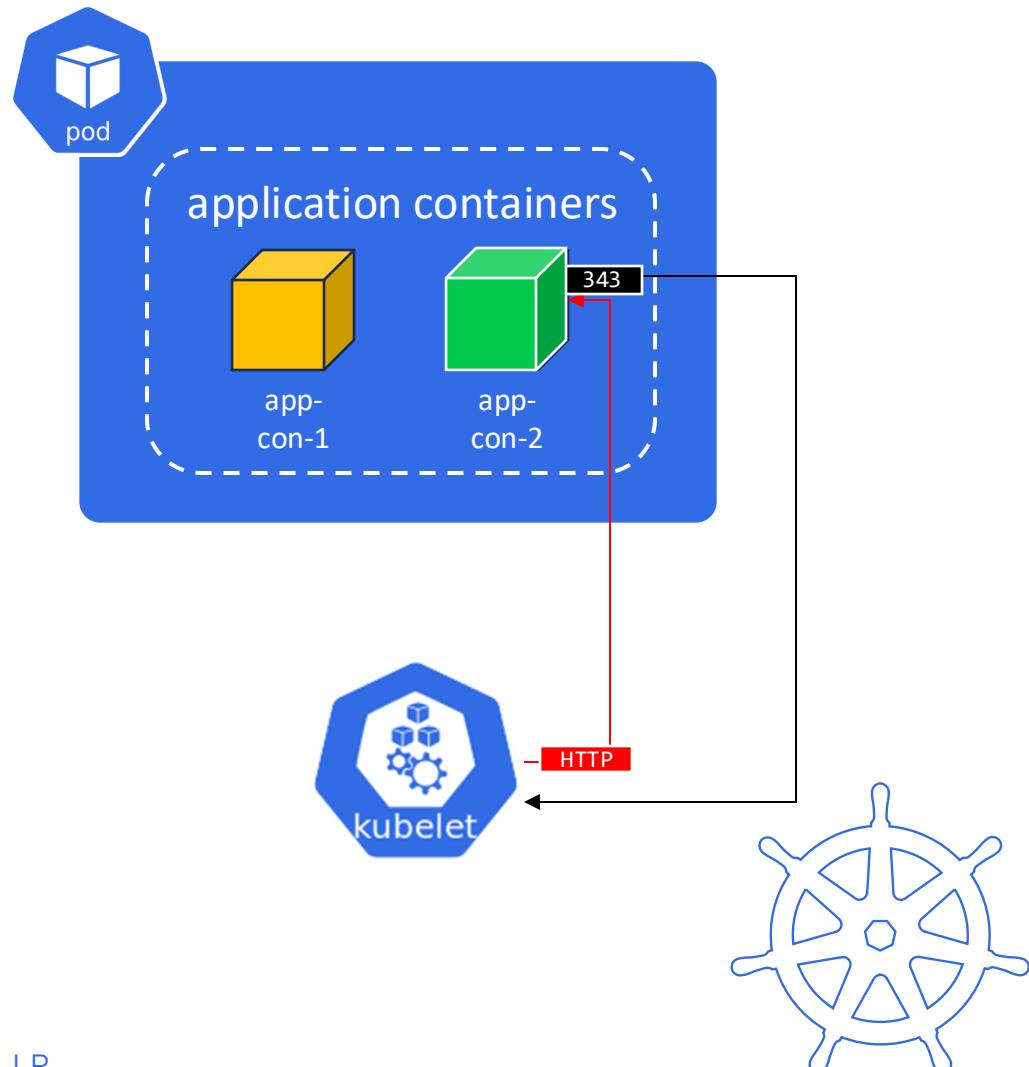


Liveness Probe

HTTP Request

Example configuration:

```
livenessProbe:  
  httpGet:  
    path: /healthz  
    port: 8080  
  initialDelaySeconds: 3  
  periodSeconds: 3
```

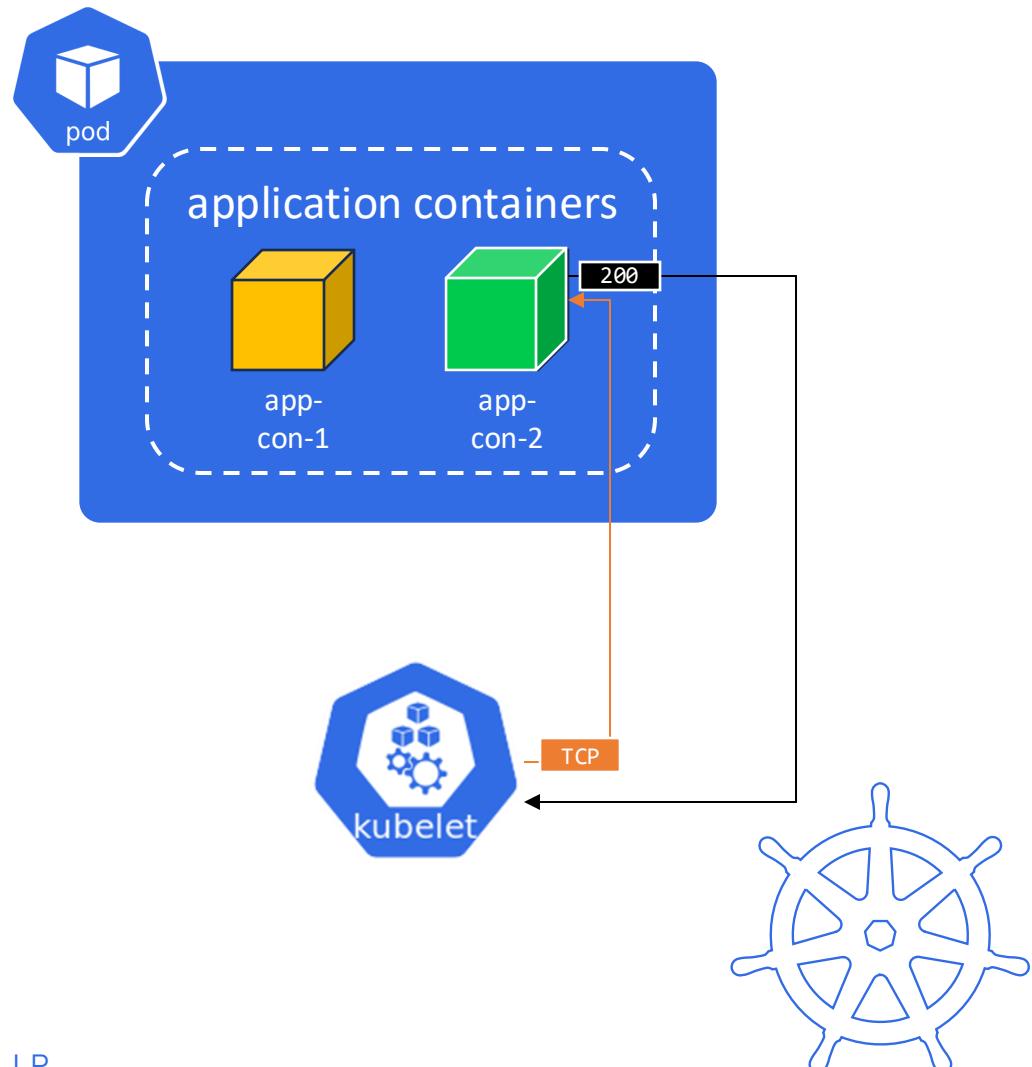


Liveness Probe

TCP Socket

Example configuration:

```
livenessProbe:  
  tcpSocket:  
    port: 8080  
    initialDelaySeconds: 15  
    periodSeconds: 10
```



Liveness Probe

Exec Command

Example configuration:

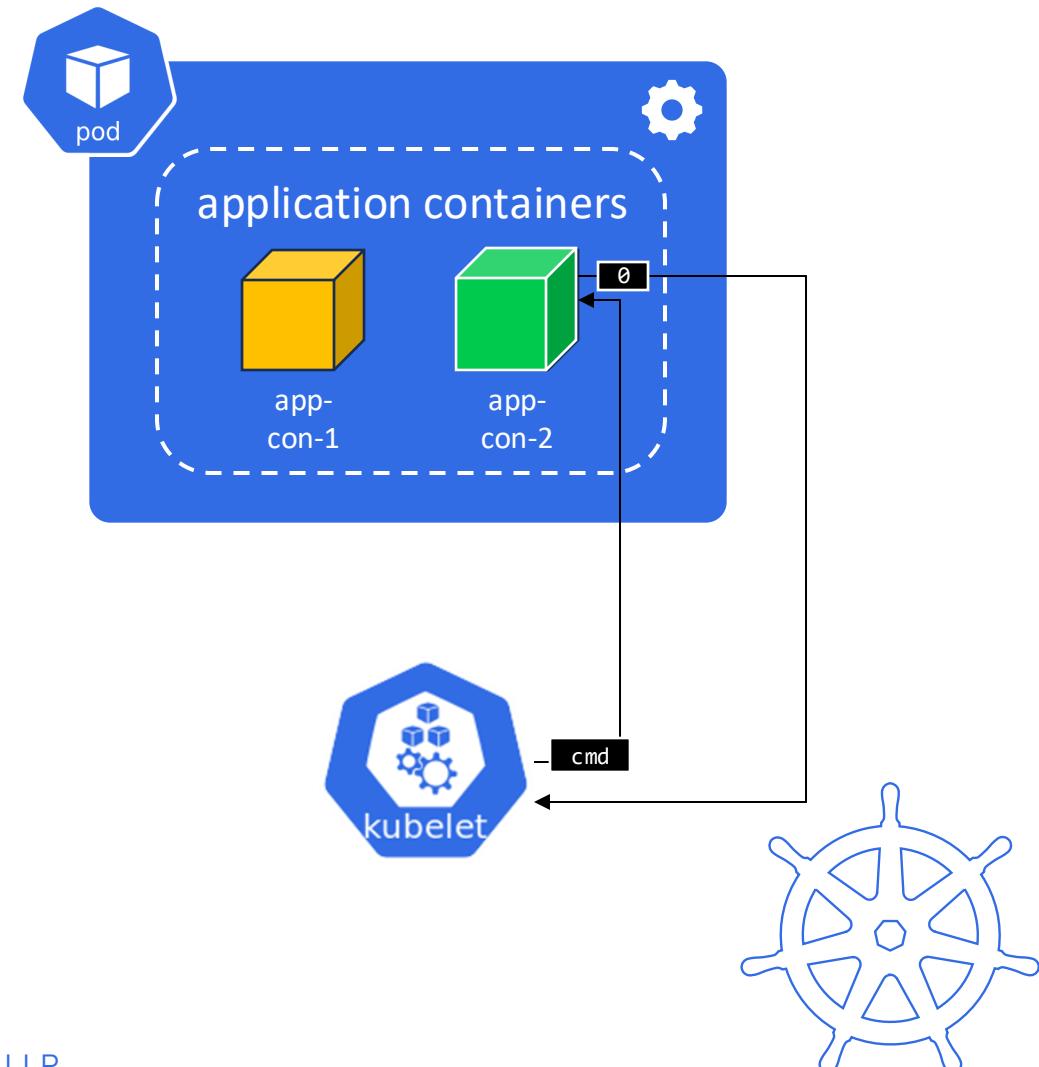
livenessProbe:

exec:

```
  command:  
    - cat  
    - /tmp/healthy
```

```
initialDelaySeconds: 5
```

```
periodSeconds: 5
```



Liveness Probe

Key Parameters

`initialDelaySeconds`

Time to wait after container start before the first probe

`periodSeconds`

Interval between successive probes

`timeoutSeconds`

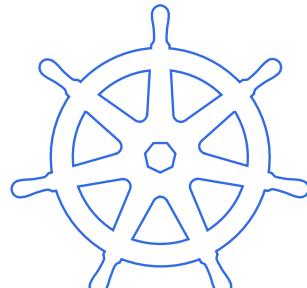
Time to wait for a probe to complete before timing out

`successThreshold`

Consecutive successes required to consider the probe successful after failure

`failureThreshold`

Consecutive failures required to mark the container as failed and kill it



Liveness Probe

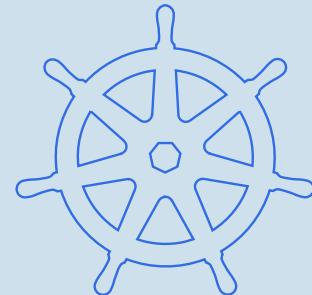


```
apiVersion: v1
kind: Pod
metadata:
  name: livenesspod
spec:
  containers:
    - name: myapp
      image: tomcat:latest
      ports:
        - containerPort: 8080
      livenessProbe:
        httpGet:
          path: /healthz
          port: 8080
        initialDelaySeconds: 3
        periodSeconds: 3
        timeoutSeconds: 1
        successThreshold: 1
        failureThreshold: 3
```



Demo

| Liveness Probe



Readiness Probe

Readiness Probe

- Readiness probes ensure a container is ready to accept traffic and, if a probe fails, the container is removed from active service endpoints

HTTP Request

Example configuration:

```
readinessProbe:  
  httpGet:  
    path: /ready  
    port: 8080  
  initialDelaySeconds: 5  
  periodSeconds: 5
```

TCP Socket

Example configuration:

```
readinessProbe:  
  tcpSocket:  
    port: 8080  
  initialDelaySeconds: 15  
  periodSeconds: 10
```

Exec Command

Example configuration:

```
readinessProbe:  
  exec:  
    command:  
      - cat  
      - /tmp/ready  
  initialDelaySeconds: 5  
  periodSeconds: 5
```



Readiness Probe

Key Parameters

`initialDelaySeconds`

Time to wait after container start before the first probe

`periodSeconds`

Time between successive probes

`timeoutSeconds`

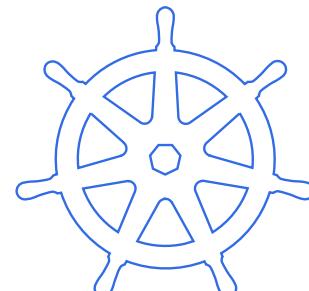
Time to wait for a probe to complete before timing out

`successThreshold`

Consecutive successes required to consider the probe successful after failure

`failureThreshold`

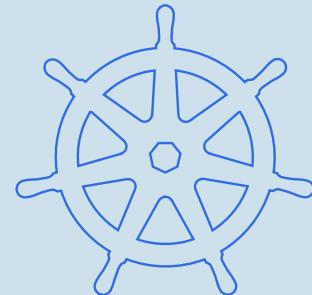
Consecutive failures required to mark the container as failed and kill it





Demo

| Readiness Probe



Startup Probe

Startup Probe

- Startup probes handle containers with slow initialization, ensuring they're not killed by liveness or readiness probes until fully started

HTTP Request

Example configuration:

```
startupProbe:  
  httpGet:  
    path: /startup  
    port: 8080  
  failureThreshold: 30  
  periodSeconds: 10
```

TCP Socket

Example configuration:

```
startupProbe:  
  tcpSocket:  
    port: 8080  
    failureThreshold: 30  
    periodSeconds: 10
```

Exec Command

Example configuration:

```
startupProbe:  
  exec:  
    command:  
      - cat  
      - /tmp/startup  
  failureThreshold: 30  
  periodSeconds: 10
```



Startup Probe

Key Parameters

`initialDelaySeconds`

Time to wait after container start before the first probe

`periodSeconds`

Time between successive probes

`timeoutSeconds`

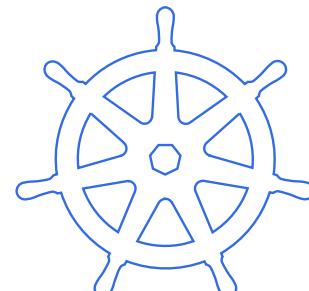
Time to wait for a probe to complete before timing out

`successThreshold`

Consecutive successes required to consider the probe successful after failure

`failureThreshold`

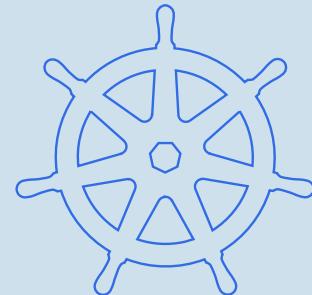
Consecutive failures required to mark the container as failed and kill it





Demo

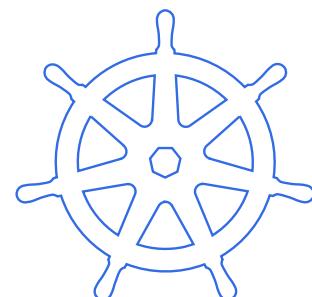
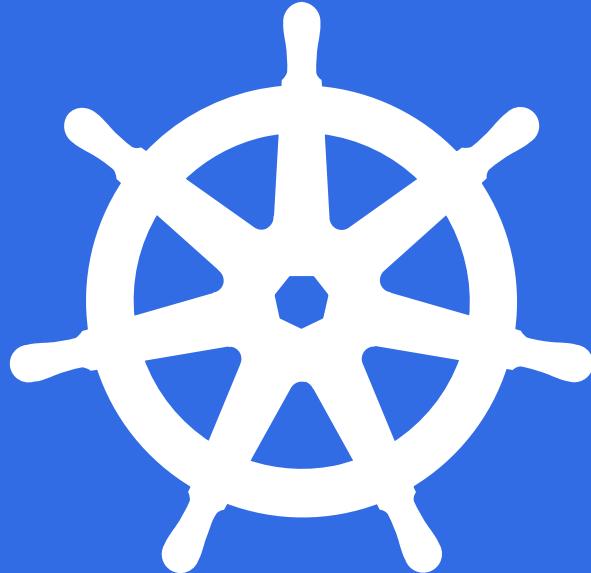
Startup Probe



Summary

Summary

- ✓ Init containers (handling pre-configuration tasks)
- ✓ Multicontainer pods (co-locating microservices)
- ✓ Limits and requests (resources management)
- ✓ Liveness, Readiness, & Startup probes

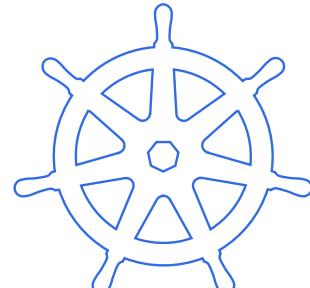


Section: 12

Section Overview

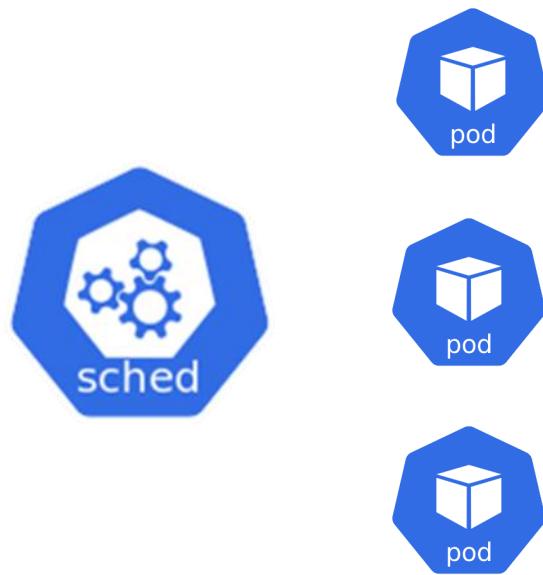
Scheduling

- ↳ **nodeName**
- ↳ **nodeSelector**
- ↳ **Node affinity & anti-affinity**
- ↳ **Inter-pod affinity & anti-affinity**
- ↳ **Taints & Tolerations**

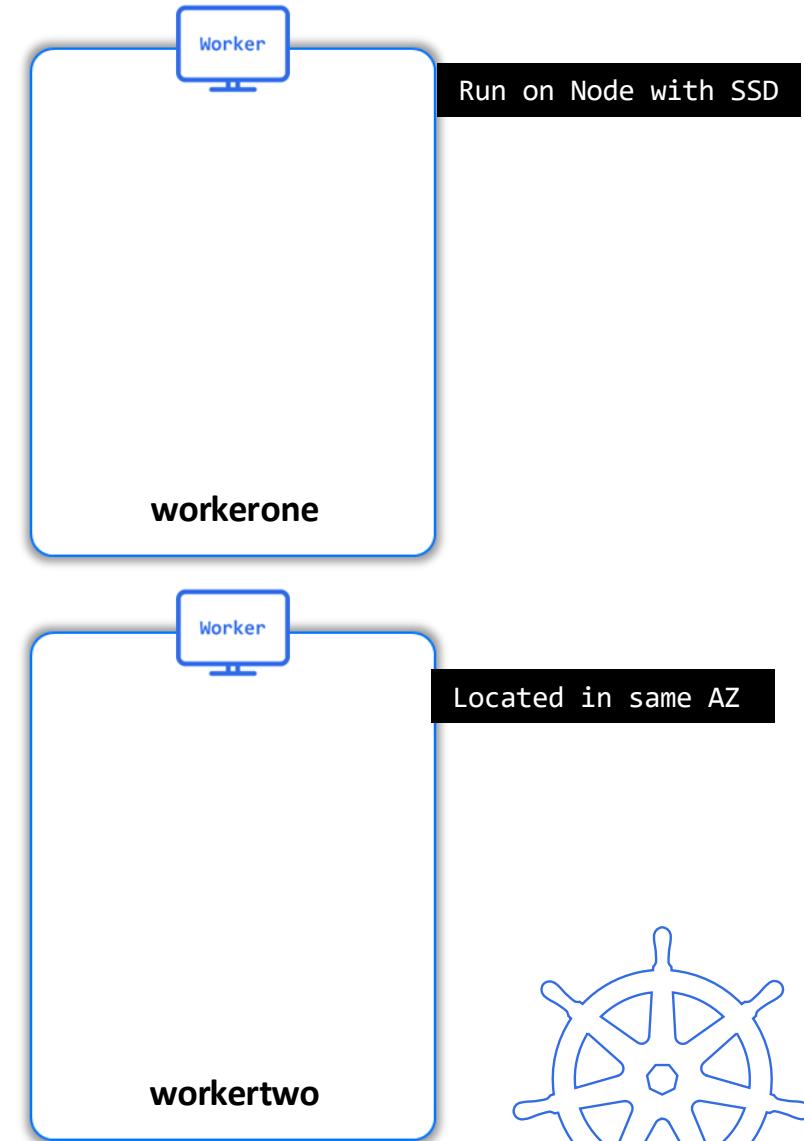


nodeName

nodeName



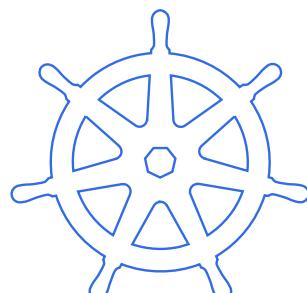
nodeName: workerone



nodeName

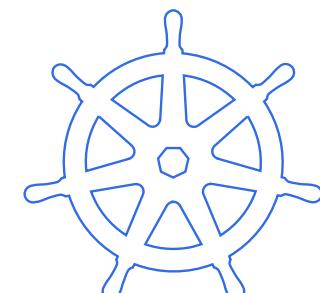
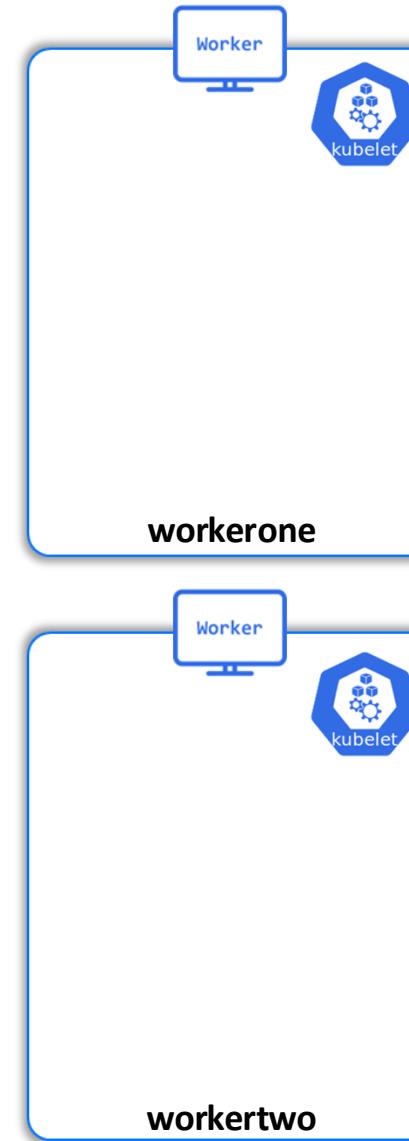
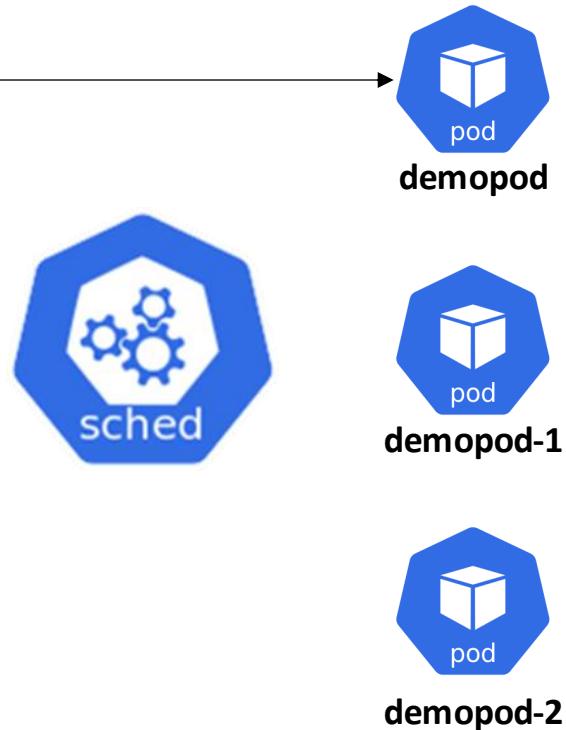
workerone

```
apiVersion: v1
kind: Pod
metadata:
  name: demopod
spec:
  containers:
    - name: demopodcon
      image: nginx:latest
  nodeName: workerone
```



nodeName

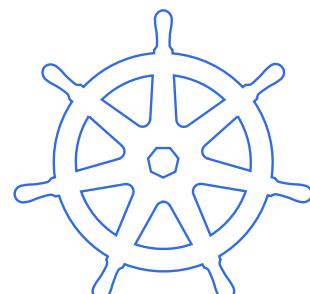
```
apiVersion: v1
kind: Pod
metadata:
  name: demopod
spec:
  containers:
  - name: demopodcon
    image: nginx:latest
  nodeName: workerone
```



nodeName

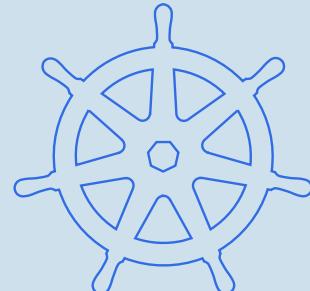
Limitations to consider

- A Pod won't schedule if the specified node is missing
- Typos in node names can prevent scheduling
- The node must have enough CPU and memory for the Pod
- Pods can fail to start if resources are insufficient
- Common errors include "OutOfMemory" and "OutOfCpu"
- In cloud environments, node names might not always be stable due to scaling events





Demo | nodeName

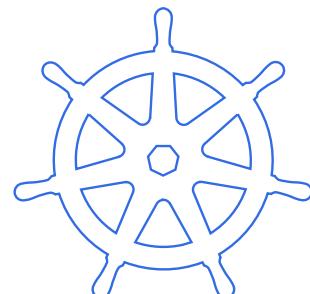


nodeSelector

nodeSelector

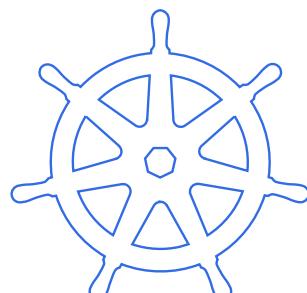
nodeSelector:

allows you to schedule Pods
based on node labels



nodeSelector

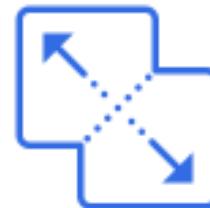
- Enables the specification of key-value pairs in node labels, ensuring that Pods are scheduled only on matching nodes
- More flexible than nodeName, as it allows multiple nodes to be labeled similarly, giving the scheduler more options to find the best fit



Why Label Nodes?



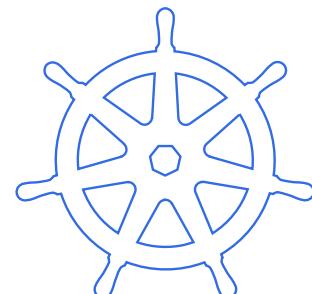
Resource
Management



Environment
Segregation



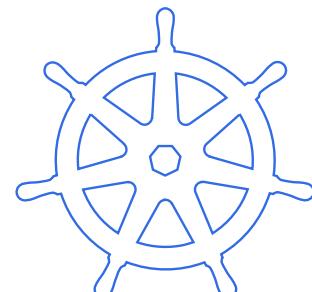
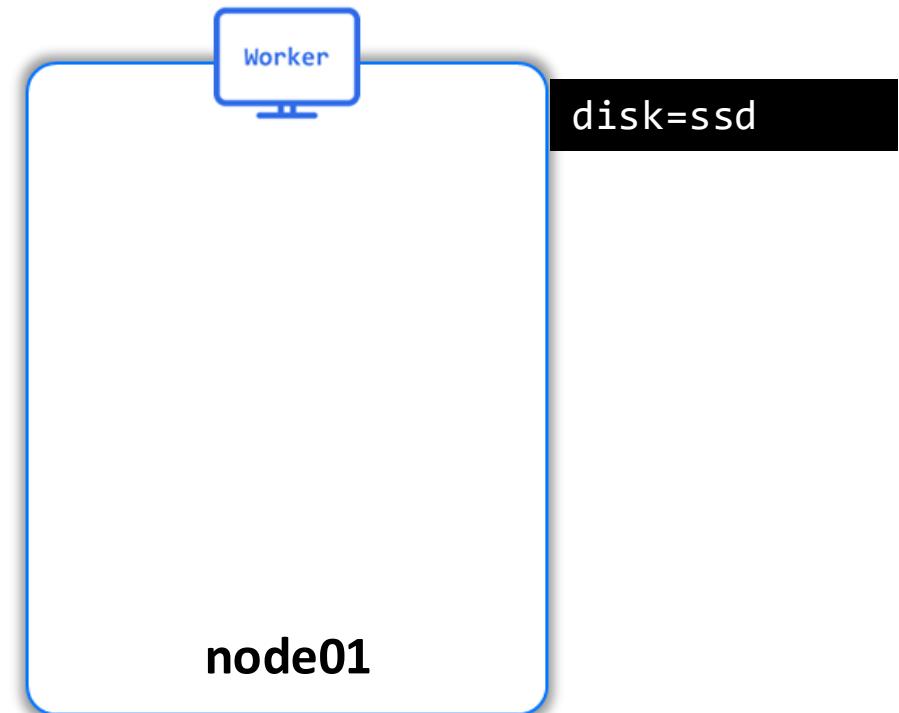
Workload
Isolation



How to Label Nodes?

kubectl

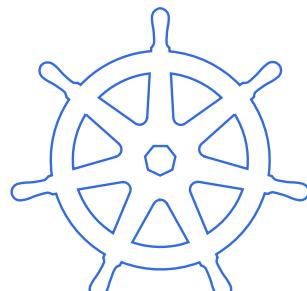
```
kubectl label nodes node01 disk=ssd
```



How to Label Nodes?

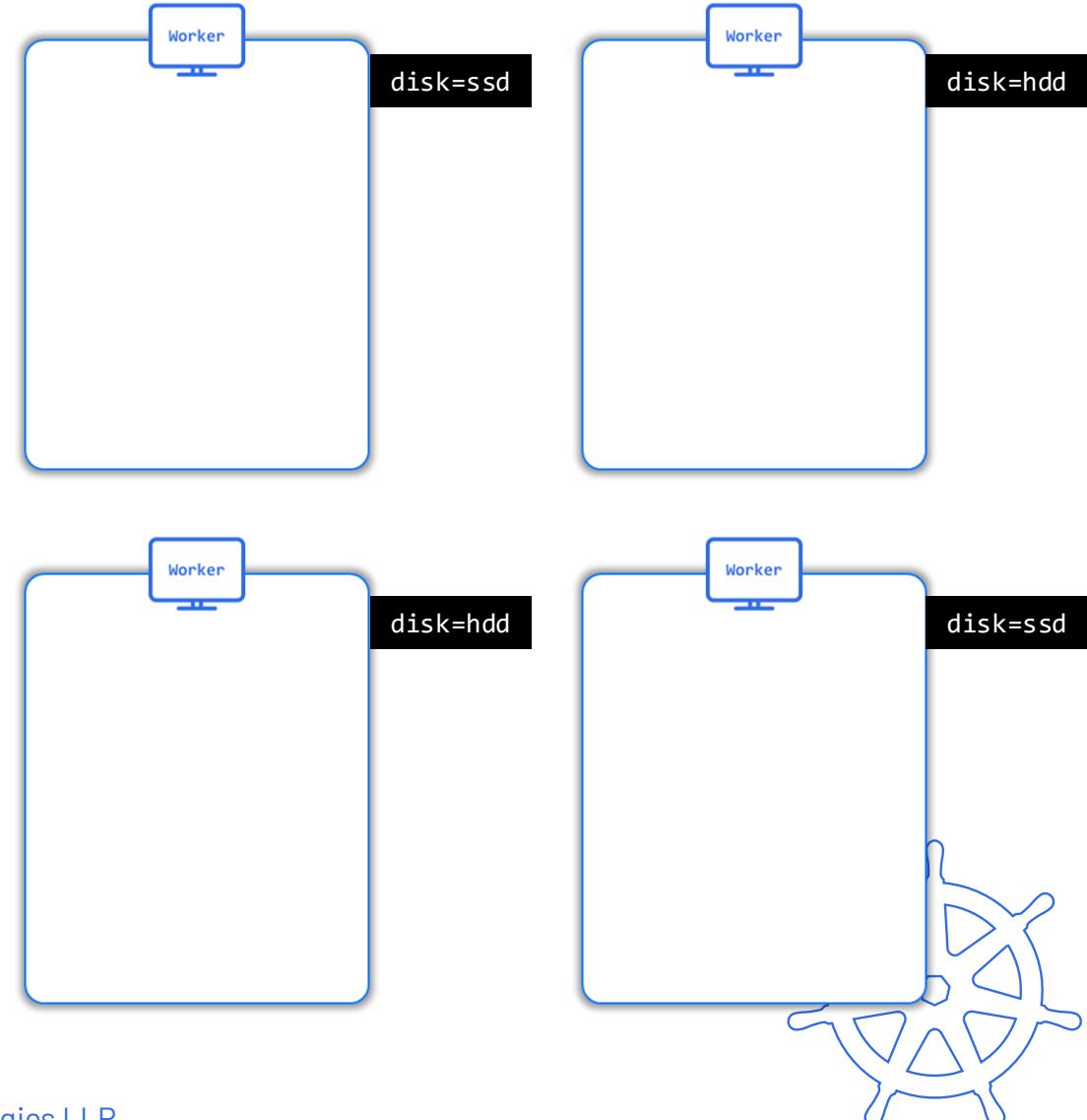
scheduler will look for nodes
labeled with **disk=ssd**

```
apiVersion: v1
kind: Pod
metadata:
  name: demopod
spec:
  containers:
    - name: demopodcon
      image: nginx:latest
  nodeSelector:
    disk: ssd
```



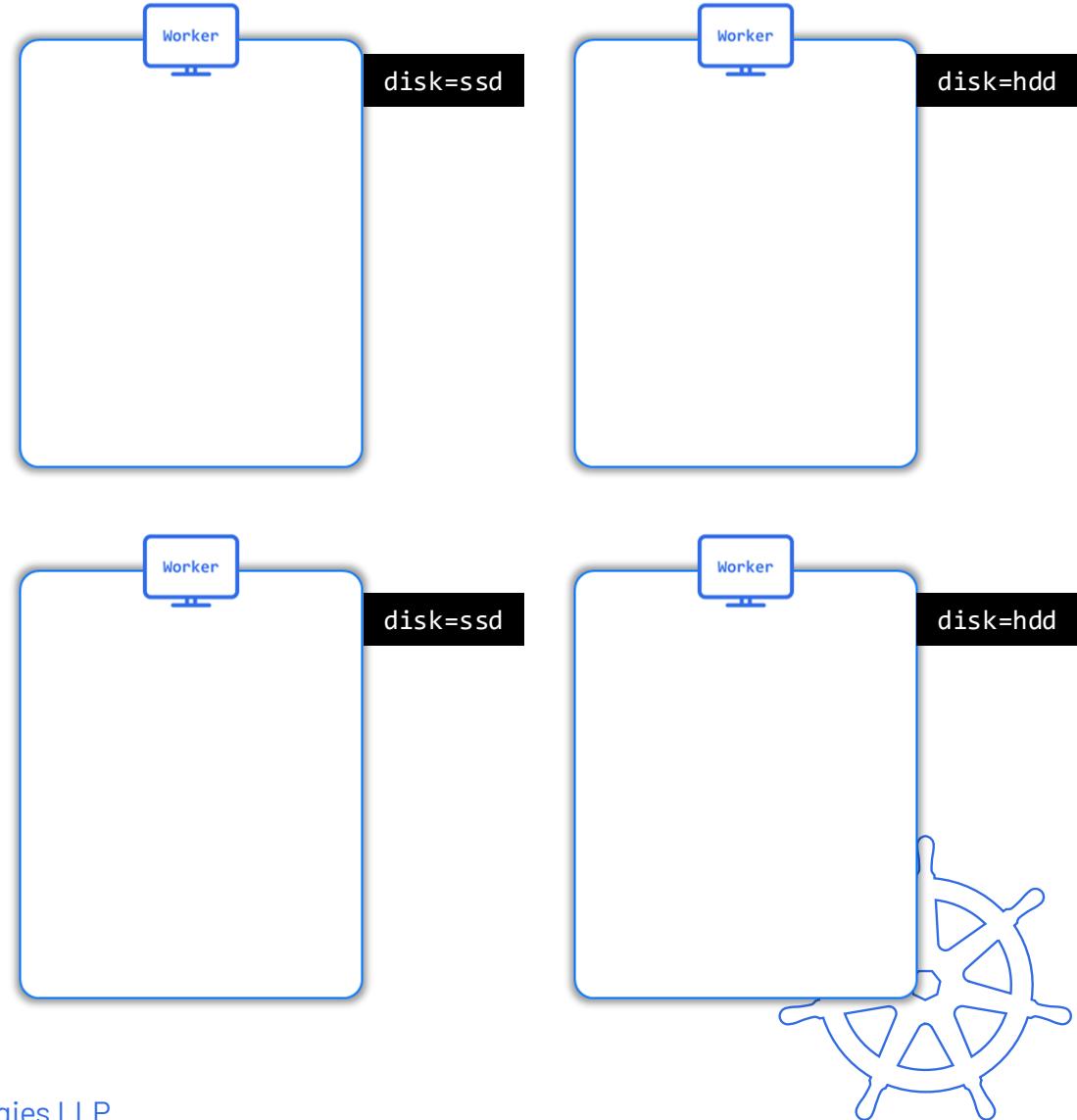
How nodeSelector Works?

```
apiVersion: v1
kind: Pod
metadata:
  name: demopod
spec:
  containers:
  - name: demopodcon
    image: nginx:latest
  nodeSelector:
    disk: ssd
```



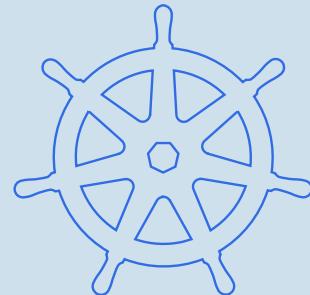
How nodeSelector Works?

```
apiVersion: v1
kind: Pod
metadata:
  name: demopod
spec:
  containers:
  - name: demopodcon
    image: nginx:latest
  nodeSelector:
    disk: ssd
```





Demo | nodeSelector



Node Affinity

Affinity & Anti-affinity

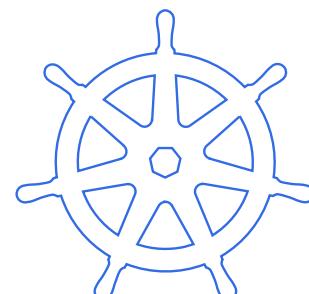
- Allows you to define complex rules for pod scheduling based on node & pod labels
- These rules help in pod scheduling based on specific criteria or pod placement preferences

Affinity

allows you to specify preferred nodes or pods for scheduling your pods

Anti-affinity

allows you to indicate nodes or pods that your pods should avoid

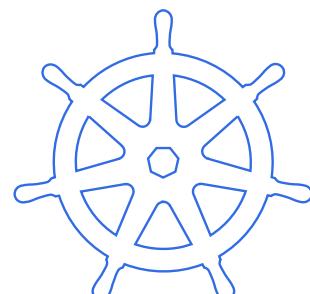


Node Affinity

nodeSelector:

nodeSelector Limitations

- Only supports equality-based requirements
- Lacks expressiveness for more complex scheduling needs
- Can't express preferences, only hard requirements



Node Affinity

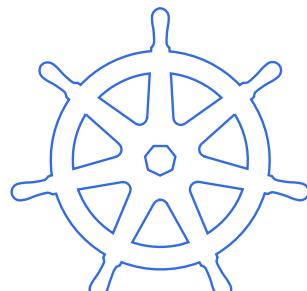
How Affinity & Anti-affinity differs from nodeSelector

Expressiveness

- More expressive
- Uses operator like – In, NotIn, Exists etc, allowing for complex scheduling rules

Soft & Hard Constraints

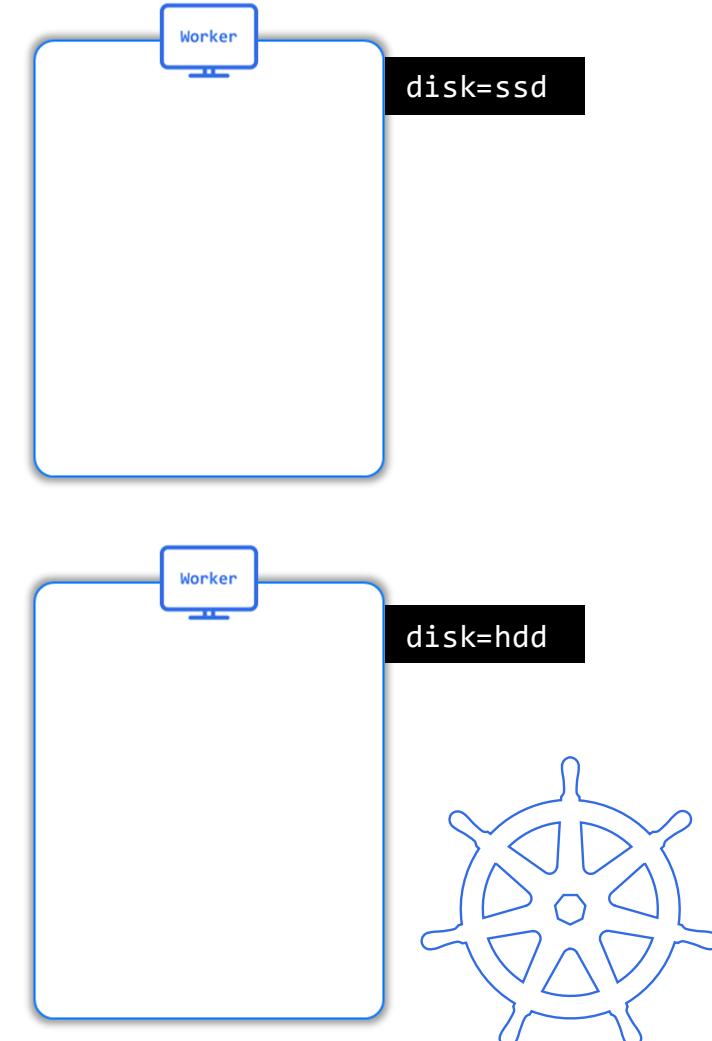
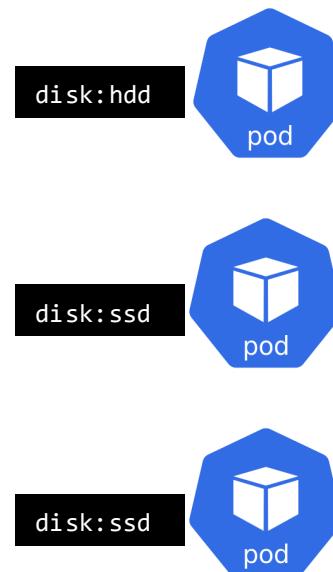
- Define rules as either hard (required) or soft (preferred)
- Soft constraints let the scheduler place the pod even if no nodes match
- Hard constraints must be met for scheduling



Node Affinity

- Node affinity lets you constrain which nodes your pod can be scheduled on based on node labels
- More flexible and expressive than nodeSelector

`.spec.affinity.nodeAffinity`



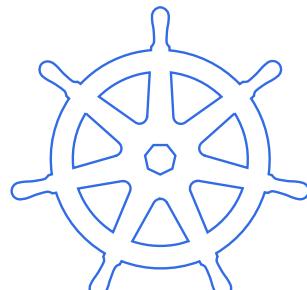
Node Affinity

requiredDuringScheduling;IgnoredDuringExecution

- This rule is a hard requirement
- Scheduler will not schedule the pod unless the rule is met
- Functions similarly to nodeSelector but with more expressive syntax

preferredDuringScheduling;IgnoredDuringExecution

- This rule is a soft preference
- The scheduler tries to find a node that meets the rule, but if no matching node is available, the scheduler will still schedule the pod
- Provides flexibility, allowing the scheduler to prefer certain nodes without strictly enforcing the rule



```
root@master:/var/tmp/k8sfiles# cat 34_nodeaffinitytest_dep.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nodeaffinitytest
    name: nodeaffinitytest
spec:
  replicas: 2
  selector:
  - matchLabels:
      app: nodeaffinitytest
  template:
    metadata:
      labels:
        app: nodeaffinitytest
    spec:
      containers:
      - image: nginx:latest
        name: nginx
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: environment
                operator: In
                values:
                - production
```



Node Affinity

In

label key must be one of the specified values

NotIn

label key must not be one of the specified values

Exists

label key must exist

DoesNotExist

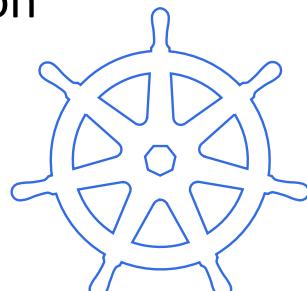
label key must not exist

Gt

label key must be greater than a specified value using numeric comparison

Lt

label key must be less than a specified value using numeric comparison



Node Affinity

Use-cases



Hardware-Specific
Workloads



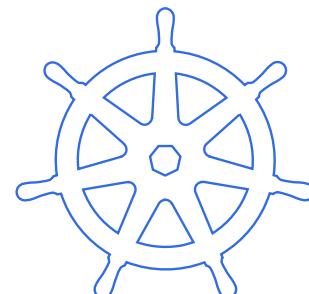
Compliance &
Security



Operational
Policies

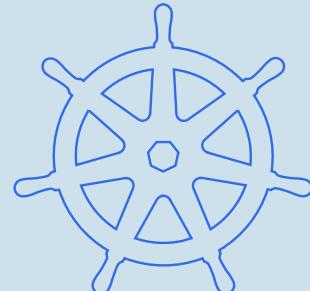


Multi-Tenancy





Demo | Node Affinity



Node Anti-Affinity

Node Anti-Affinity

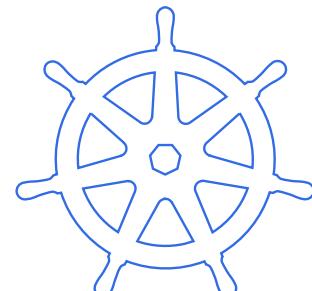
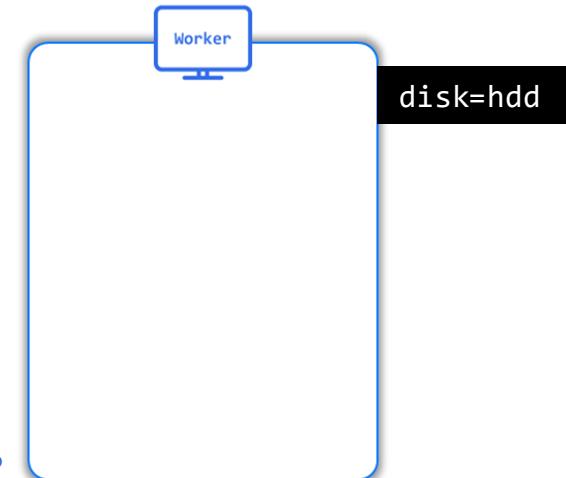
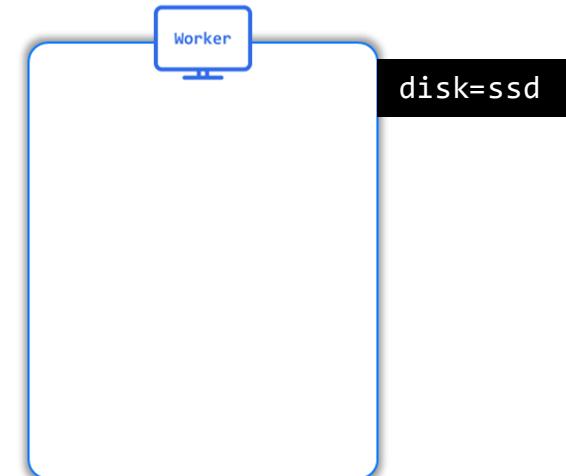
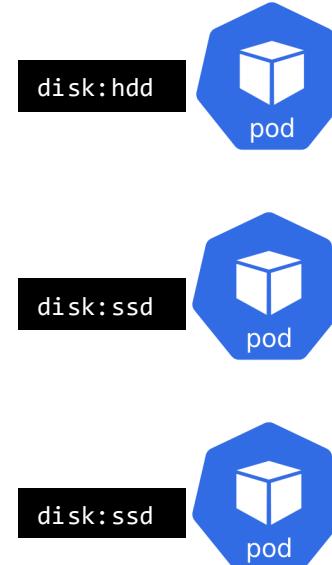
- Node anti-affinity specifies where pods should not be placed
- Useful for avoiding certain nodes or distributing workloads more evenly across the cluster

NotIn

DoesNotExist



.spec.affinity.nodeAffinity



```
root@master:/var/tmp/k8sfiles# cat 35_nodeaffinitytest_dep.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nodeantiaffinitytest
    name: nodeantiaffinitytest
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodeantiaffinitytest
  template:
    metadata:
      labels:
        app: nodeantiaffinitytest
    spec:
      containers:
        - image: httpd:latest
          name: httpd
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: environment
                    operator: NotIn
                    values:
                      - production
```

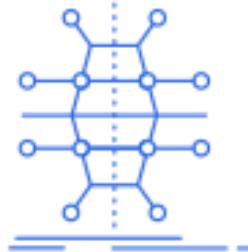


Node Anti-Affinity

Use-cases



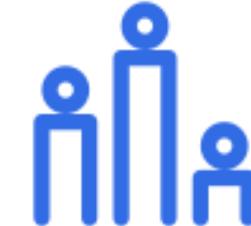
Avoiding Specific
Hardware Types



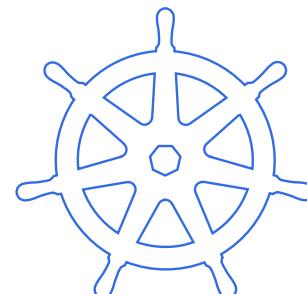
Distributing
Workloads



Compliance &
Security



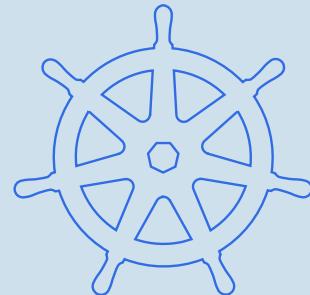
Resource
Contention





Demo

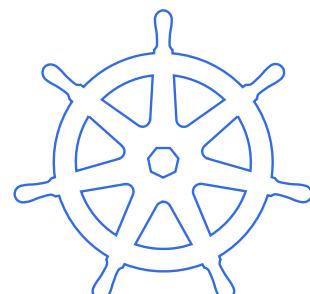
| Node Anti-Affinity



Inter-Pod Affinity

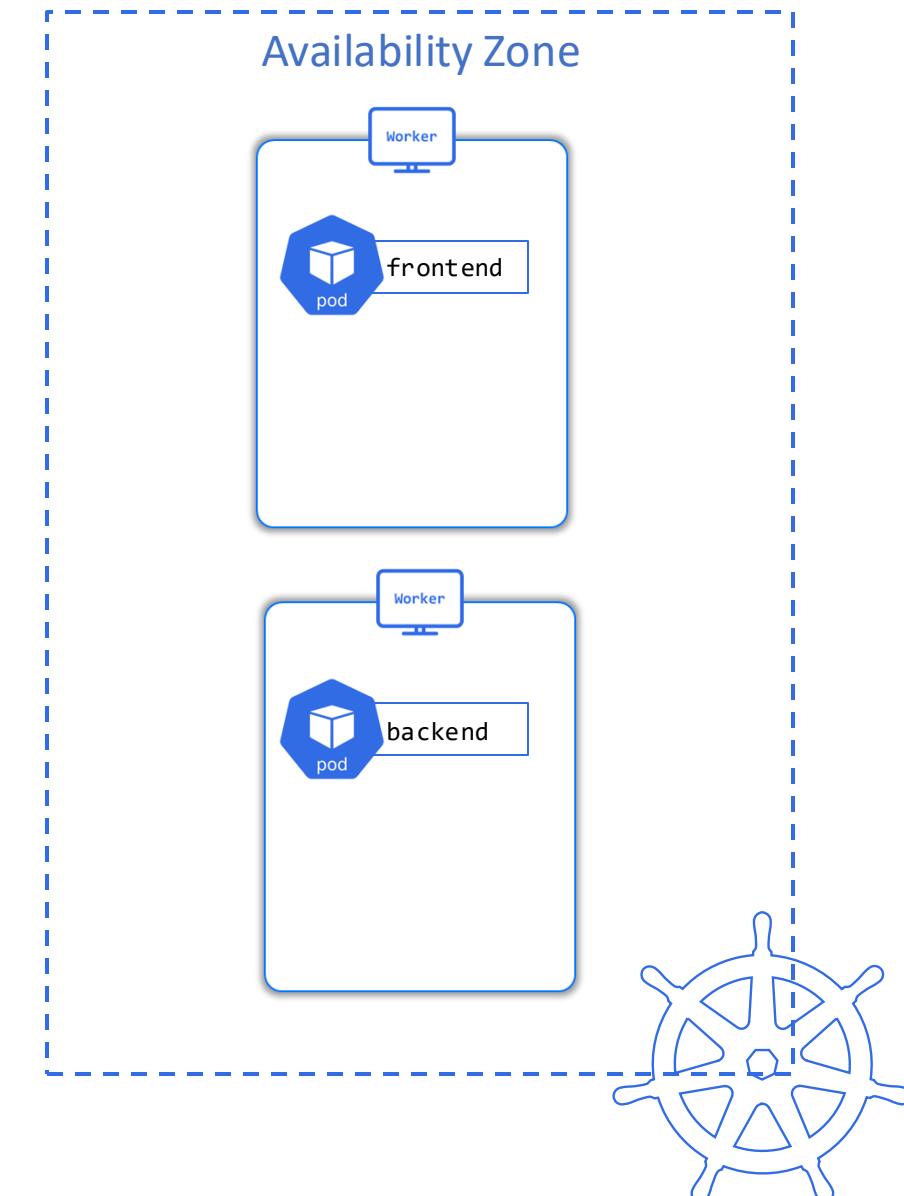
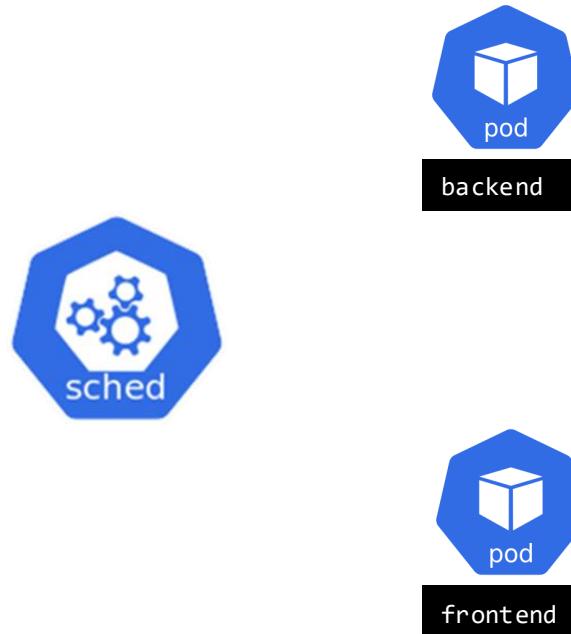
Inter-Pod Affinity

- Allows you to control pod placement based on the labels of existing pods on a node, rather than the node's labels
- Useful in cloud environments with multiple availability zones or data centers



Inter-Pod Affinity

topologyKey: "kubernetes.io/hostname"



Inter-Pod Affinity

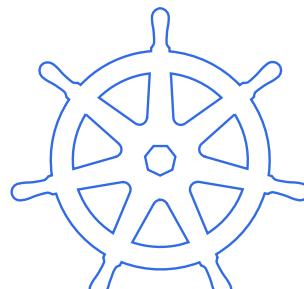
Types

requiredDuringSchedulingIgnoredDuringExecution

- This rule is a hard requirement
- Scheduler will only place the pod on a node that meets this condition

preferredDuringSchedulingIgnoredDuringExecution

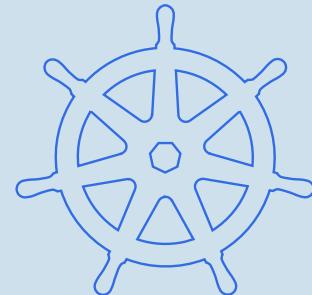
- This rule is a soft preference
- The scheduler tries to place the pod on a node that meets this condition, but it will still place the pod elsewhere if no such node is available





Demo

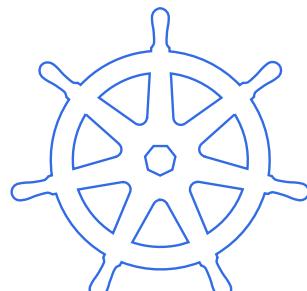
Inter-Pod Affinity



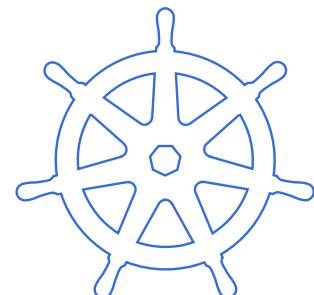
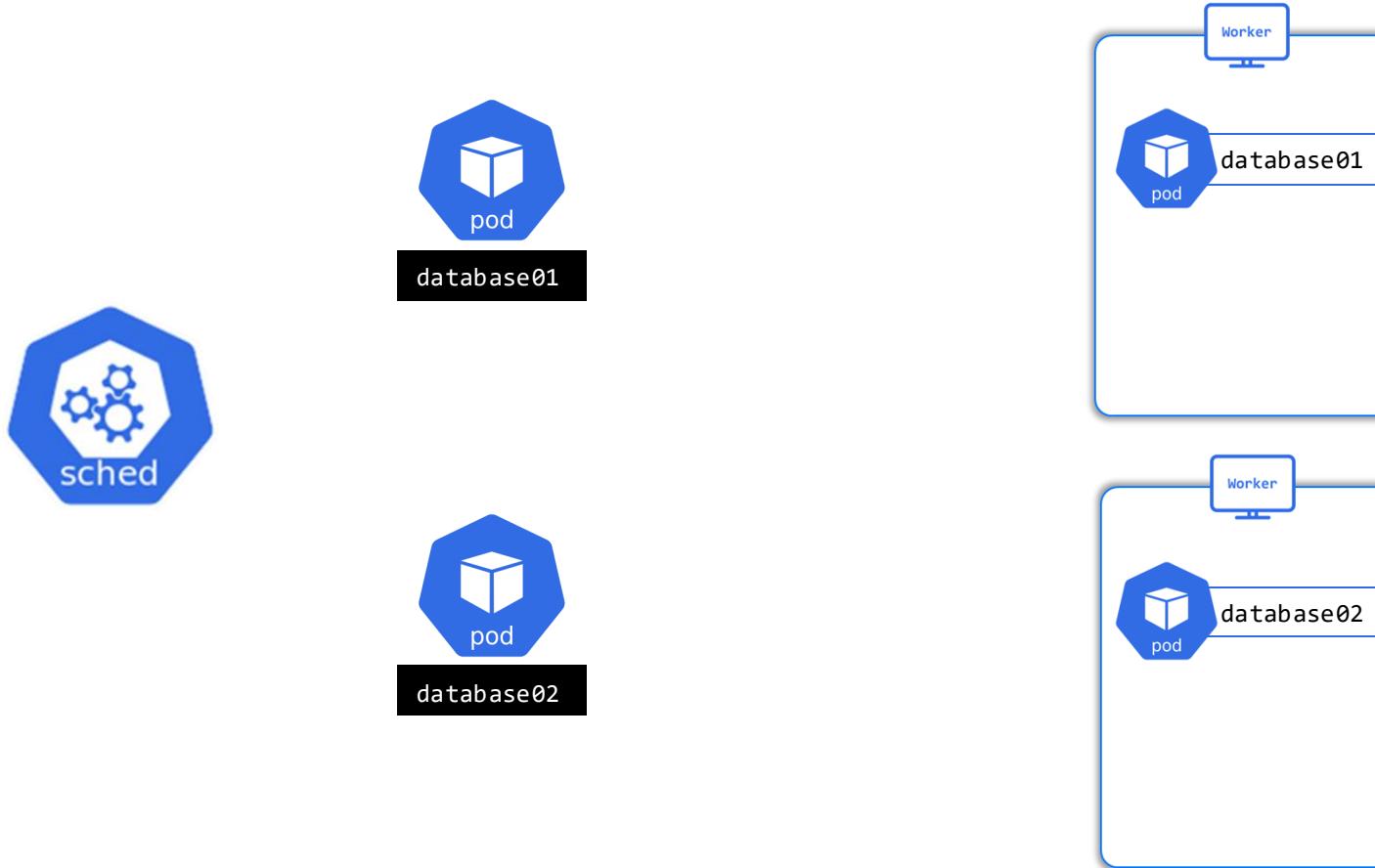
Pod Anti-Affinity

Pod Anti-Affinity

- Controls pod placement by ensuring pods with specified labels are not scheduled on the same node or domain
- Useful when you want to distribute certain applications or components across different nodes or data centers



Pod Anti-Affinity



Pod Anti-Affinity

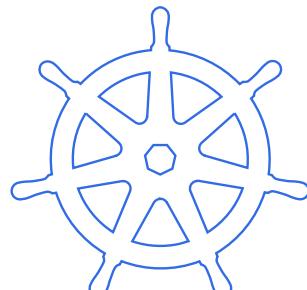
Types

requiredDuringSchedulingIgnoredDuringExecution

- This rule is a hard requirement
- scheduler will not place the pod on a node that meets this condition

preferredDuringSchedulingIgnoredDuringExecution

- This rule is a soft preference
- The scheduler tries to avoid placing the pod on a node that meets this condition, but it will still place the pod there if no other node is available



Pod Anti-Affinity

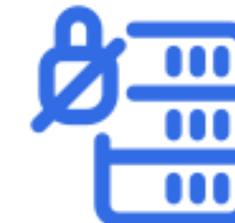
Important Considerations



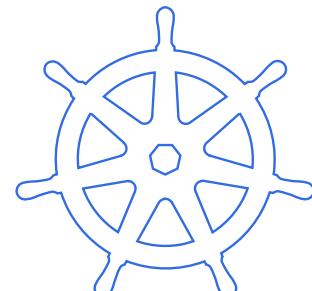
Processing
Overhead



Consistent
Labelling



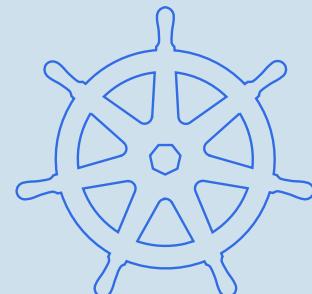
Deadlock
Prevention





Demo

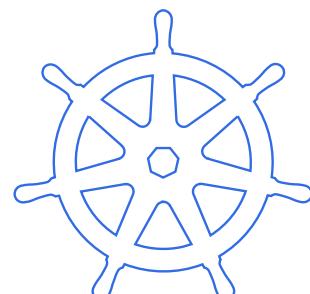
| Pod Anti-Affinity



Taints & Tolerations

Taints & Tolerations

- Taints & tolerations work together to ensure that pods are scheduled onto appropriate nodes
- This mechanism helps to repel specific pods from certain nodes



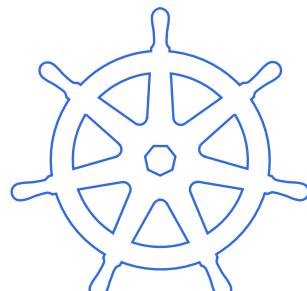
Taints & Tolerations

Taints

- Applied to nodes and allow a node to repel a set of pods
- Pods must have matching tolerations to be allowed on tainted nodes

Tolerations

- Enable pods to be scheduled on nodes with matching taints
- Allow scheduling on both tainted and untainted nodes, but do not guarantee it



Applying Taints & Tolerations

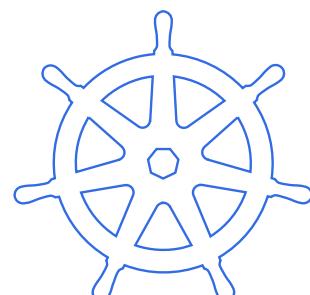
Adding a Taint

effect:

```
kubectl taint nodes node1 key1=value1:NoSchedule
```

Removing a Taint

```
kubectl taint nodes node1 key1=value1:NoSchedule-
```



Applying Taints & Tolerations

Adding a Taint

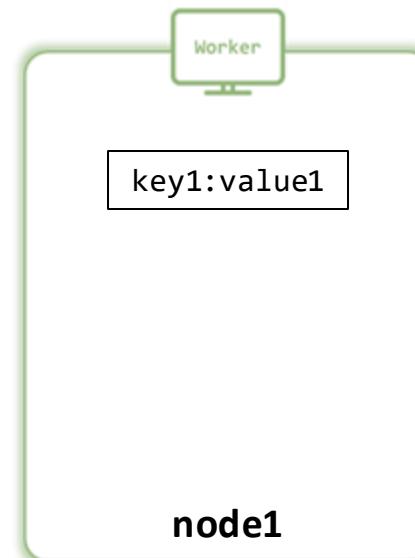
```
kubectl taint nodes node1 key1=value1:NoSchedule
```

Adding Tolerations to Pods

PodSpec:

```
tolerations:  
- key: "key1"  
  operator: "Equal"  
  value: "value1"  
  effect: "NoSchedule"
```

```
tolerations:  
- key: "key1"  
  operator: "Exists"  
  effect: "NoSchedule"
```



Taint Effects

NoSchedule

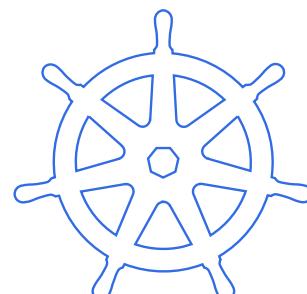
Pods without the taint tolerance won't be scheduled on the node

PreferNoSchedule

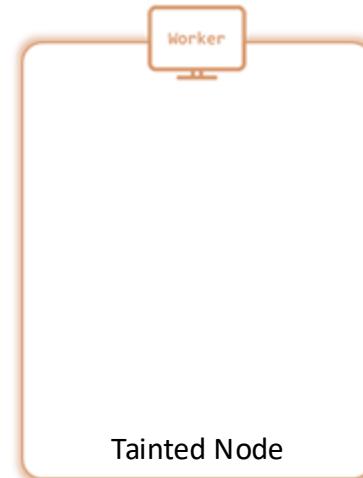
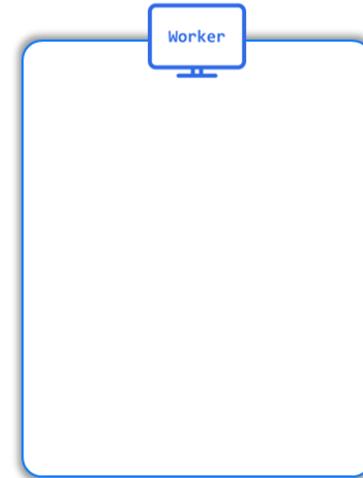
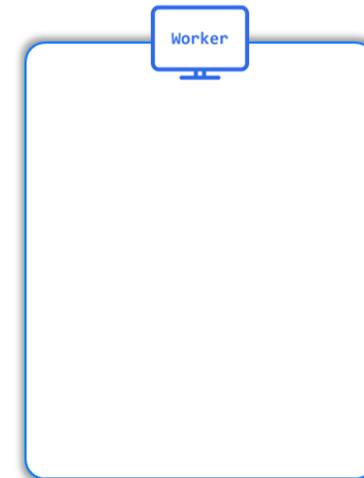
Scheduler attempts to avoid placing non-tolerant pods, but it's not guaranteed

NoExecute

Non-tolerant pods are evicted from the node; tolerant pods remain for a set time (**tolerationSeconds**) before eviction



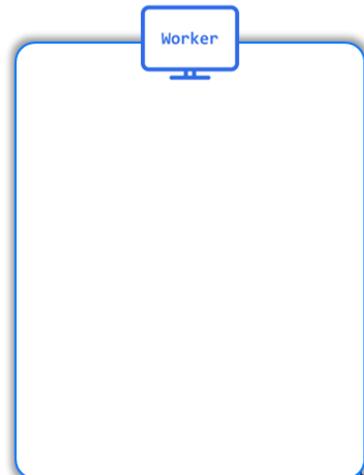
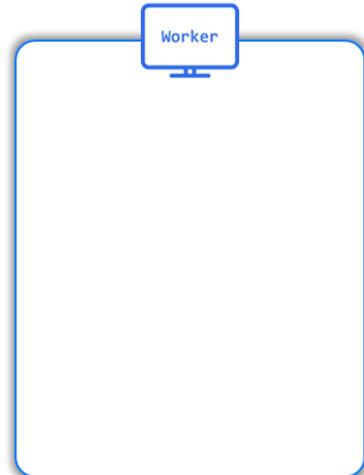
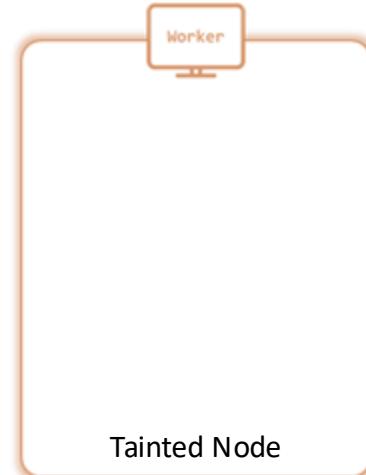
Interaction Between Taints & Toleration



Interaction Between Taints & Toleration

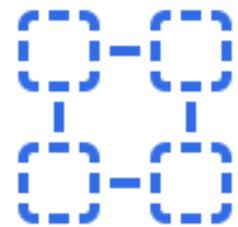


Remained
Unscheduled



Taints & Tolerations

Use-cases



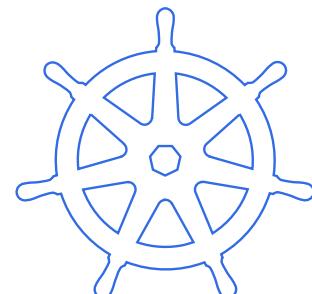
Dedicated
Nodes



Specialized
Hardware

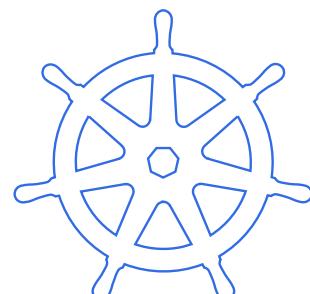


Node
Conditions



Taints & Tolerations

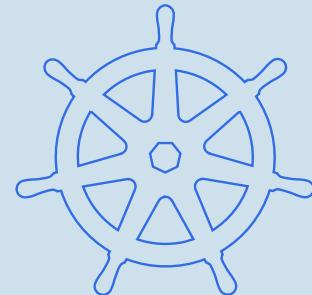
- Taints and tolerations control pod placement on suitable nodes
- It ensures that workloads run where they are most appropriate
- Understanding these features aids in managing pod scheduling and eviction





Demo

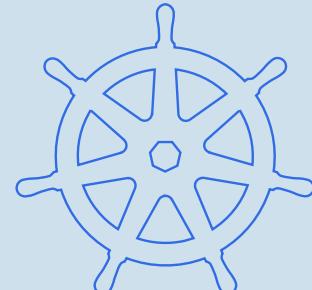
Taints & Tolerations





Demo

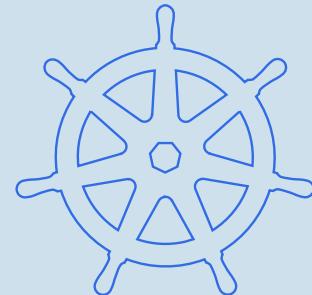
Daemonsets with
Tolerations





Demo

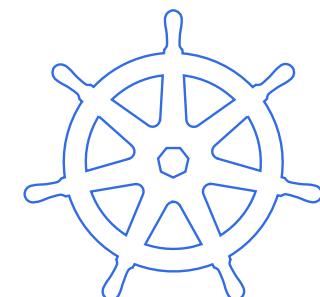
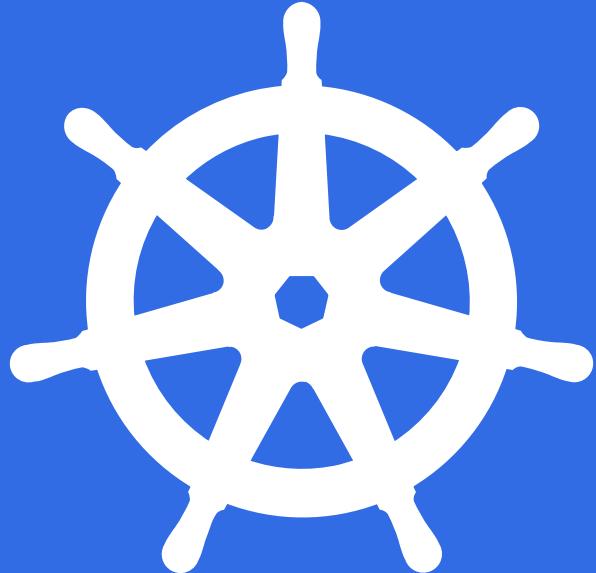
Removing
Taints & Labels



Summary

Summary

- ✓ nodeName
- ✓ nodeSelector
- ✓ Node affinity & anti-affinity
- ✓ Pod affinity & anti-affinity
- ✓ Taints & Tolerations

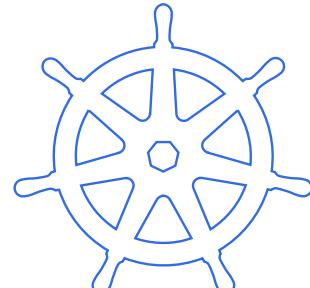


Section: 13.1

Section Overview

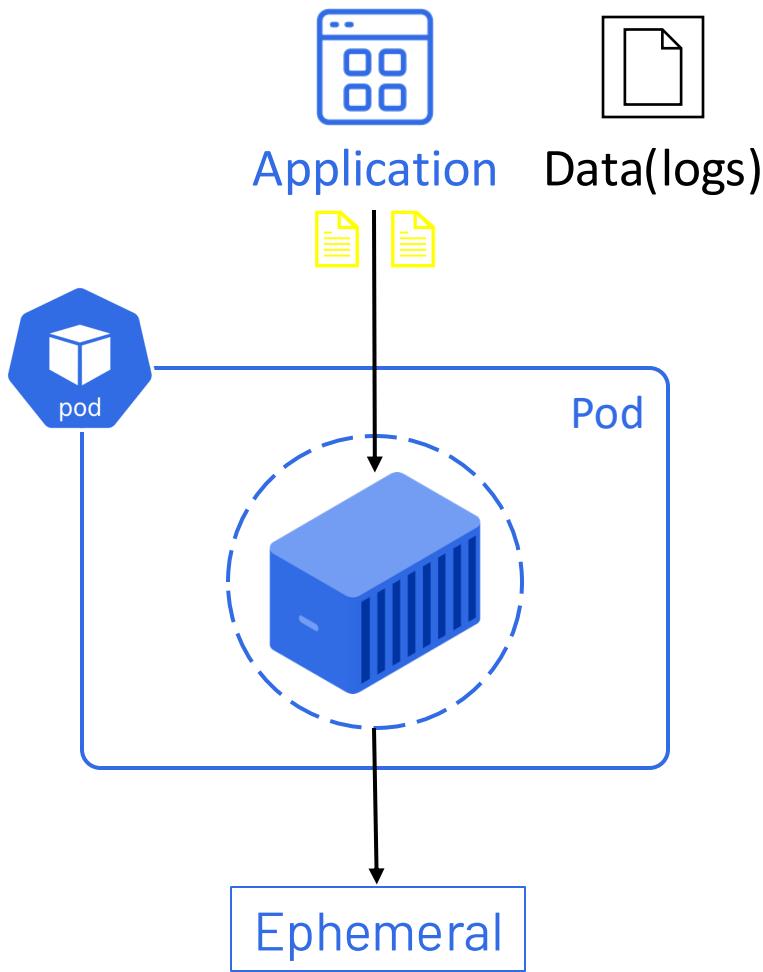
Storage (Part-1)

- └ **Introduction to Volumes**
- └ **Volume Types**
 - **EmptyDir**
 - **HostPath**
 - **ConfigMaps and Secrets**
 - **Projected Volumes**

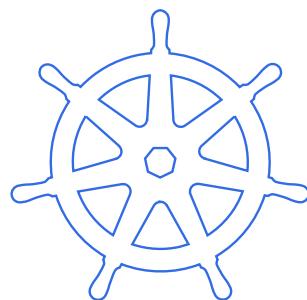


Volumes

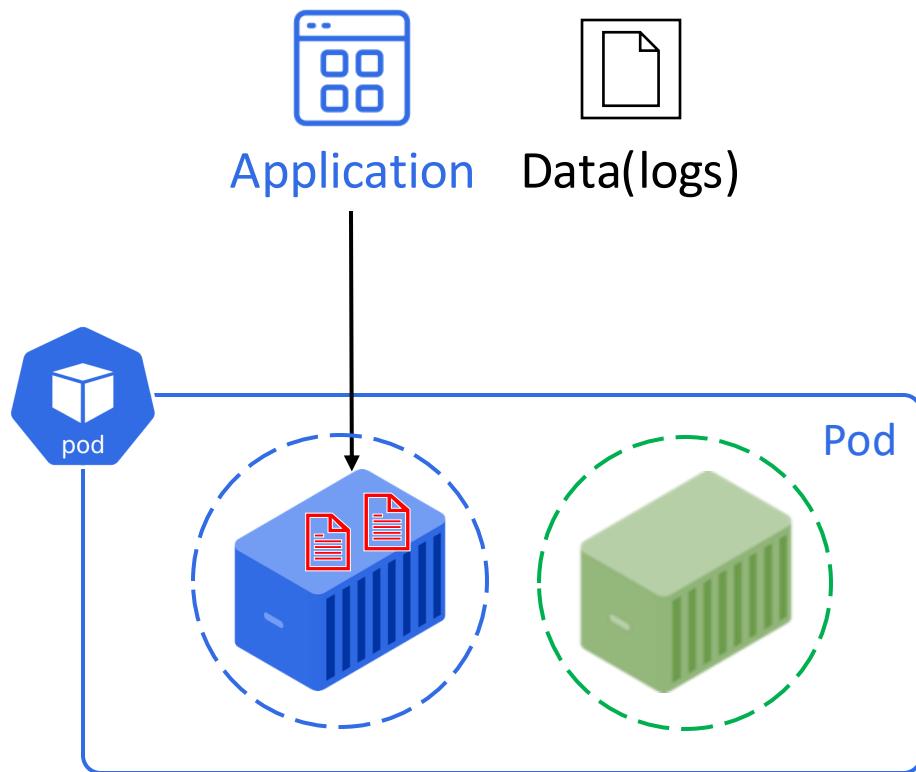
Volumes



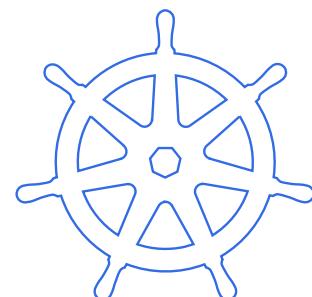
- **Container Crashes**
- **kubelet Restarted Container**



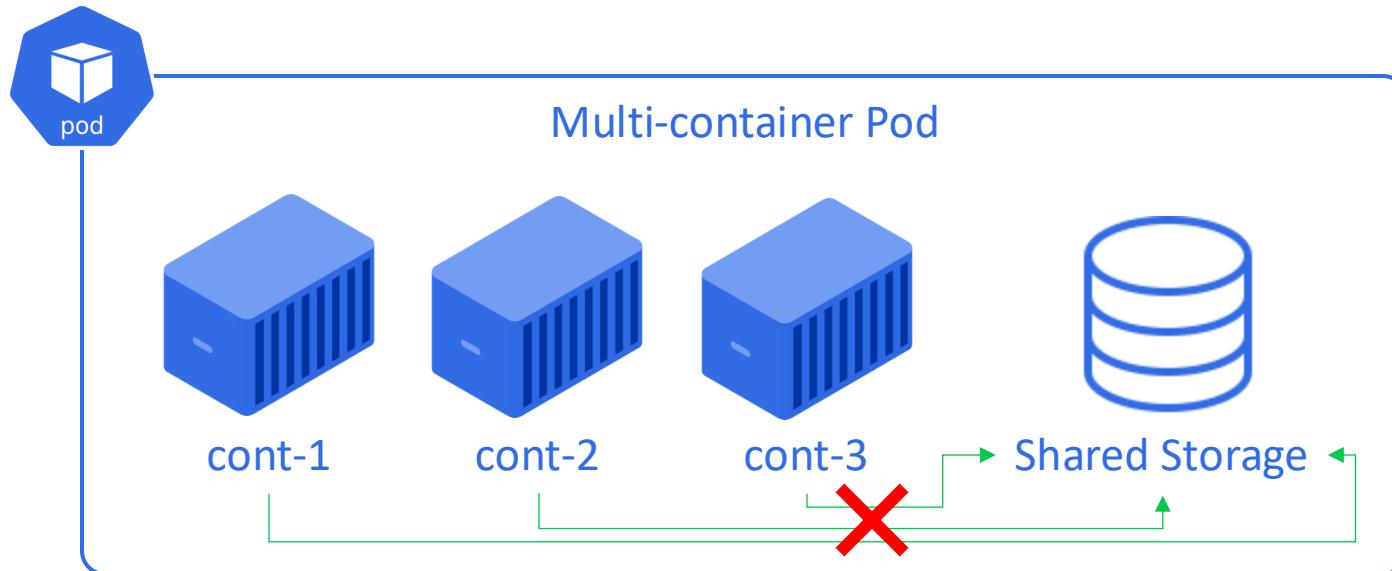
Volumes



- **Container Crashes**
- **kubelet Restarted Container**

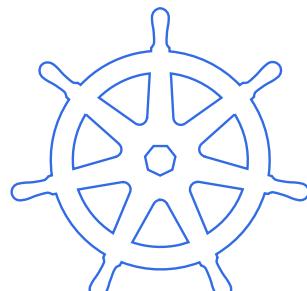


Volumes



- **Container Crashes**
- **kubelet Restarted Container**
- **Shared Storage for Multi-container pod**

Kubernetes Volumes
Abstraction Layer



Volume Types

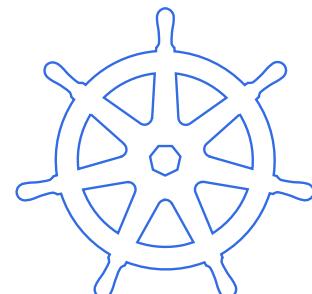
Volume Types



Ephemeral Volume Type

Projected Volume Type

Persistent Volume Type



Volume Types

Ephemeral Volume Type

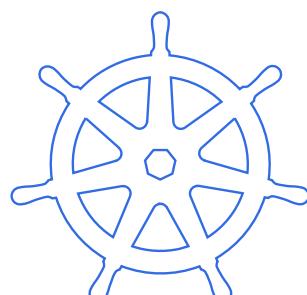
Persistent Volume Type

Projected Volume Type

- You need storage but you don't care about storing data persistently

Types of Ephemeral Volumes

- emptyDir
- ConfigMaps
- secret



Volume Types

Ephemeral Volume Type

Persistent Volume Type

Projected Volume Type

- You need storage but you care about storing data persistently

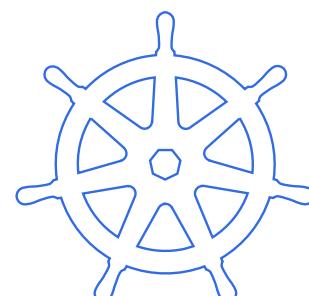
Persistent Volume

Persistent Volume Claim

Storage Classes

Types of Persistent Volumes

- hostpath
- nfs
- iscsi
- fc



Volume Types

Ephemeral Volume Type

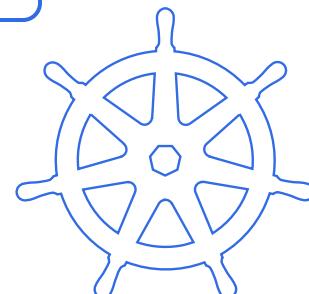
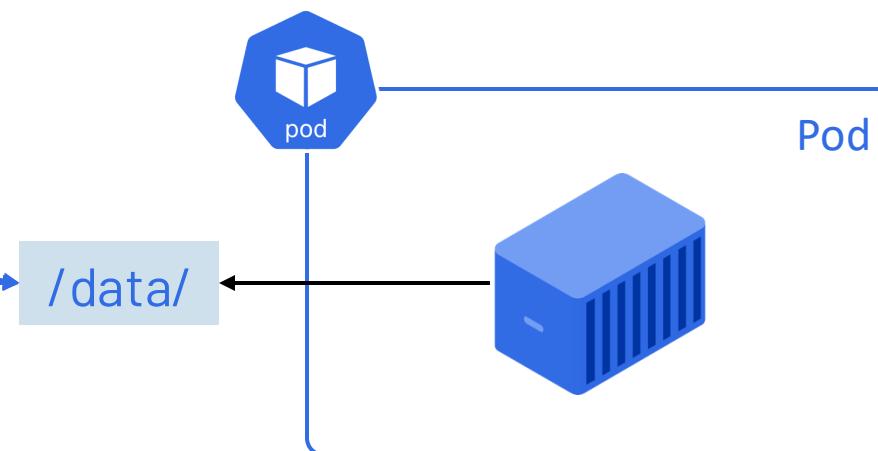
Persistent Volume Type

Projected Volume Type

- All-in-one Volumes

Types of Projected Volumes

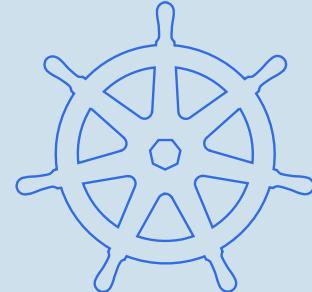
- secret
- ConfigMaps
- serviceAccountToken
- downwardAPI
- clusterTrustBundle





Demo

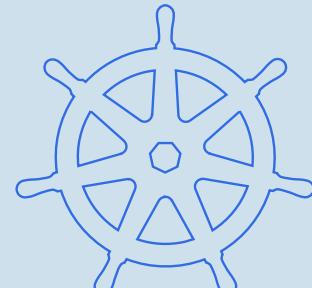
Empty dir Volume Type
(Ephemeral Volumes)





Demo

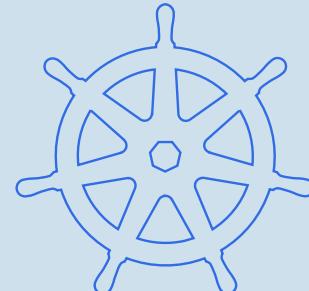
Hostpath Volume Type
(Persistent Volumes)





Demo

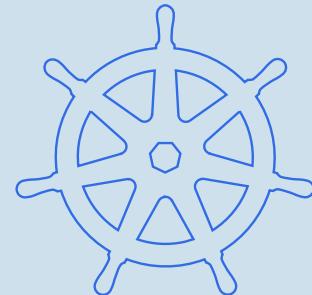
ConfigMaps and
Secrets using Volumes





Demo

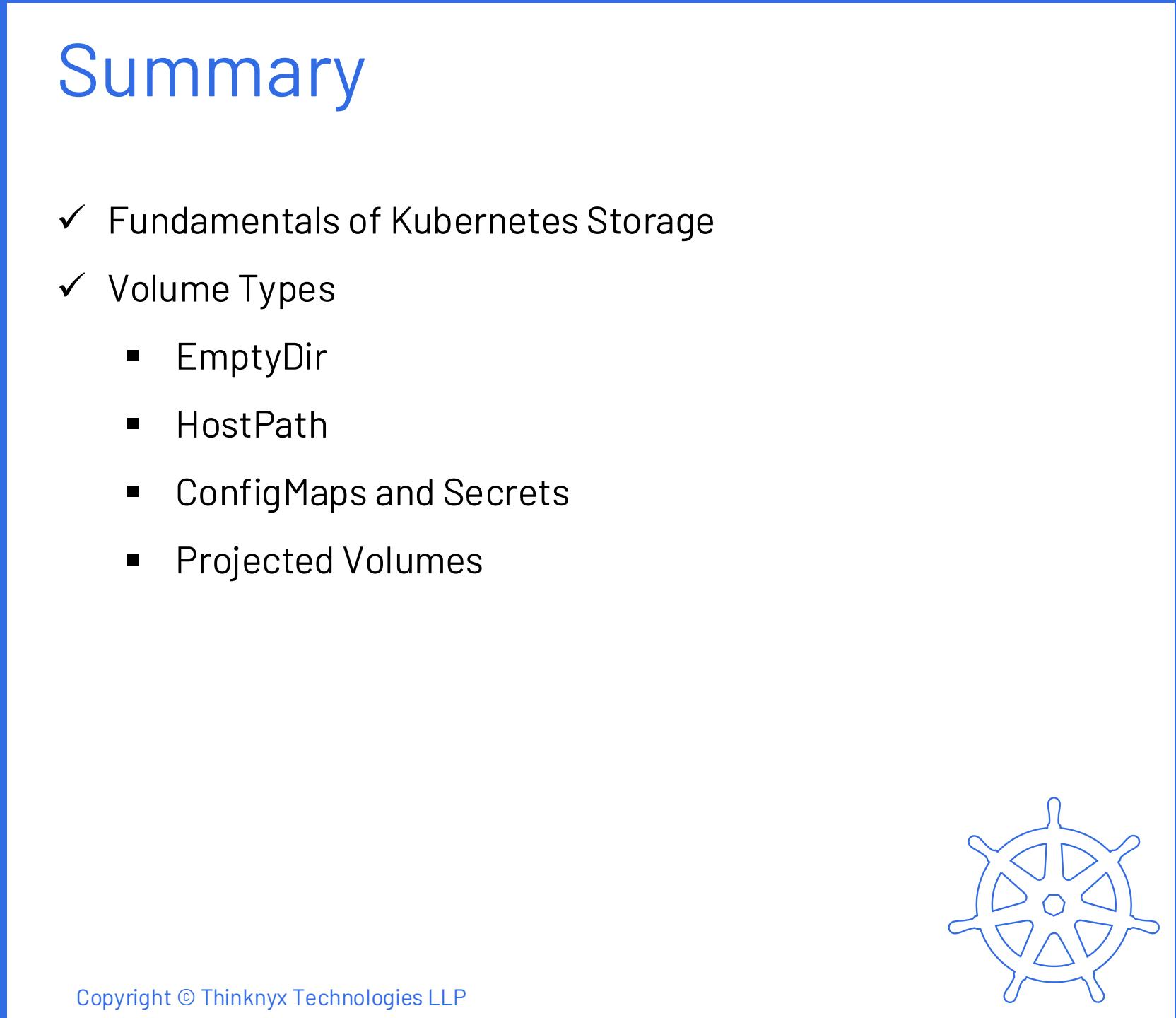
Projected Volumes
(All in one Volumes)



Summary



Summary

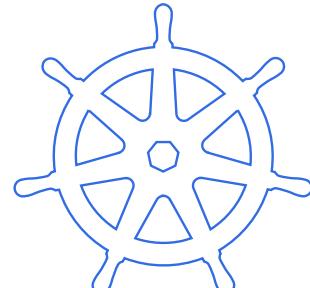
- ✓ Fundamentals of Kubernetes Storage
 - ✓ Volume Types
 - EmptyDir
 - HostPath
 - ConfigMaps and Secrets
 - Projected Volumes
- 

Section: 13.2

Section Overview

Storage

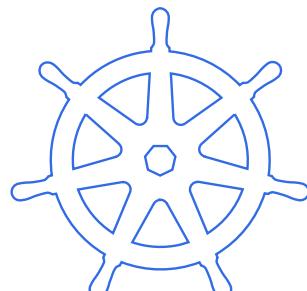
- └ **Persistent Volumes (PVs)**
- └ **Persistent Volume Claims (PVCs)**
- └ **Storage Classes**



Persistent Volume & Persistent Volume Claim

Persistent Volume & Persistent Volume Claim

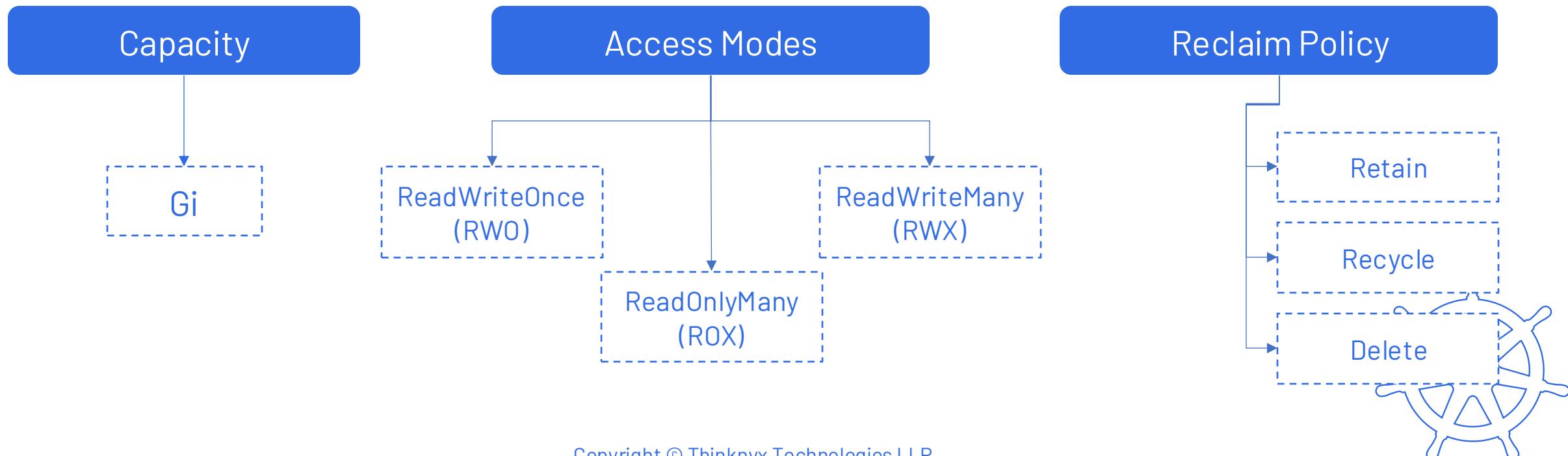
- Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) allow storage management independently of Pods
- They ensure data persists across Pod restarts, rescheduling, and node failures



Persistent Volume (PV)

Persistent Volume (PV)

- A Persistent Volume (PV) is a storage resource managed by Kubernetes, either manually created or automatically provisioned
- PVs abstract the underlying storage infrastructure, enabling Kubernetes to work with different storage solutions
- PVs can integrate with various storage types, including cloud provider storage, NAS, or local storage



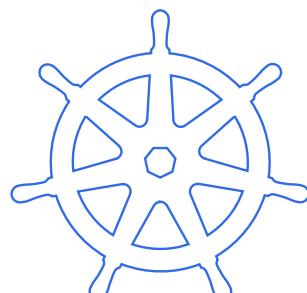
Persistent Volume (PV)

Statically Provisioned

PVs are manually created and defined by administrators in advance

Dynamically Provisioned

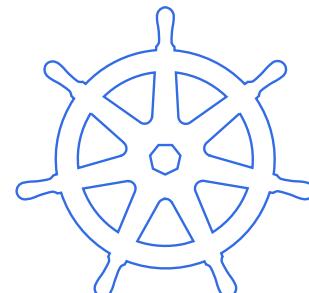
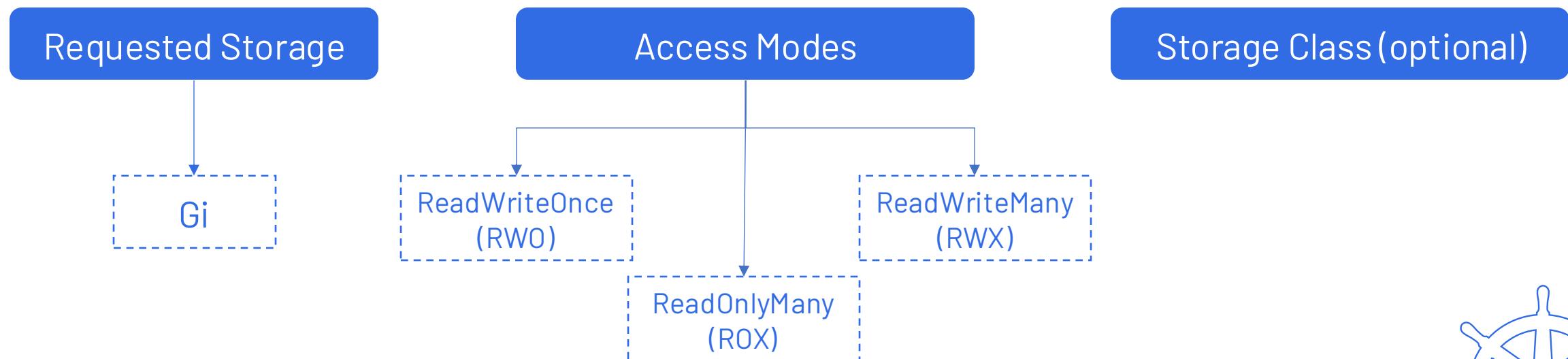
PVs are automatically created based on StorageClass and PVC requests



Persistent Volume Claim (PVC)

Persistent Volume Claim (PVC)

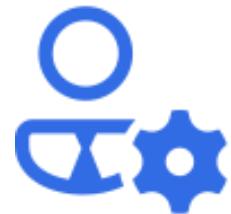
- A Persistent Volume Claim (PVC) is a request for storage by an application running in Kubernetes
- Kubernetes attempts to find and bind a matching Persistent Volume (PV) to fulfill the PVC's requirements



PV & PVC Workflow

PV & PVC Workflow

Provisioning the PV



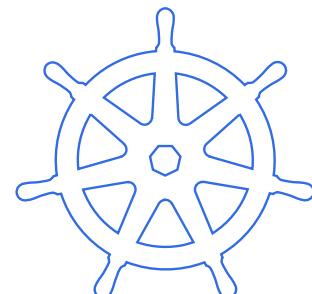
Creating a PVC



Binding



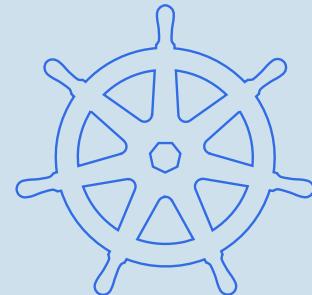
Using the Volume





Demo

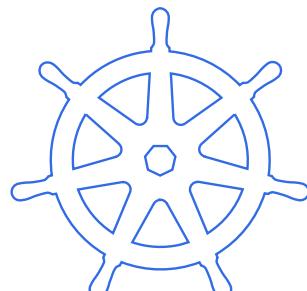
PV and PVC
with NFS



Dynamic Volumes using Storage Classes

Dynamic Volumes using Storage Classes

- Storage Classes enable dynamic provisioning of Persistent Volumes (PVs) on demand
- Storage Classes simplify storage management by eliminating the need for pre-existing PVs
- Storage Classes define storage properties like performance, availability, and cost to match application needs



What is a Storage Class?

What is a Storage Class?

- A Storage Class in Kubernetes is an object that defines a "class" of storage based on the underlying storage provider
 - ✓ Type
 - ✓ Provisioner
 - ✓ Parameters
 - ✓ Reclaim Policy

Storage
Class

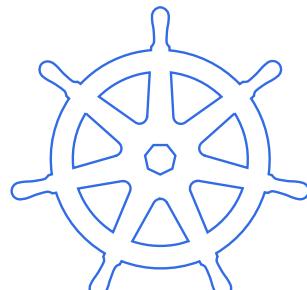
abstract the details of the
underlying storage
infrastructure

PVs

create PVs
dynamically

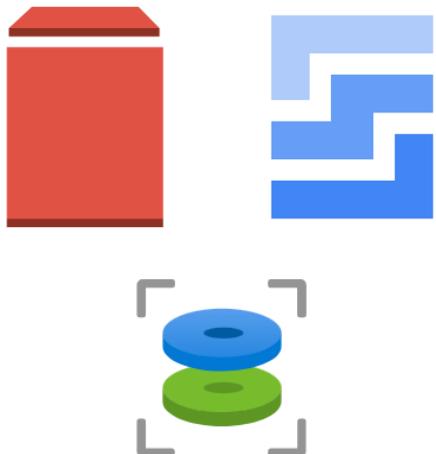
PVC

applications only need to
specify their storage
requirements through PVC



What is a Storage Class?

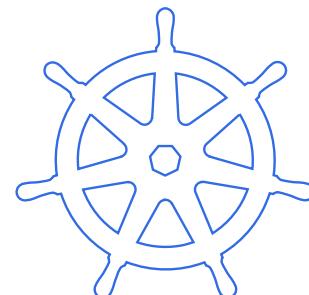
Provisioner



Parameters

- ✓ Storage performance settings
- ✓ Disk types
- ✓ Other storage-specific characteristics

Reclaim Policy



How Dynamic Provisioning Works

How Dynamic Provisioning Works

Creating a Storage Class



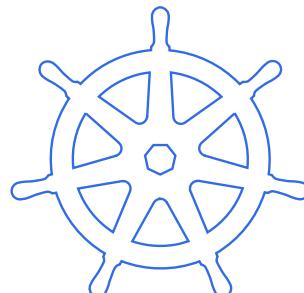
Requesting Storage through PVC



Automatic PV Creation & Binding



Using the Storage

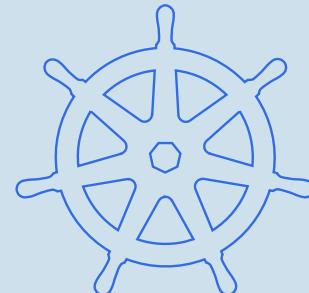


storageClassName



Demo

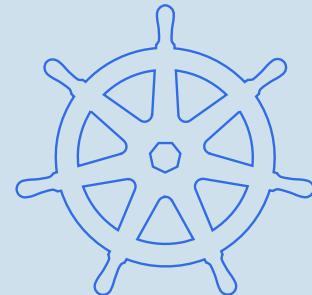
NFS Provisioner
Setup





Demo

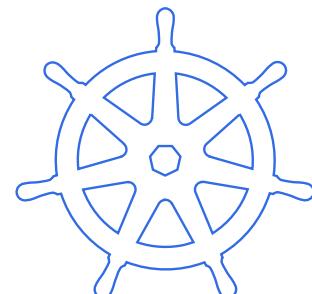
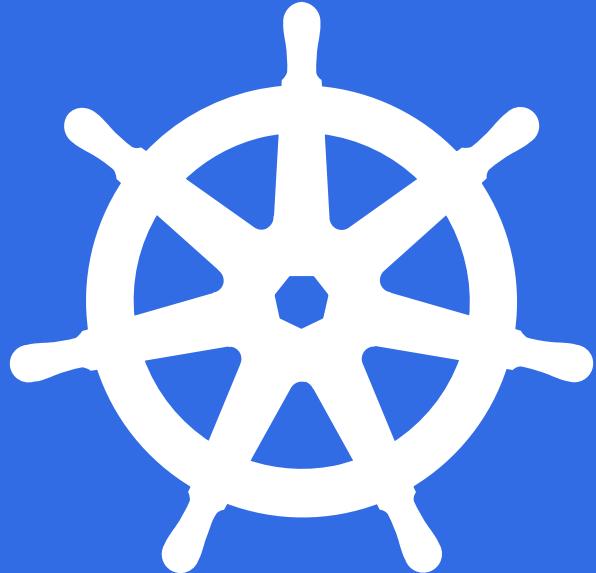
Storage using
NFS



Summary

Summary

- ✓ Persistent Volumes (PVs)
- ✓ Persistent Volume Claims (PVCs)
- ✓ Storage Classes in Kubernetes
- ✓ Demonstrated Requesting Persistent Storage Resources
- ✓ Hands-on Insight into Storage Allocation

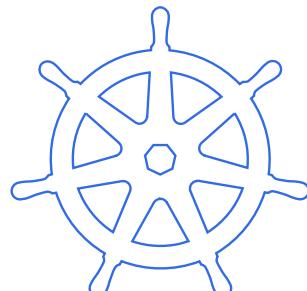


Section: 13.3

Section Overview

Storage

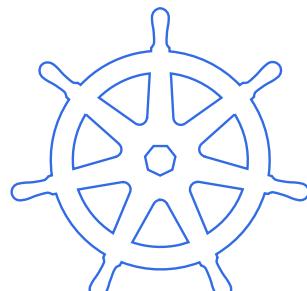
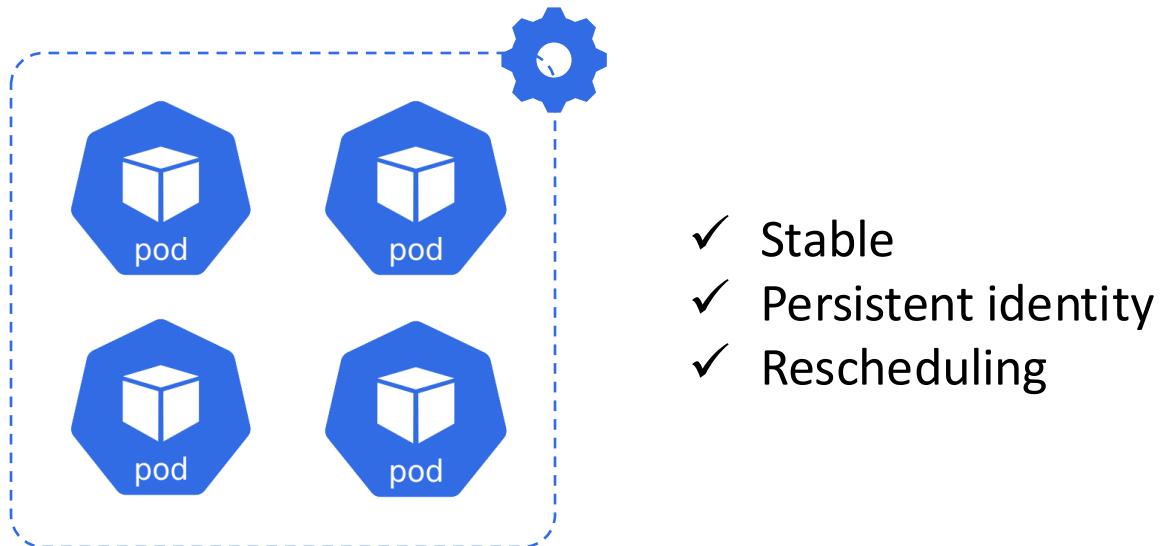
- ↳ **StatefulSets**
- ↳ **How StatefulSets help manage stateful applications**



StatefulSets

StatefulSets

- StatefulSet in Kubernetes is a workload API object used to manage stateful applications
- StatefulSets ensure that stateful applications maintain a consistent identity for each pod, even after restarts or failures



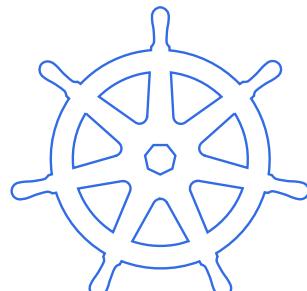
StatefulSets

- StatefulSets manage the deployment and scaling of Pods while ensuring proper ordering and uniqueness
- StatefulSets assign unique, persistent identities to each Pod
- These unique identities ensure Pods are not interchangeable and remain consistent, even when rescheduled

Persistent Storage

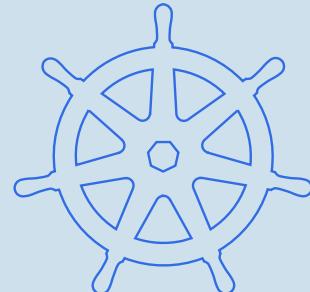
StatefulSets

PersistentVolumes (PVs)





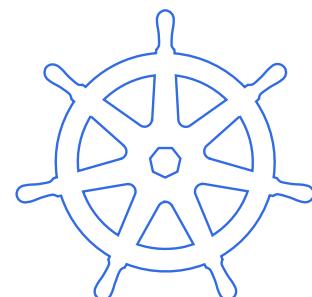
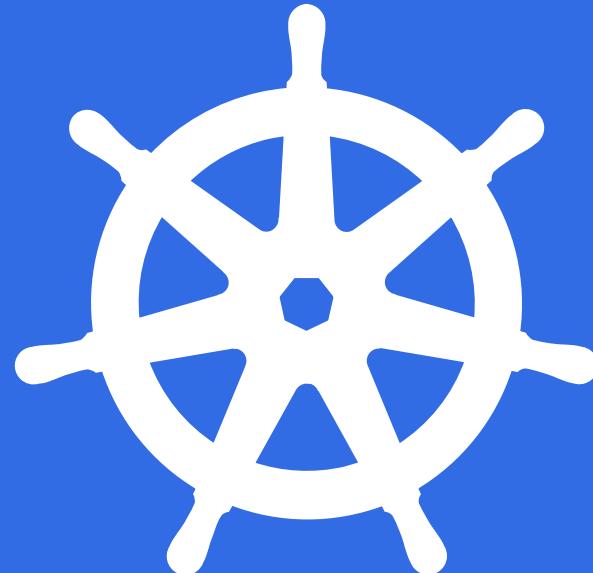
Demo | StatefulSets



Summary

Summary

- ✓ How StatefulSets manage stateful applications

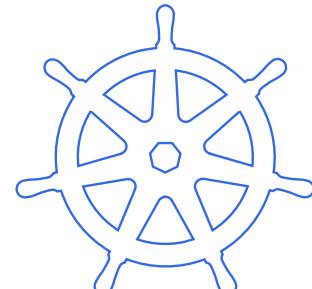


Section: 14

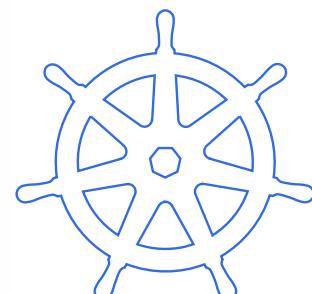
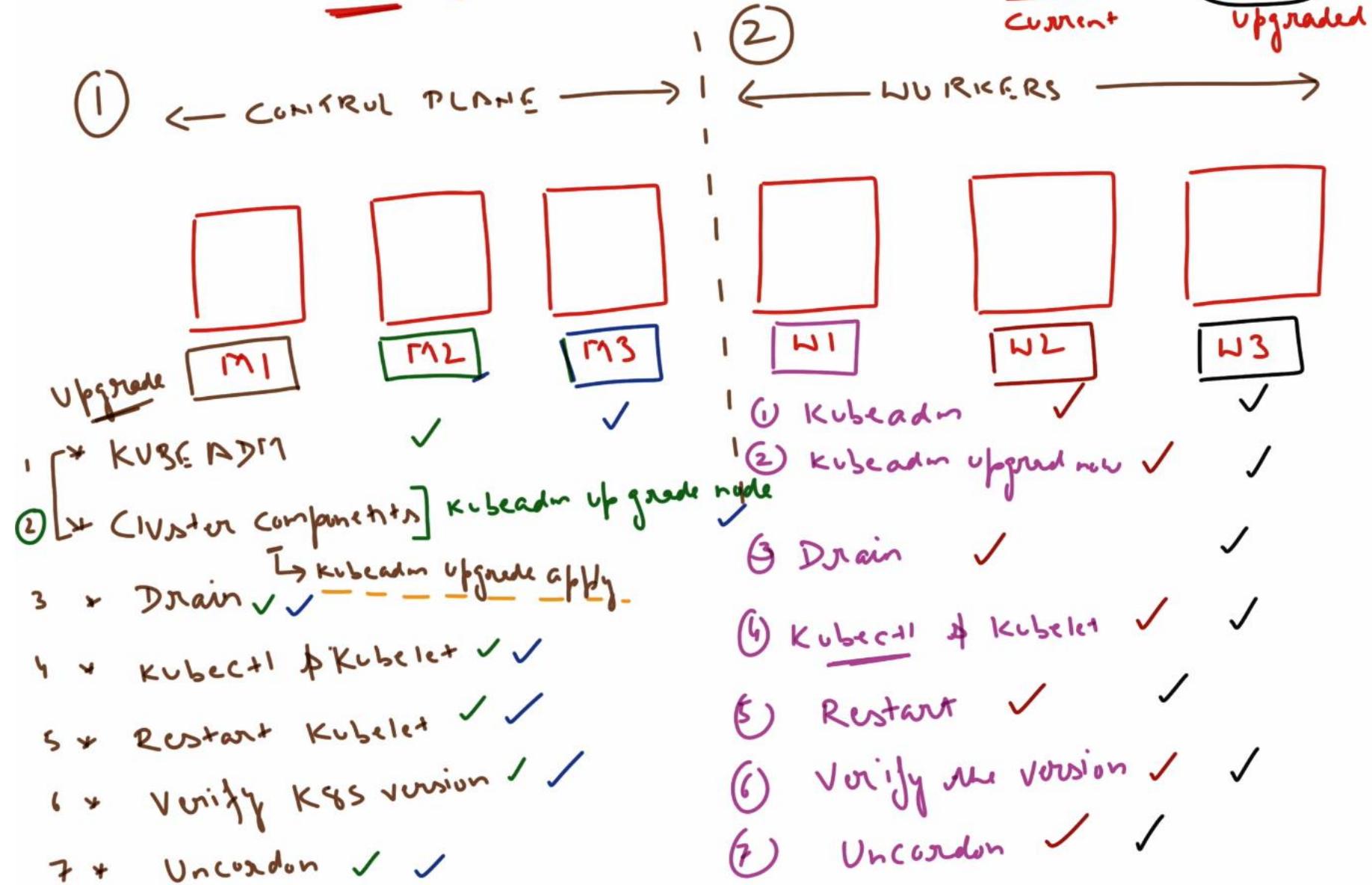
Cluster Upgrade

Section Overview

- **Kubernetes cluster upgrade**
 - Control plane upgrade
 - Worker upgrade



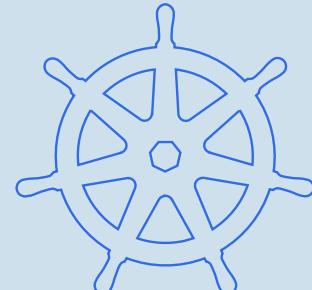
→ Backups (VM, etc)





Demo

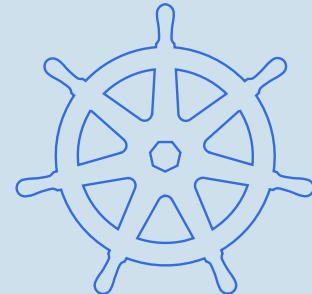
Control Plane Upgrade





Demo

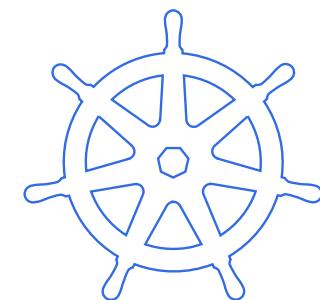
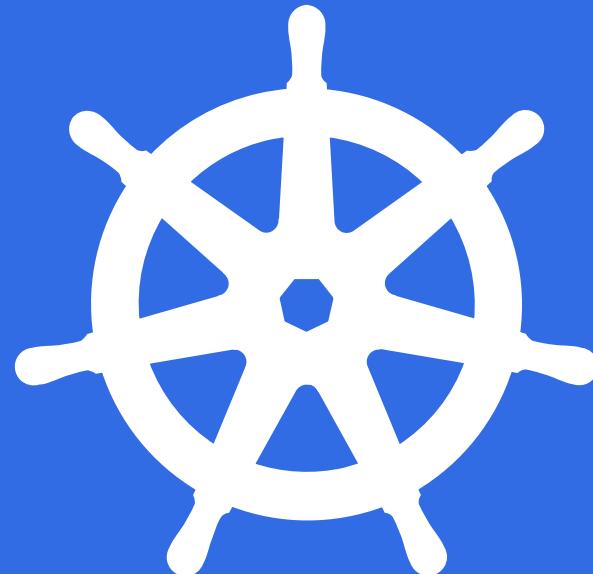
Worker Upgrades



Summary

Summary

- ✓ Kubernetes cluster upgrade

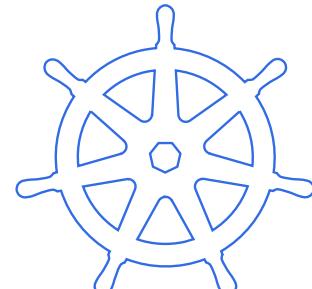


Section: 15

RBAC in Kubernetes

Section Overview

- ↳ RBAC (Role-Based Access Control)



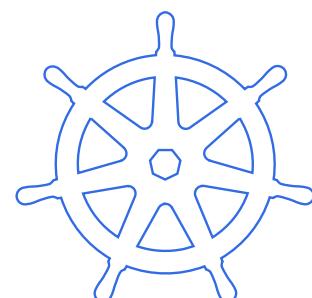
Role-Based Access Control

Roles

Role Bindings

Cluster Roles

Cluster Role Bindings



Role-Based Access Control

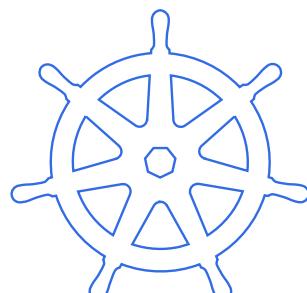
Roles

Role Bindings

Cluster Roles

Cluster Role Bindings

Roles define permissions within a specific namespace, allowing actions like creating, updating, or deleting resources, but only within that namespace



Role-Based Access Control

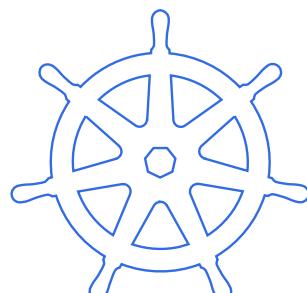
Roles

Role Bindings

Cluster Roles

Cluster Role Bindings

Role Bindings connect a Role to a user, group, or service account, allowing them to perform certain actions in a specific namespace



Role-Based Access Control

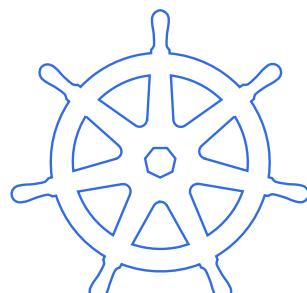
Roles

Role Bindings

Cluster Roles

Cluster Role Bindings

Cluster Roles provide permissions that apply across the entire system, allowing access to multiple namespaces for broader tasks like managing nodes or reading resources



Role-Based Access Control

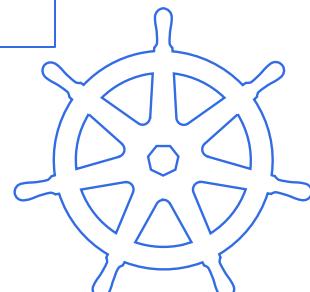
Roles

Role Bindings

Cluster Roles

Cluster Role Bindings

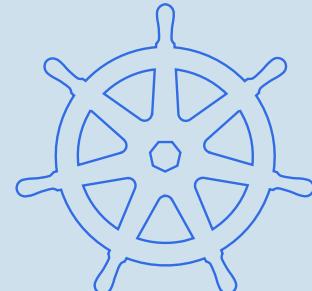
Cluster Role Bindings link a Cluster Role to users, groups, or service accounts, granting permissions that apply across the entire Kubernetes cluster, rather than just one namespace





Demo

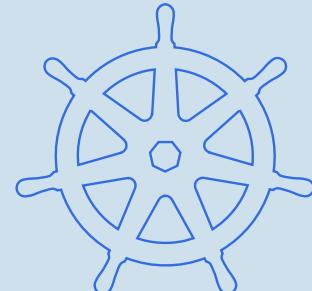
Authentication
(User Creation)





Demo

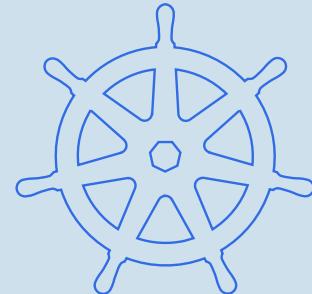
Role & RoleBindings





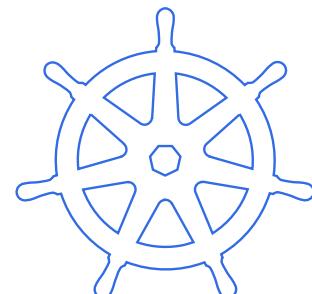
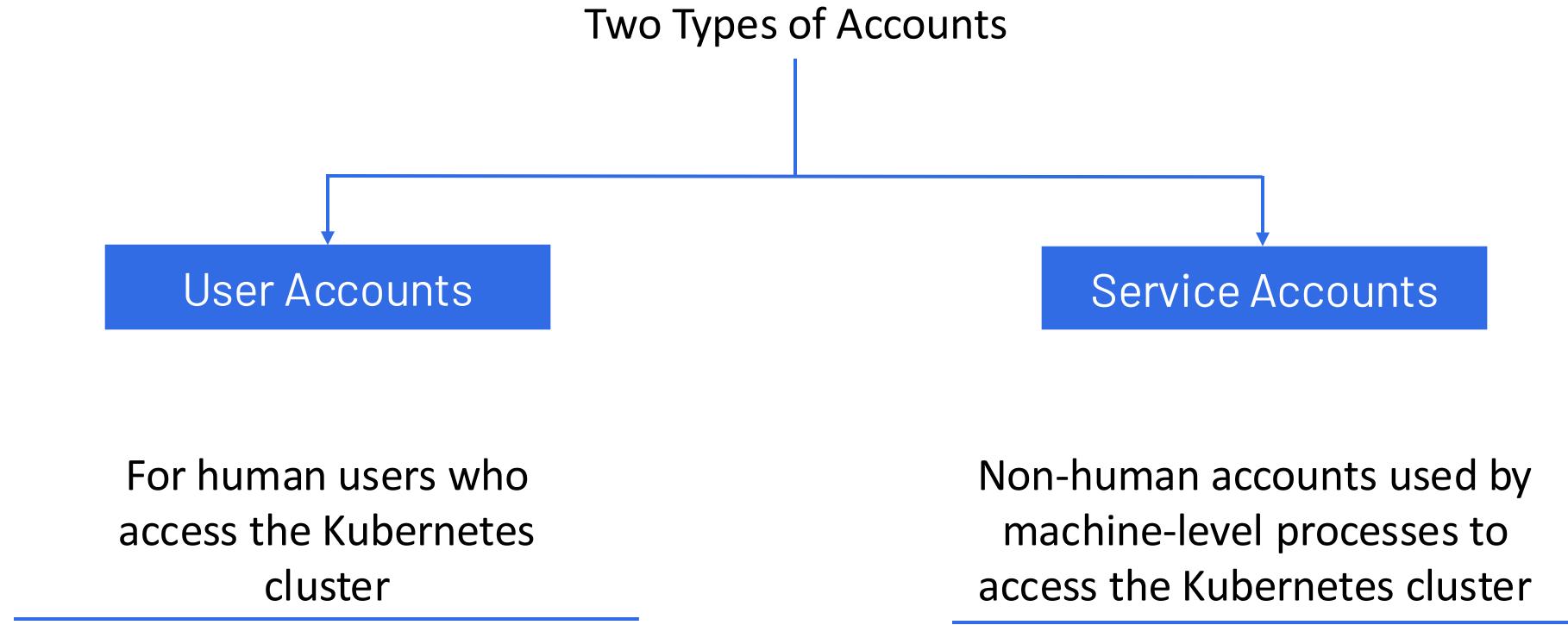
Demo

ClusterRole &
ClusterRoleBindings



Service Accounts

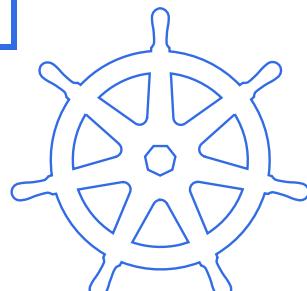
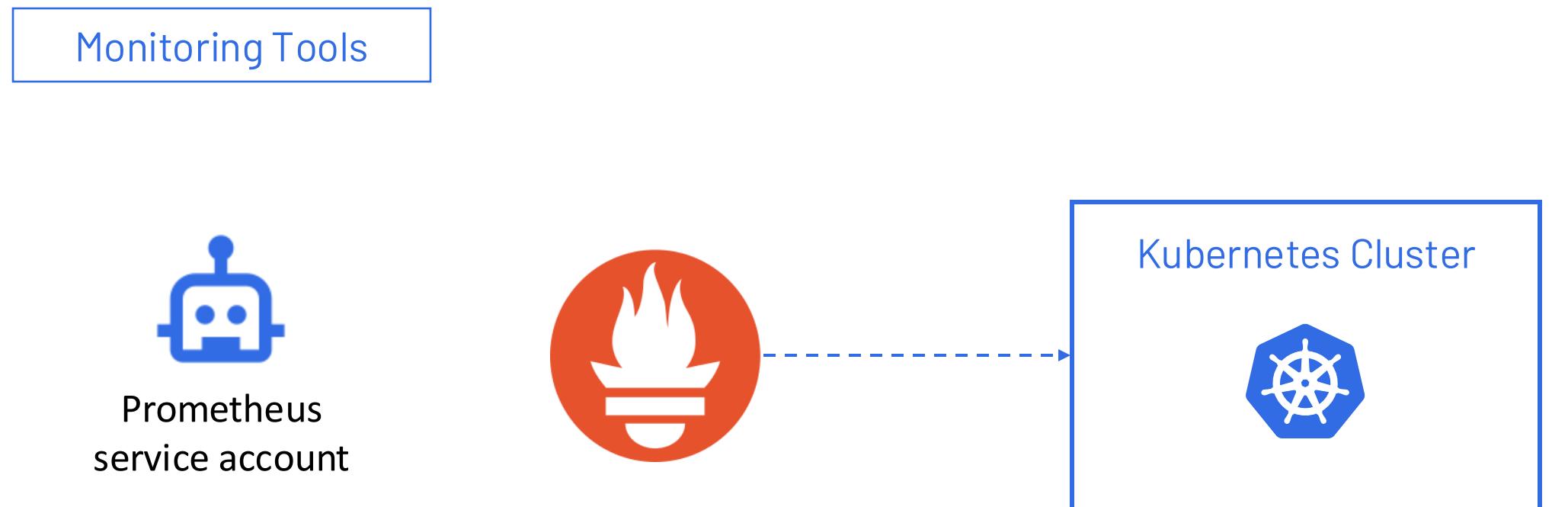
Service Accounts



Why Service Accounts?

Why Service Accounts?

If two applications in the cluster need to communicate, they use service accounts instead of user accounts



Why Service Accounts?

If two applications in the cluster need to communicate, they use service accounts instead of user accounts

Monitoring Tools

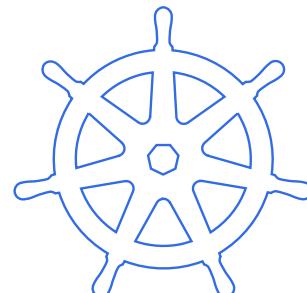
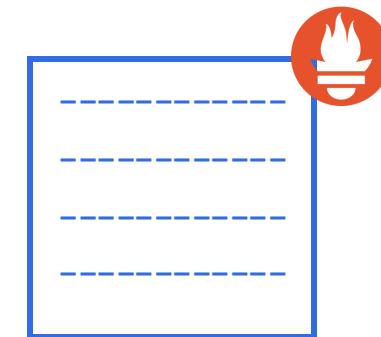


Prometheus
service account

authenticate

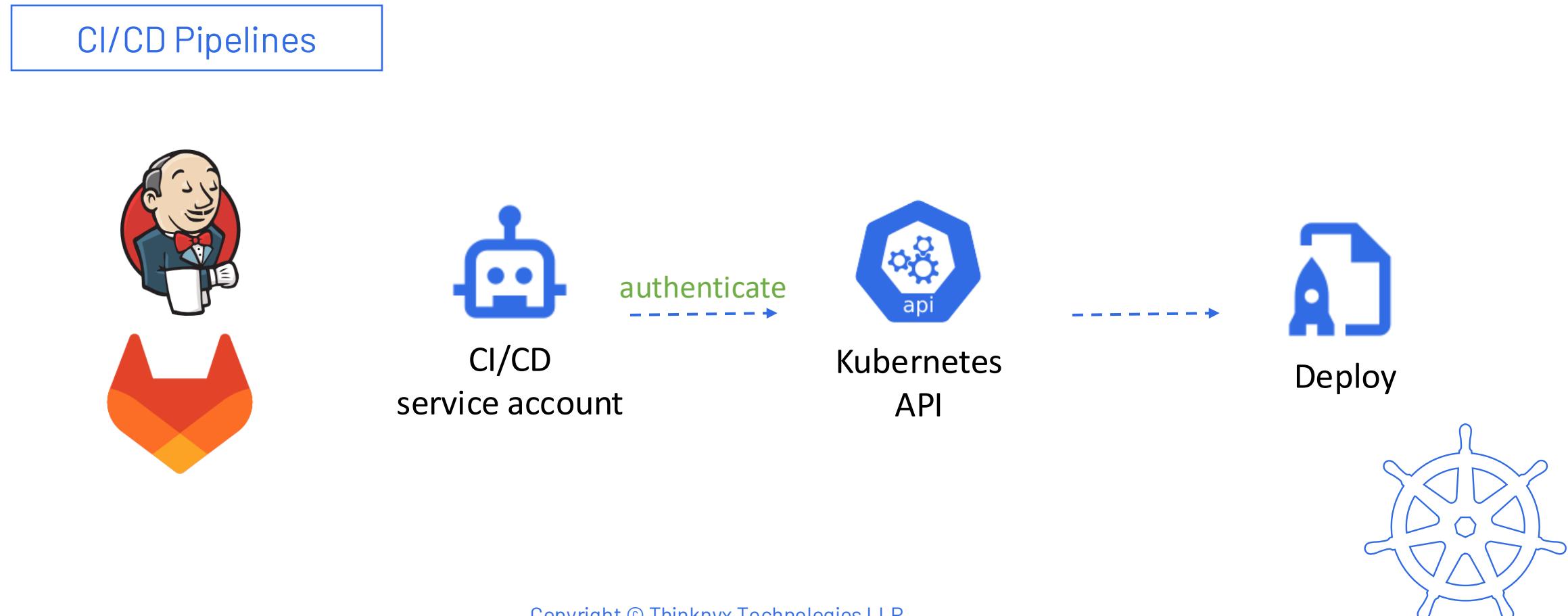


Kubernetes
API



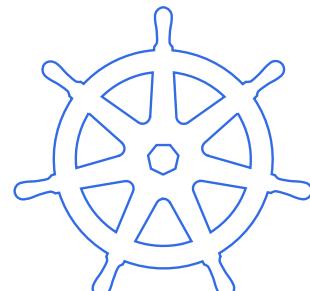
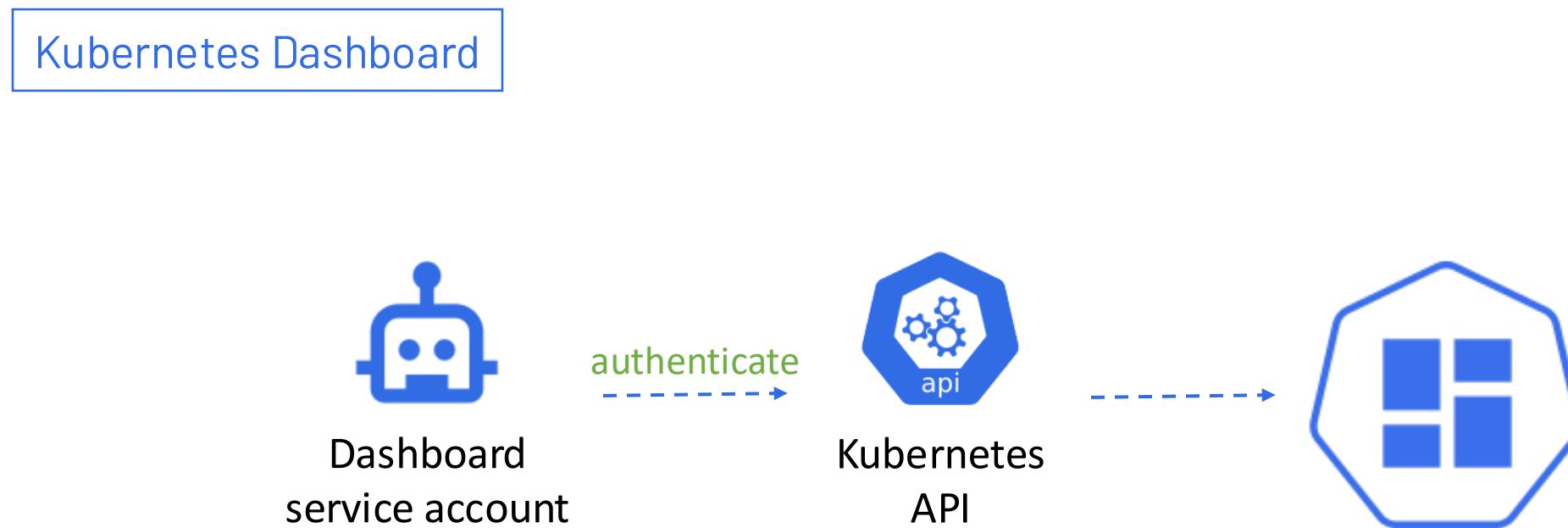
Why Service Accounts?

If two applications in the cluster need to communicate, they use service accounts instead of user accounts



Why Service Accounts?

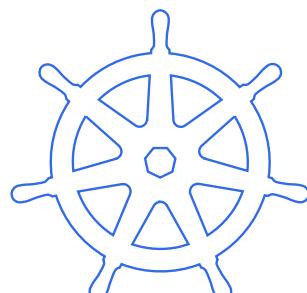
If two applications in the cluster need to communicate, they use service accounts instead of user accounts



Why Service Accounts?

- Most Kubernetes applications use service accounts
- Kubernetes automatically assigns it a **default service account** when you create a regular pod
- Pods cannot be created without a service account, ensuring compliance with Kubernetes internal requirements

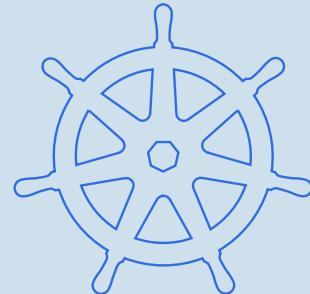
Note: Service accounts are namespace-specific





Demo

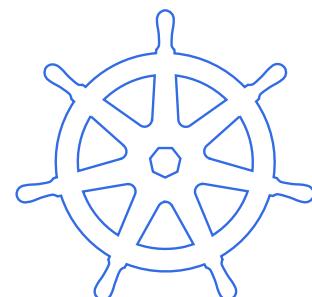
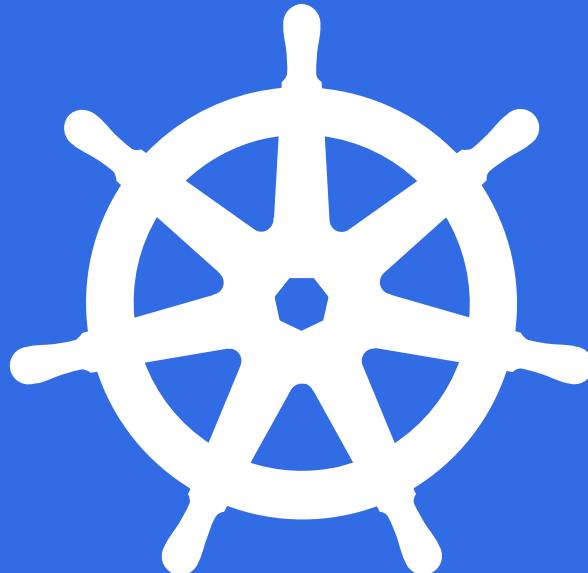
Service Accounts



Summary

Summary

- ✓ Role-Based Access Control (RBAC)
- ✓ Service Accounts

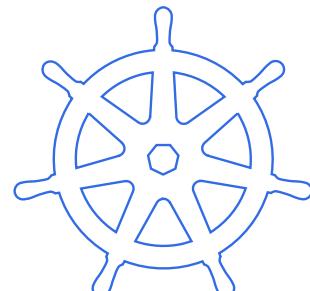


Section: 16

Section Overview

Ingress

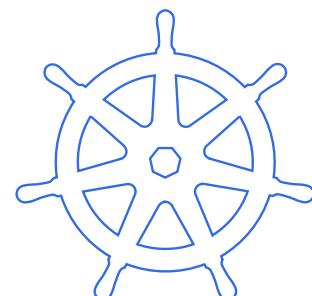
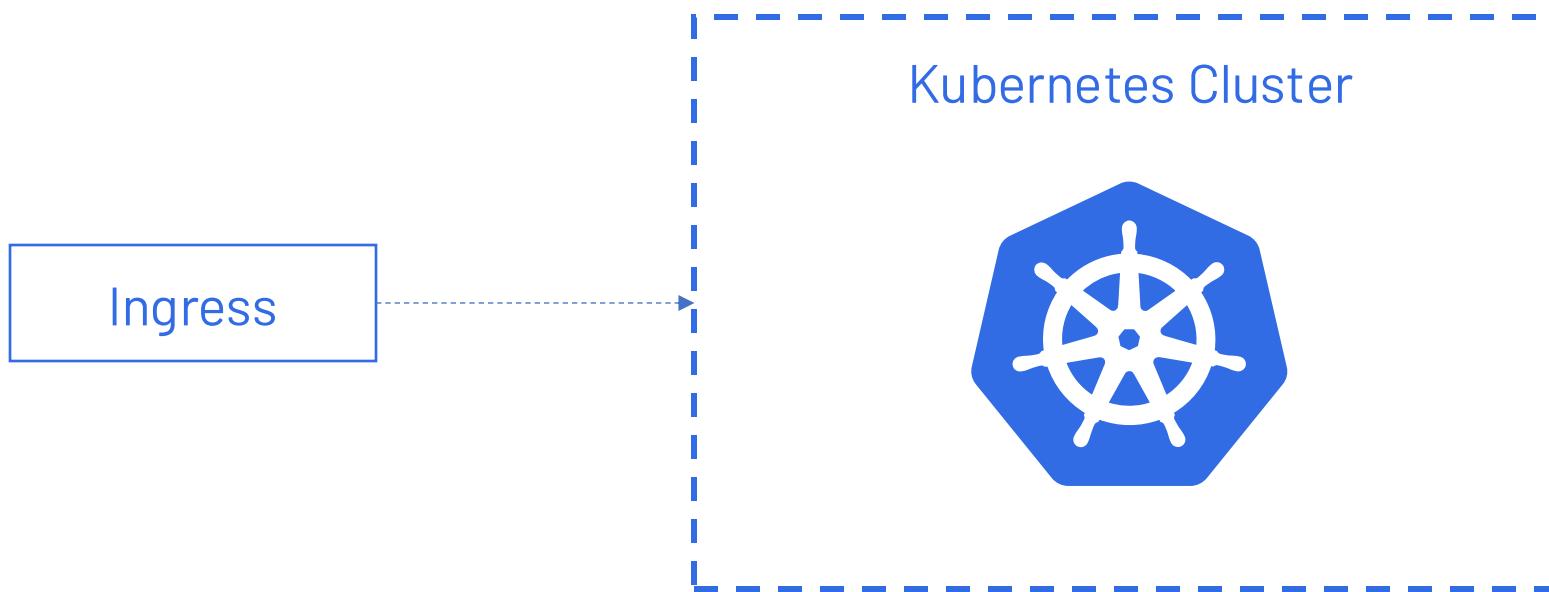
- ↳ **Traffic Routing with Ingress**
- ↳ **Centralized Access Management**



Ingress

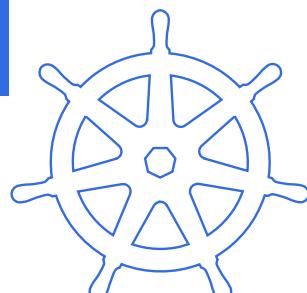
Ingress

- Ingress is a Kubernetes resource that manages external access to services within a cluster
- Acts as a router, defining rules to control how external users can access services running inside the cluster



Ingress

- Ingress defines rules for routing HTTP and HTTPS traffic to services within the cluster



Ingress

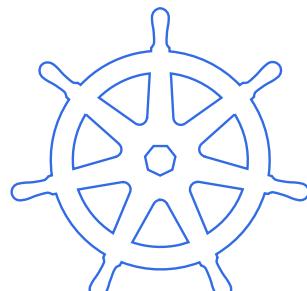
Components of Ingress

Defines routing rules to direct incoming requests to backend services

Ingress Resource

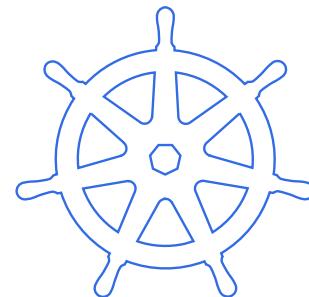
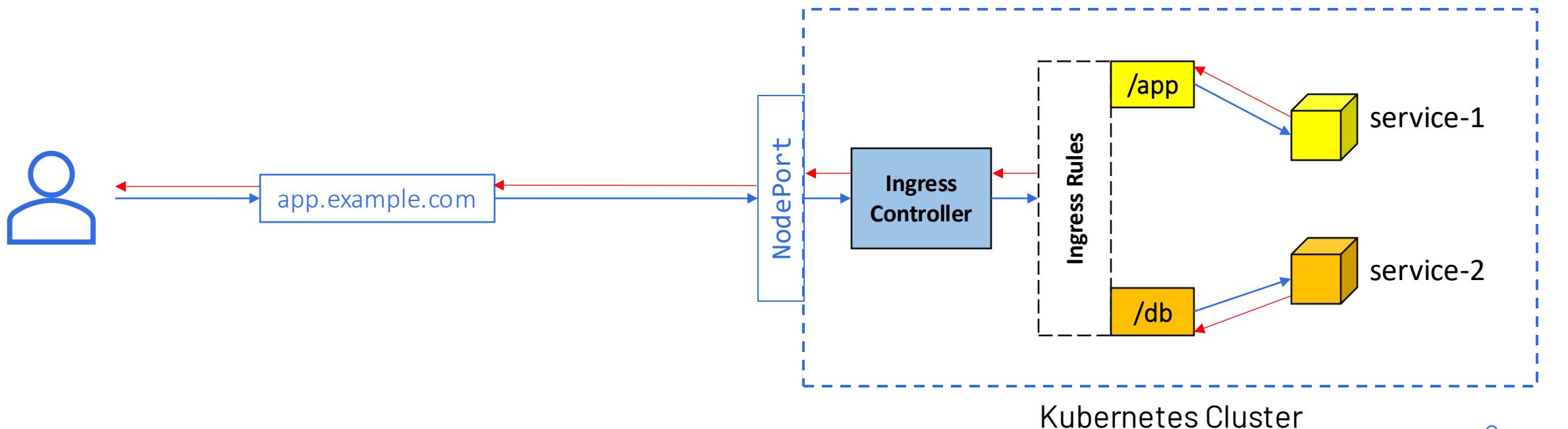
Ingress Controller

Enforces Ingress rules, but Kubernetes lacks a built-in Ingress controller



Ingress

Flow of Traffic in Ingress



```
root@master:~# cat 28_IngressRule_Hostbased.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
  name: ingressrule
spec:
  ingressClassName: nginx
  rules:
  - host: demo.example.com
    http:
      paths:
      - backend:
          service:
            name: pageonesvc
            port:
              number: 80
            path: /webpageone
            pathType: Exact
          - backend:
              service:
                name: pagetwosvc
                port:
                  number: 80
                path: /webpagetwo
                pathType: Exact
```



Ingress

Type of Ingress Service

Single-Service Ingress

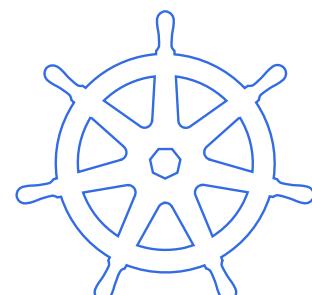
Exposes one service to external traffic

Multiple-Host Ingress

Supports multiple hostnames in one Ingress

TLS/SSL Ingress

Handles HTTPS by terminating SSL connections with a TLS secret

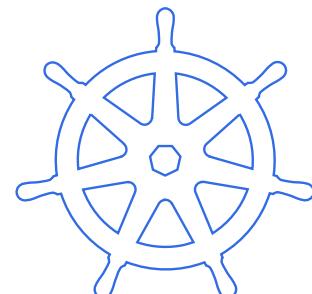


Ingress Controllers

NGINX



HAPROXY
COMMUNITY EDITION



Ingress

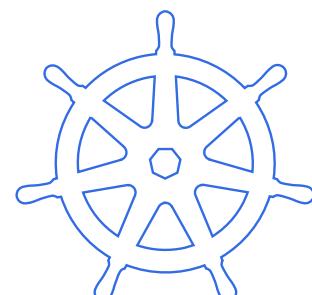
Annotations for Customization

`nginx.ingress.kubernetes.io/ssl-redirect`

Force HTTP-to-HTTPS redirection

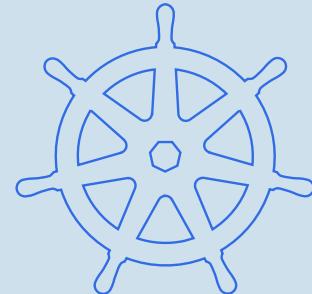
`nginx.ingress.kubernetes.io/rewrite-target`

Rewrite the incoming request's URL





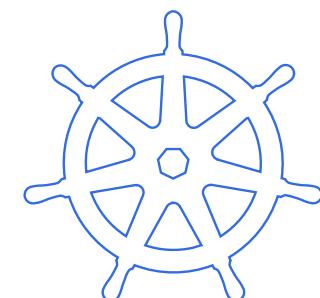
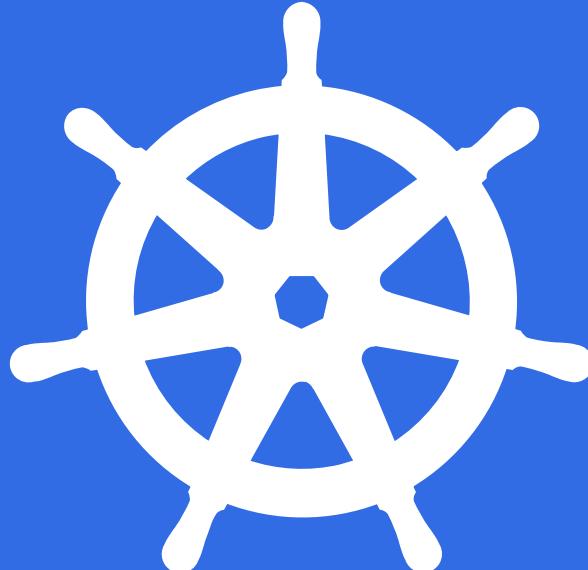
Demo | Ingress



Summary

Summary

- ✓ Traffic Routing with Ingress
- ✓ Centralized Access Management

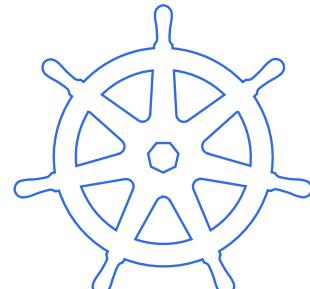


Section: 17

Network Policies

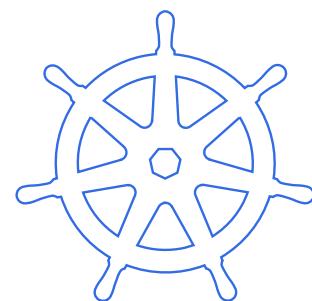
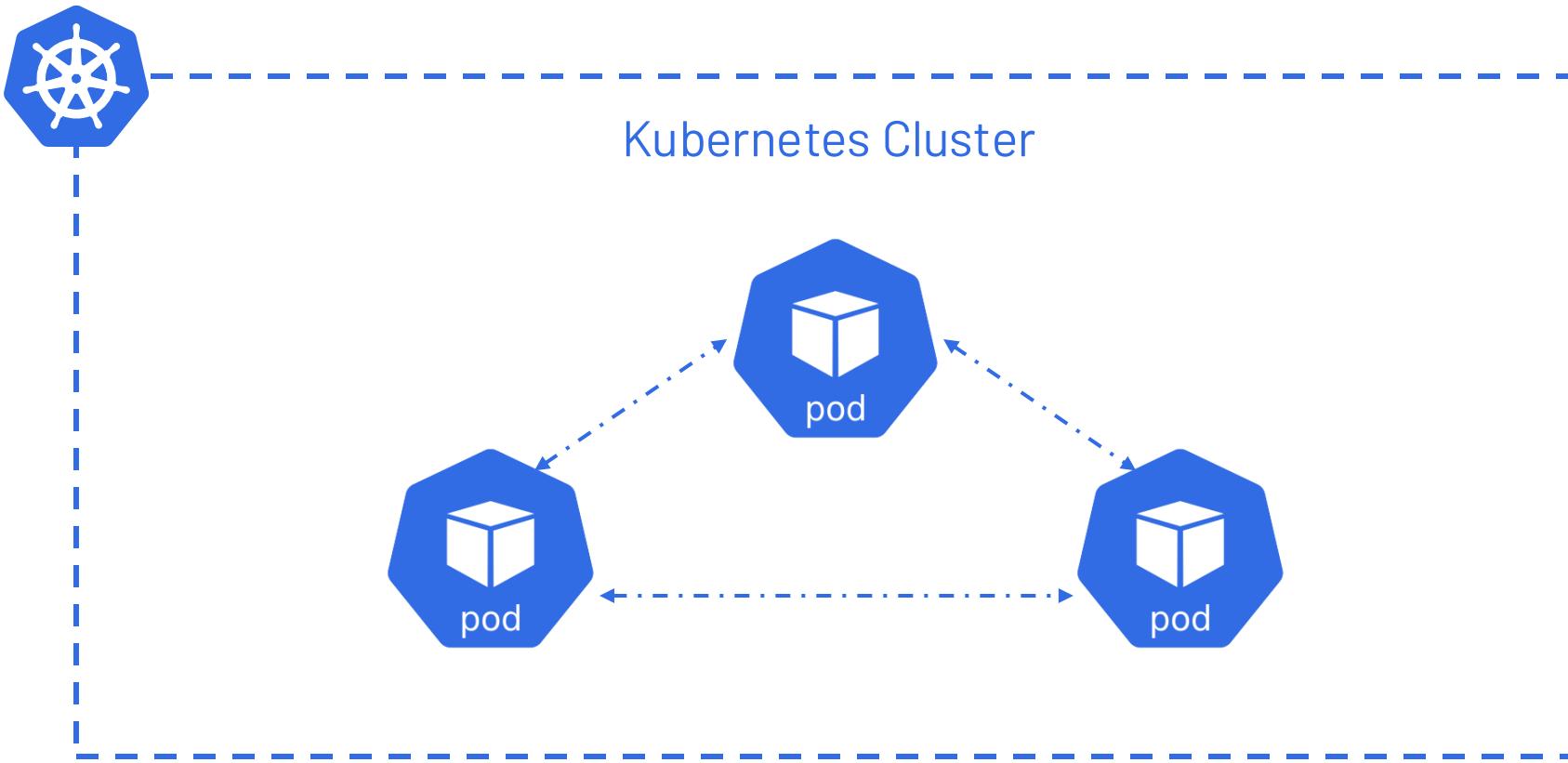
Section Overview

- **Network Policies**
- **How to define communication rules**



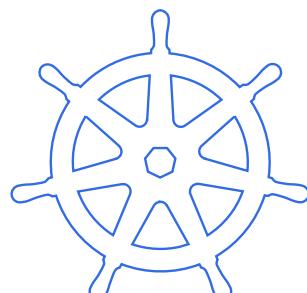
Network Policies

- Network Policies are essential for managing network traffic between pods within a cluster



Network Policies

- Network Policy controls how pods communicate with each other and external services in Kubernetes
- It specifies rules for incoming (ingress) and outgoing (egress) traffic based on pod labels, IP ranges, and ports
- Network Policies are declarative



Network Policies

Key Components

podSelector

```
podSelector:  
  matchLabels:  
    app: testapp
```

Used to specify which Pods
the policy applies to

namespaceSelector

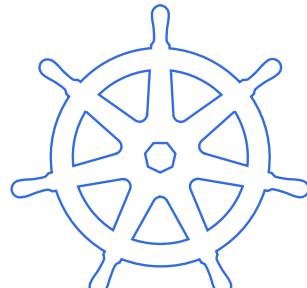
```
namespaceSelector:  
  matchLabels:  
    environment: production
```

Used to select Pods based on
the labels of the namespace
they reside in

ipBlock

```
ipBlock:  
  cidr: 192.168.1.0/24  
  except:  
    - 192.168.1.10/32
```

Allows the policy to apply to
specific IP address ranges



Network Policies

Traffic Rules

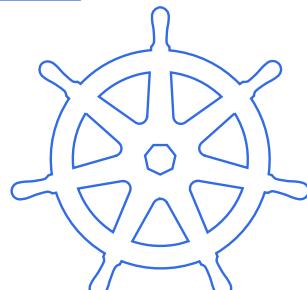
- Network Policy defines rules that control traffic by specifying allowed sources for incoming or outgoing connections based on various selectors and criteria

Controls incoming traffic to
the selected pods

Ingress

Controls outgoing traffic from
the selected pods

Egress



```
root@master:/var/tmp/k8sfiles# cat 25_ingresspol.yaml
```

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
```

```
metadata:
```

```
  name: ingresspol
```

```
  namespace: default
```

```
spec:
```

```
  podSelector:
```

```
    matchLabels:
```

```
      app: thinknyxtwo
```

```
  policyTypes:
```

- Ingress
- Egress

```
  ingress:
```

```
    - from:
```

```
      - podSelector:
```

```
        matchLabels:
```

```
          app: thinknyxone
```

```
    - ports:
```

```
      - port: 80
```

```
        protocol: TCP
```

```
  egress:
```

```
    - to:
```

```
      - namespaceSelector:
```

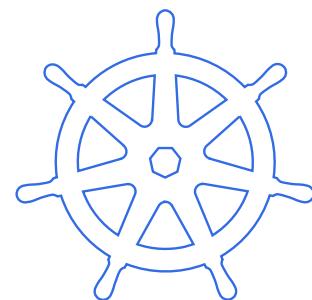
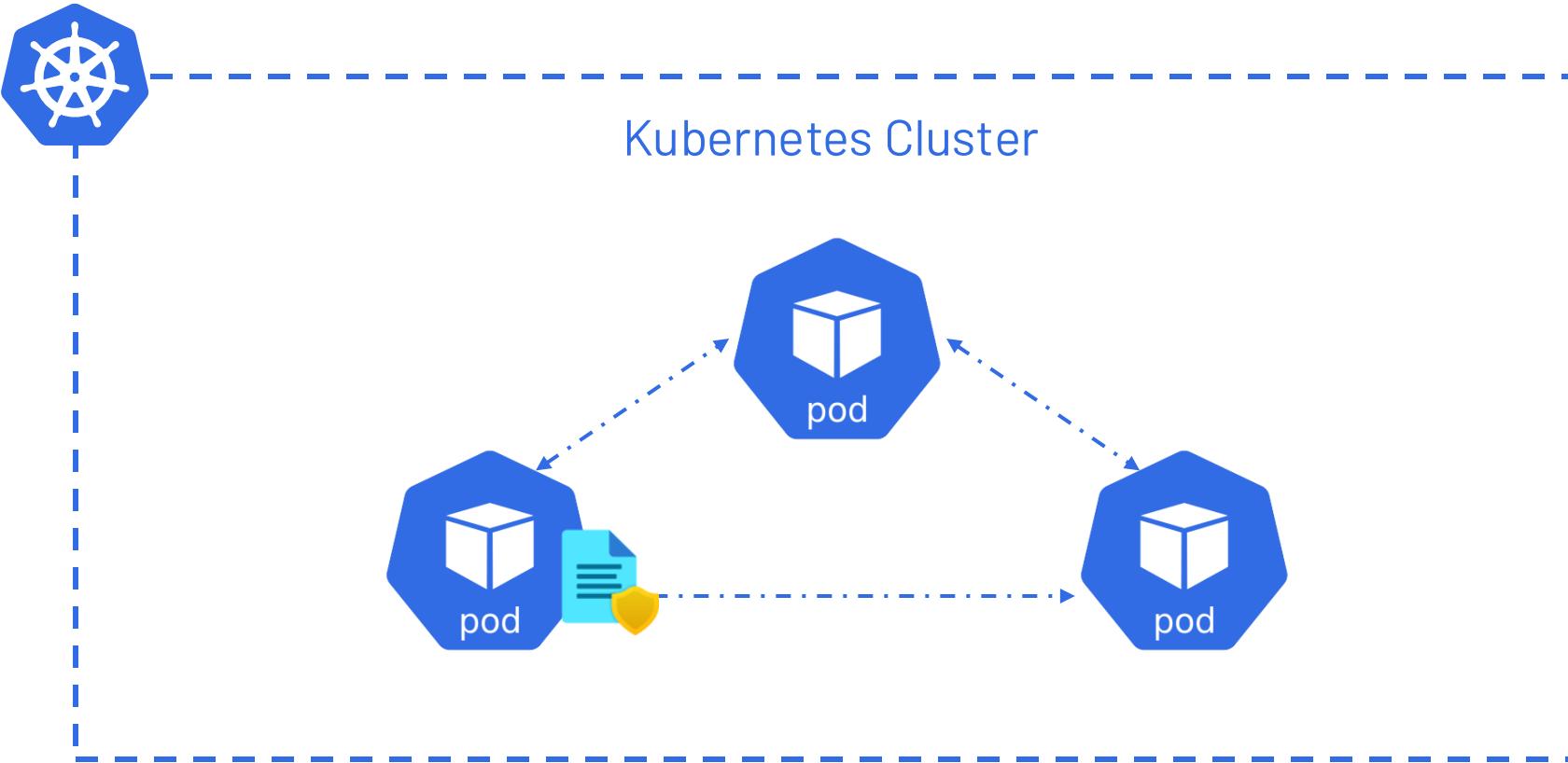
```
        matchLabels:
```

```
          app: thinknyxns
```



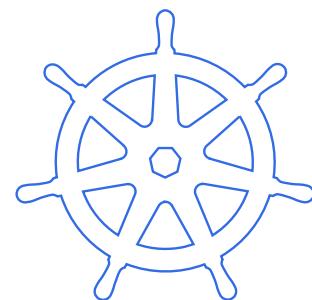
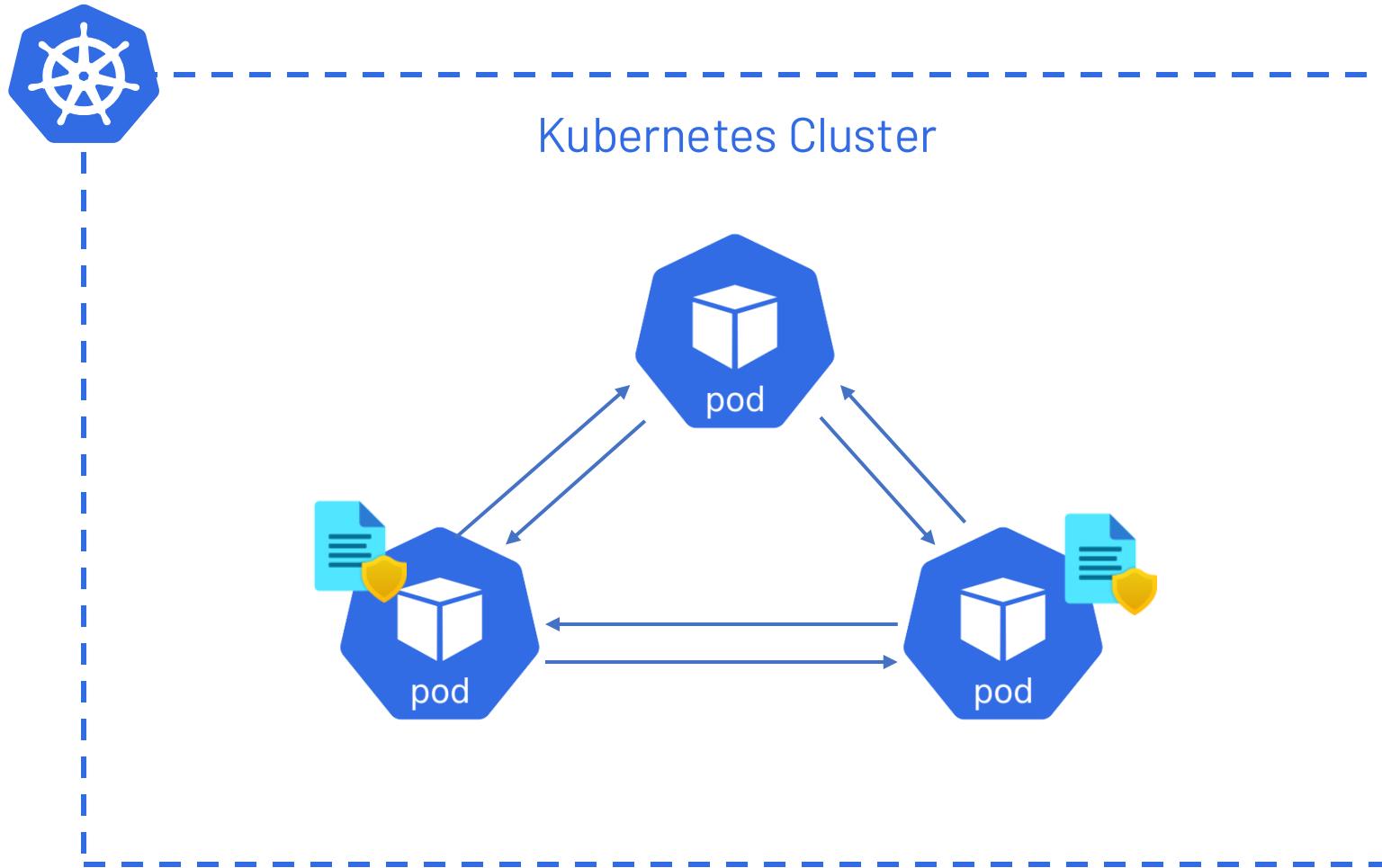
Network Policies

Default behavior



Network Policies

Default behavior



Network Policies

Limitations



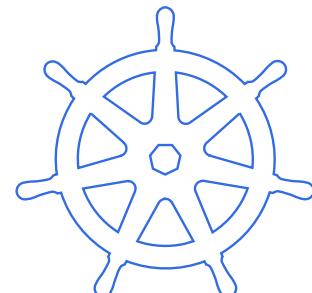
Scope



Kubernetes
Network Plugins



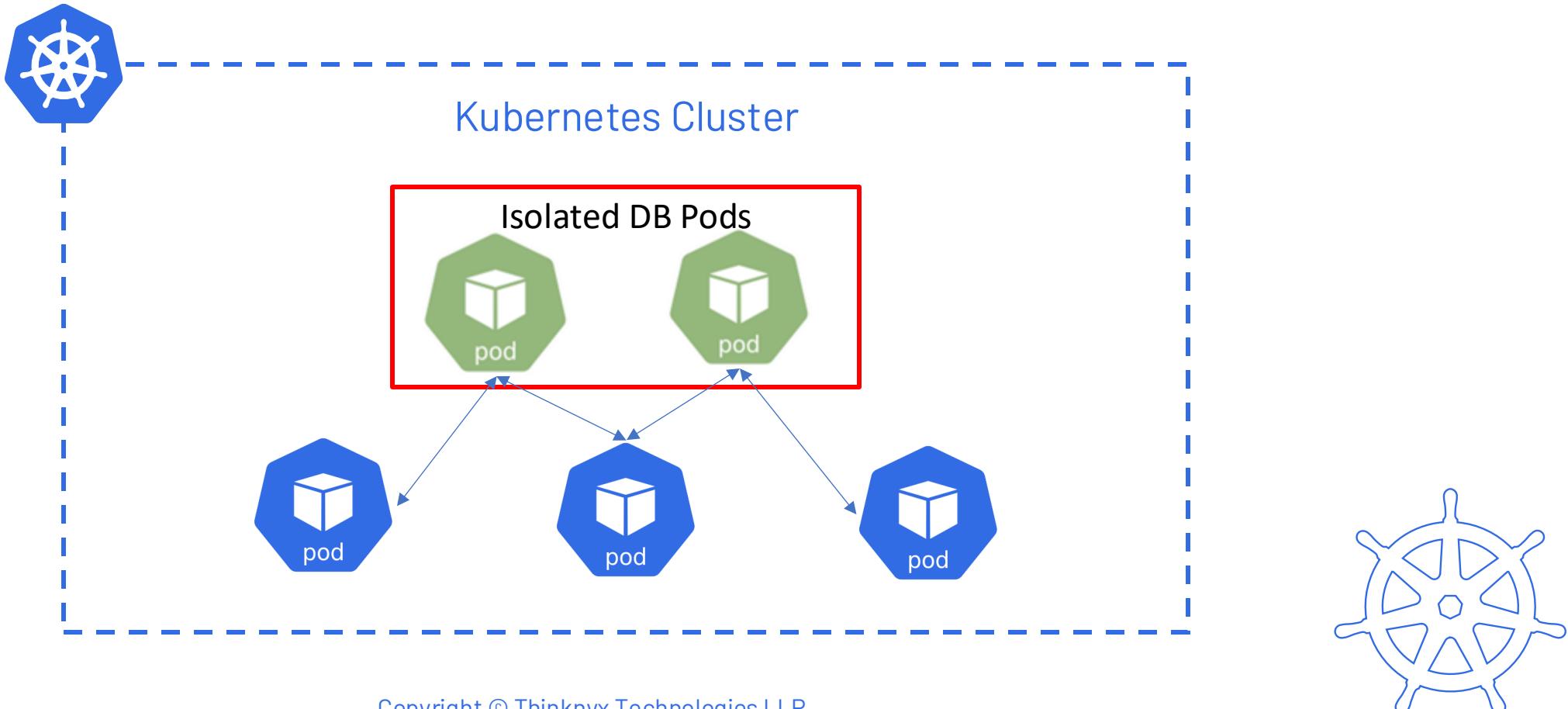
Cluster External
Traffic



Network Policies

Practical Uses

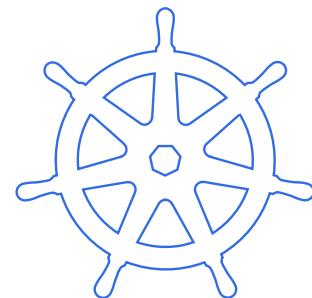
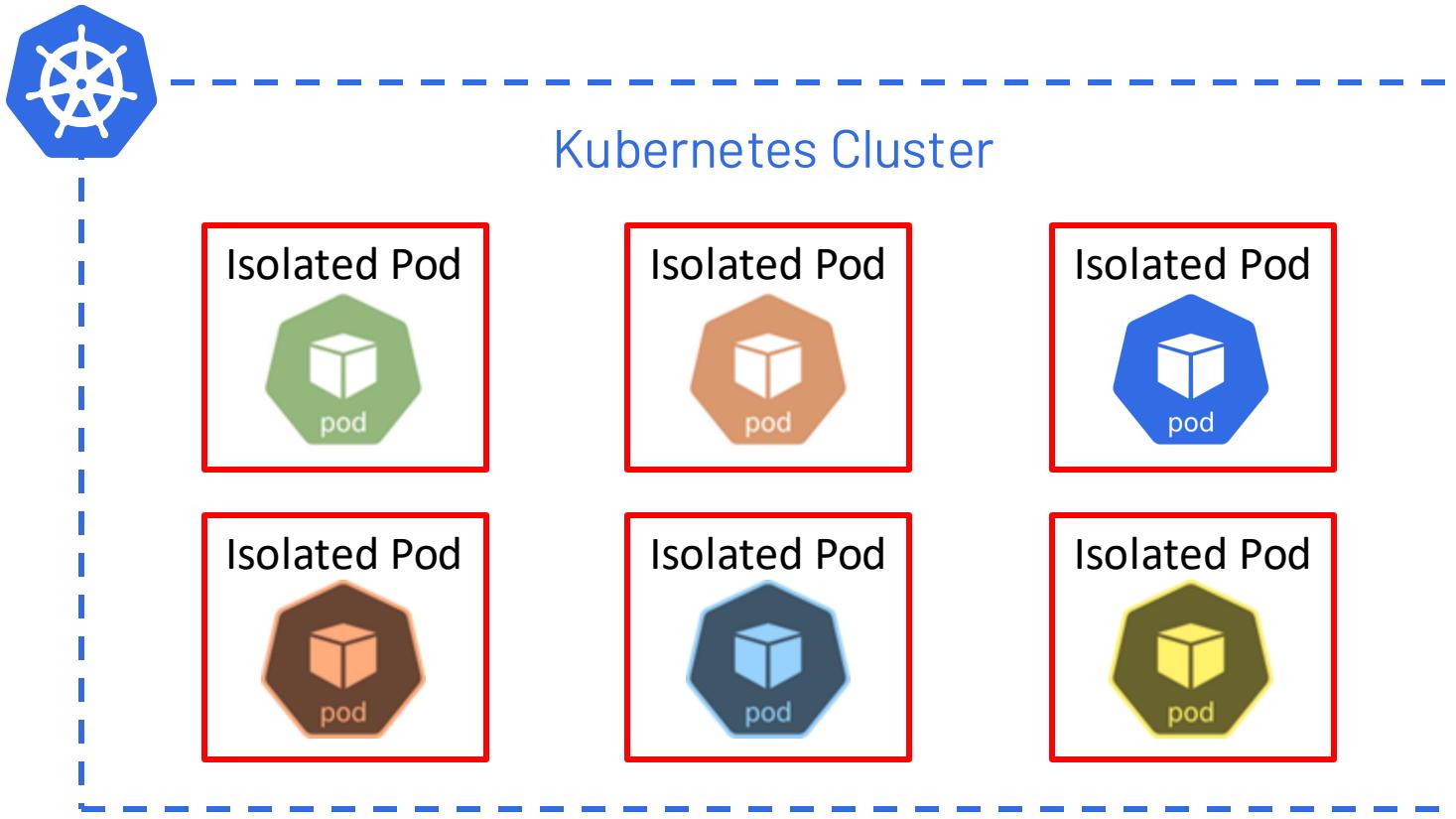
- Isolating Sensitive Pods



Network Policies

Practical Uses

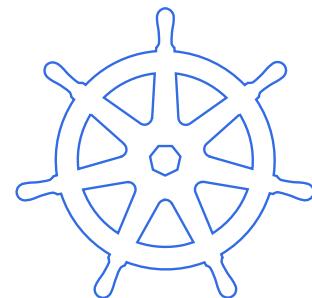
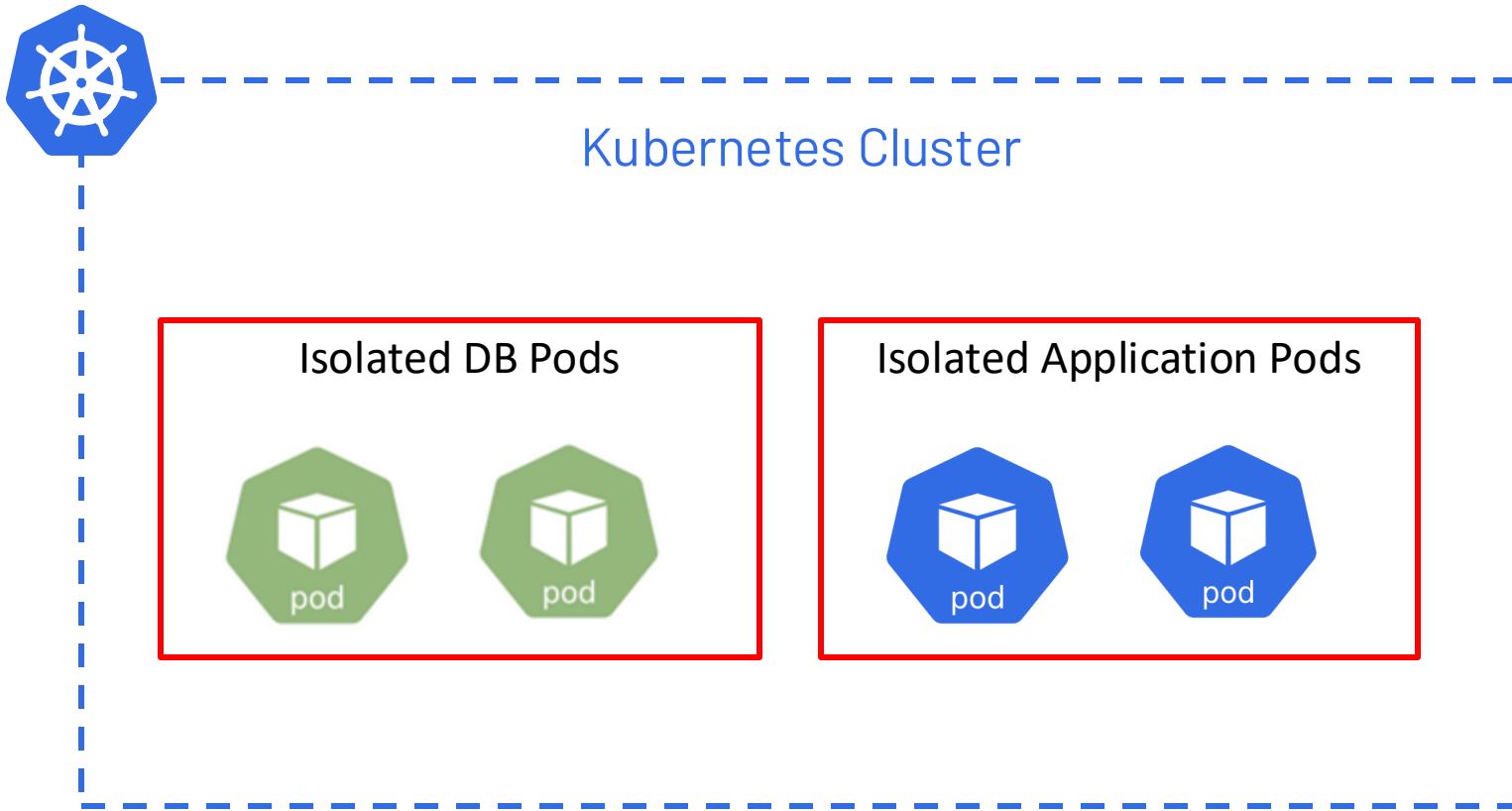
- Securing Microservices



Network Policies

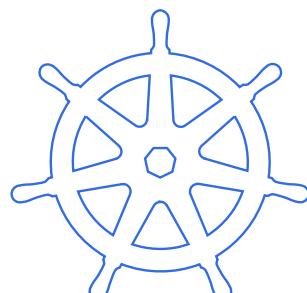
Practical Uses

- Regulatory Compliance



Network Policies

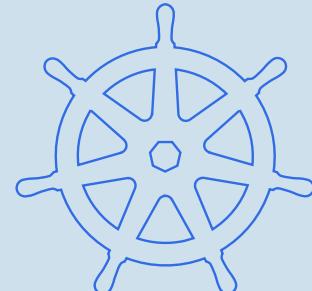
- Network Policies control how pods communicate inside Kubernetes clusters
- Enhance cluster security by enforcing rules that restrict traffic based on defined criteria
- Administrators can define incoming and outgoing traffic to prevent unauthorized access





Demo

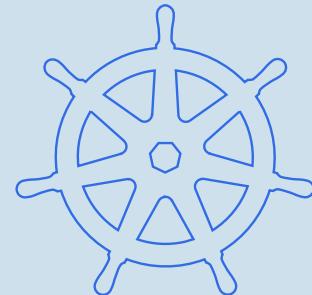
| Default All Allow
Policy





Demo

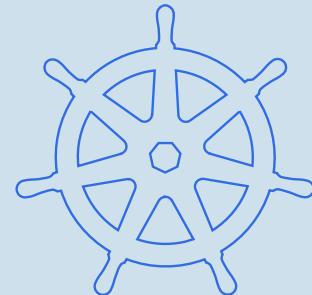
| All Deny Policy





Demo

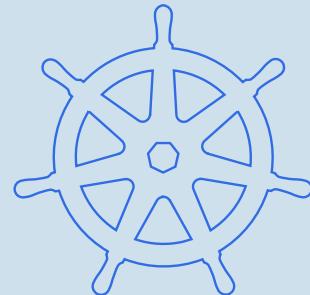
Ingress & Egress
Policy with DNS Allow





Demo

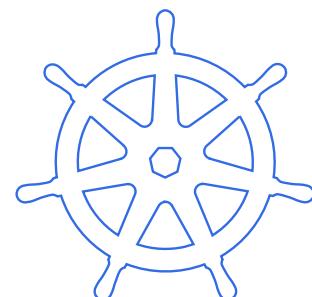
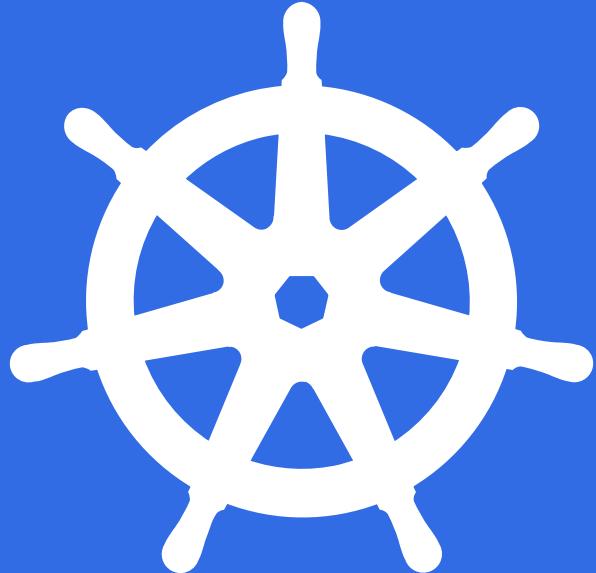
Namespace Scoped Policy



Summary

Summary

- ✓ Network Policies to control internal traffic between pods
- ✓ How to secure and manage traffic flow within the cluster

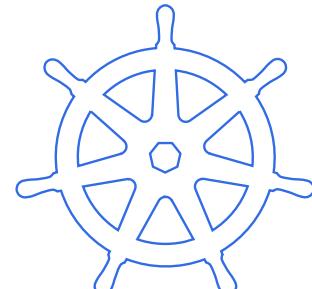


Section: 18

Monitoring & Cluster Dashboard

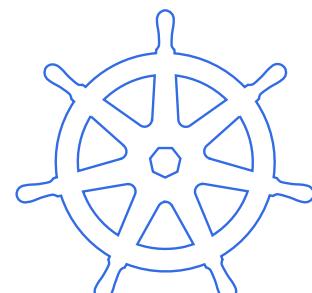
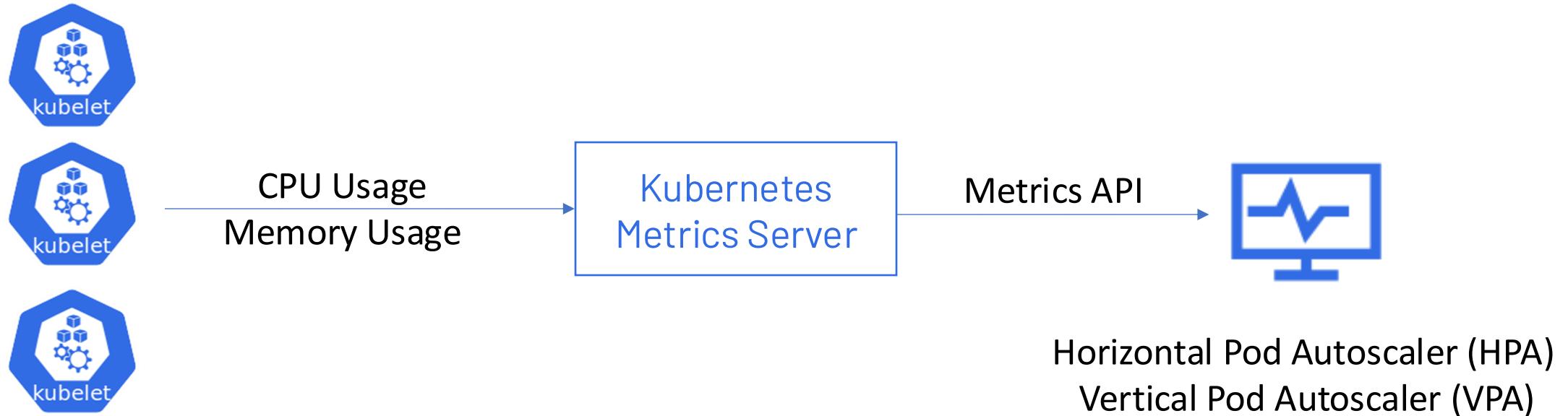
Section Overview

- └ Metrics-server
- └ Horizontal Pod Autoscaler (HPA)
- └ Kubernetes Dashboard



Monitoring through Metrics-server

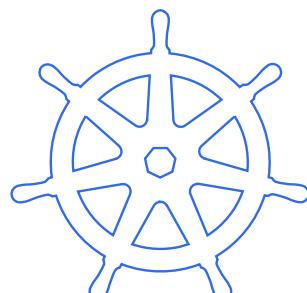
Monitoring through Metrics-server



Monitoring through Metrics-server

Kubernetes
Metrics Server

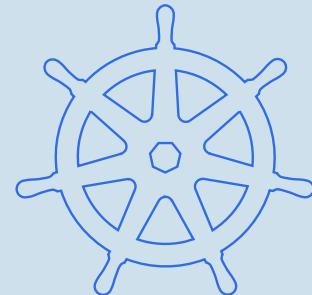
- ✓ Lightweight and efficient resource usage
- ✓ Collects metrics every 15 seconds
- ✓ Supports clusters with up to 5,000 nodes
- ✓ Not for detailed monitoring/logging
- ✓ Use alongside tools like Prometheus





Demo

Monitoring through
Metrics-server

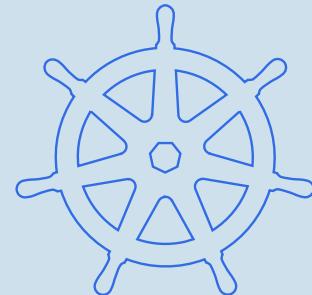


Horizontal Pod Autoscaler (HPA)



Demo

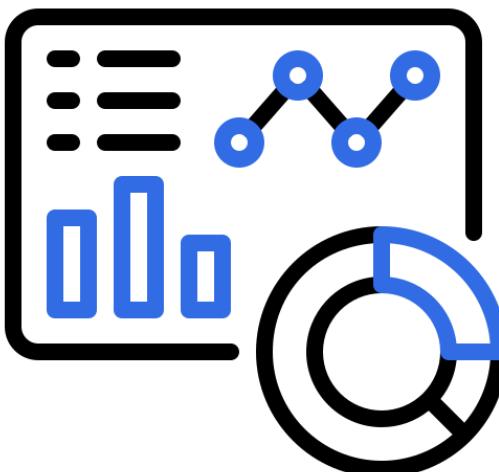
Horizontal Pod
Autoscaler(HPA)



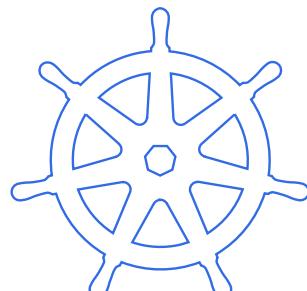
Kubernetes Dashboard

Kubernetes Dashboard

Web-based user interface that allows users to manage & monitor their Kubernetes clusters



- ✓ View & control running applications
- ✓ Troubleshoot issues
- ✓ Perform various administrative tasks



☰ Workloads

Workload Status

Workloads (N)

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

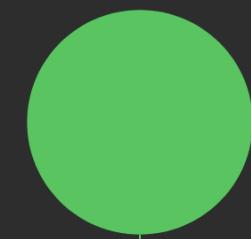
Service

Ingresses (N)

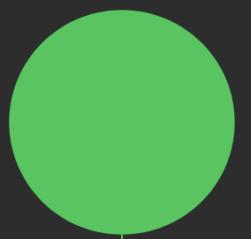
Ingress Classes

Services (N)

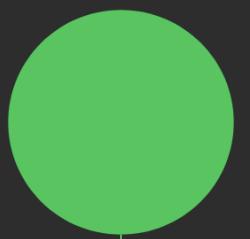
Config and Storage

Config Maps (N)Persistent Volume Claims (N)Secrets (N)

Deployments



Pods



Replica Sets

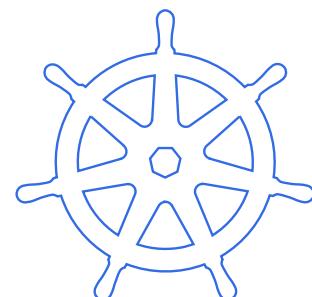
Deployments

| Name | Images | Labels | Pods | Created | ⋮ |
|-----------|--------------|----------------|-------|-------------|---|
| webappdep | httpd:latest | app: webappdep | 3 / 3 | an hour ago | ⋮ |

Pods

| Name | Images | Labels | Node | Status | Restarts | CPU Usage (cores) | Memory Usage (bytes) | Created | ⋮ |
|---------------------|--------------|----------------|--------|---------|----------|-------------------|----------------------|-----------|---|
| webappdep-54888f54t | httpd:latest | app: webappdep | worker | Running | 0 | - | - | 3 minutes | ⋮ |

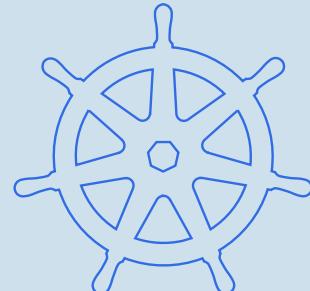
Kubernetes Dashboard





Demo

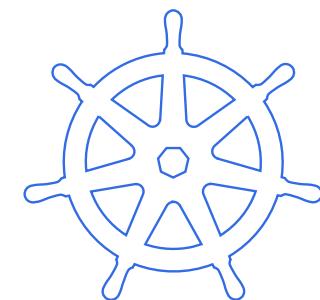
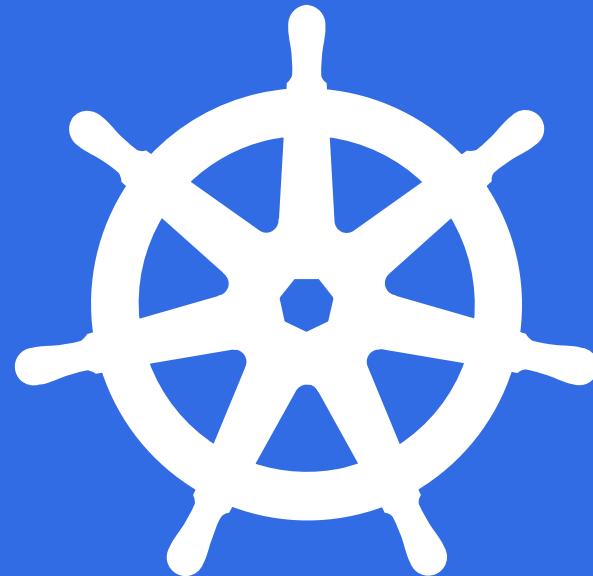
Kubernetes Dashboard



Summary

Summary

- ✓ Metrics-server
- ✓ Horizontal Pod Autoscaler (HPA)
- ✓ Kubernetes Dashboard

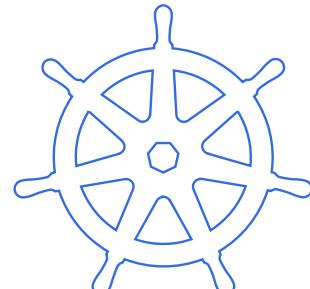


Section: 19

Kubernetes Cluster Maintenance

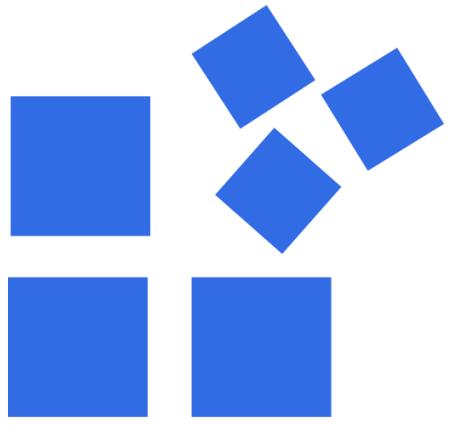
Section Overview

- ↳ **Private Registry Integration**
- ↳ **Setup and Integrate with Cluster**

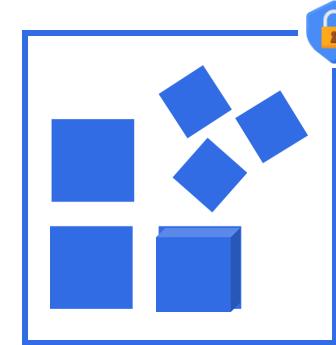


Private Registry Integration

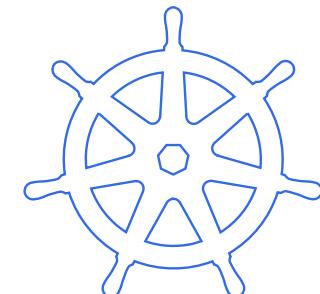
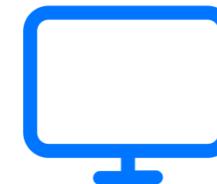
Private Registry Integration



Public Registry



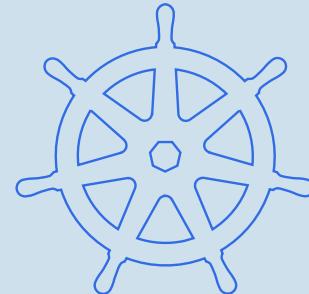
Private Registry



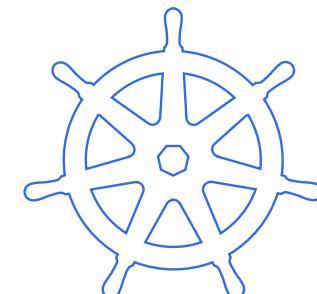
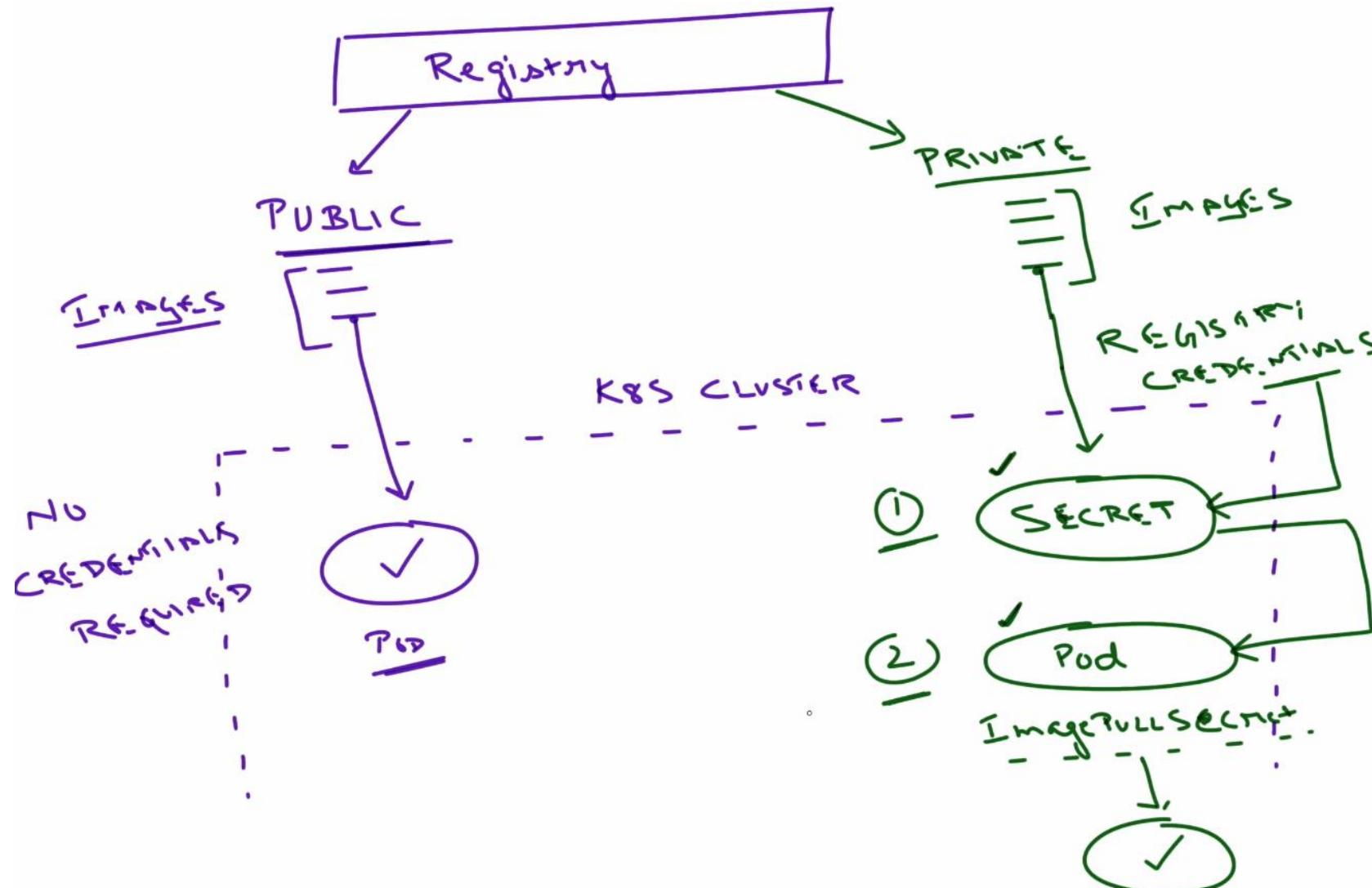


Demo

Private Registry
Integration



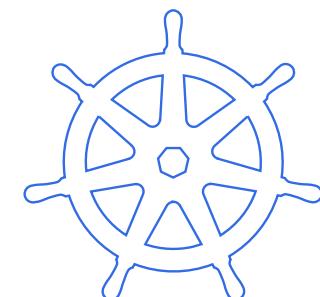
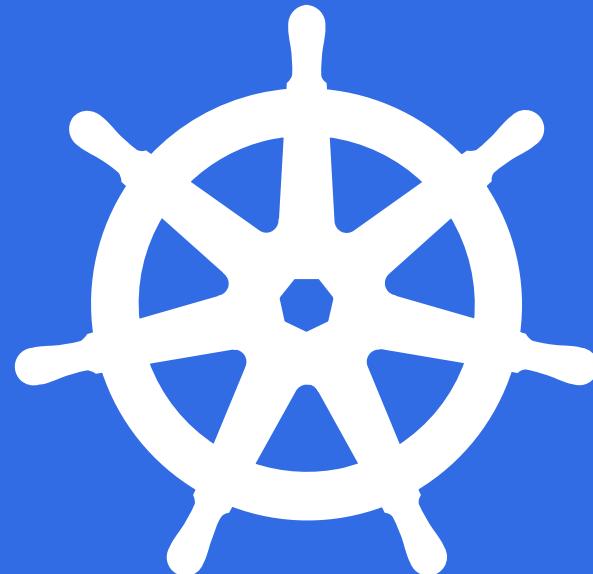
Private Registry Integration



Summary

Summary

- ✓ Private Registry Integration with Kubernetes
- ✓ Secure Image access

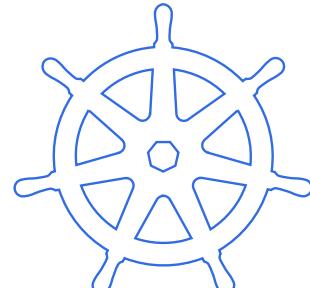


Section: 20

Section Overview

Helm Charts

- ↳ **Helm**
- ↳ **Helm Charts**
- ↳ **Helm Charts Creation**
- ↳ **Artifact Hub**

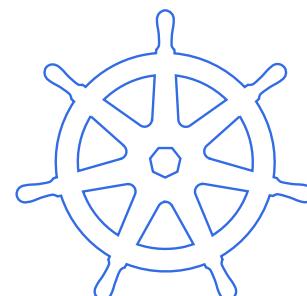


What is Helm?

What is Helm?



- Package Manager for Kubernetes
- Simplifies the process of defining, installing and upgrading applications in Kubernetes
- Charts: Pre-packaged collections of Kubernetes resources & configurations



Why Helm?

Why Helm?



Application
Packaging



Efficient
Deployment



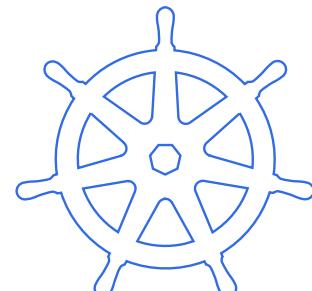
Reusability &
Consistency



Community
Collaboration



Customization &
Flexibility



Key Terminologies

Helm Charts

Helm Repositories

values.yaml

Hooks

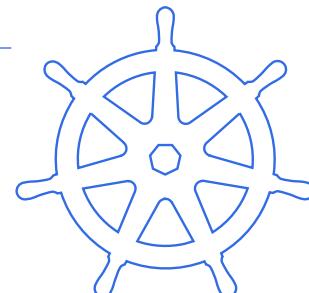


Releases

Template Rendering

Chart.yaml

Helm Plugins



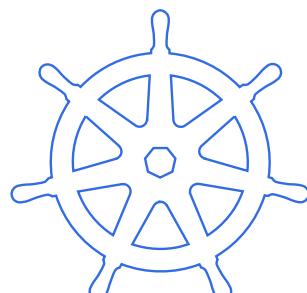
Installing Helm

Step-1: Download Helm

```
$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3  
$ chmod 700 get_helm.sh  
$ ./get_helm.sh
```

Step-2: Verify Installation

```
$ helm version
```



Helm Chart Structure

```
$ helm create mychart
```

```
mychart/
├── Chart.yaml          # Metadata about the chart
├── values.yaml         # Default configuration values for this chart
└── charts/              # Subcharts (dependencies)
  └── templates/          # Template files for Kubernetes resources
    ├── deployment.yaml
    ├── service.yaml
    └── _helpers.tpl       # Template helpers for reusability
```

Operating with Helm

helm create

```
$ helm create my-chart
```

helm lint

```
$ helm lint my-chart
```

helm package

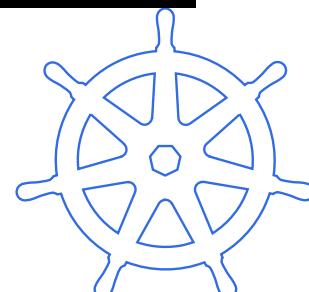
```
$ helm package my-chart
```

helm install

```
$ helm install my-release my-chart
```

helm ls

```
$ helm ls
```



Operating with Helm

helm status

```
$ helm status my-release
```

helm history

```
$ helm history my-release
```

helm upgrade

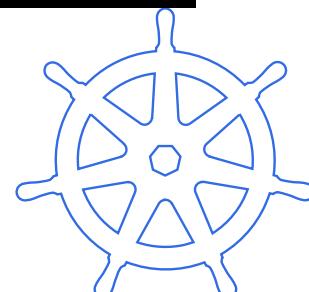
```
$ helm upgrade my-release my-chart
```

helm uninstall

```
$ helm uninstall my-release
```

helm repo

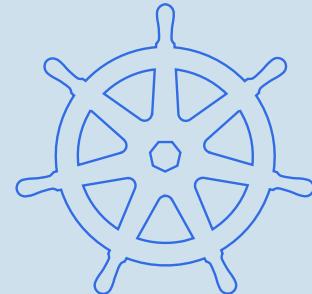
```
$ helm repo add stable https://charts.helm.sh/stable
```





Demo

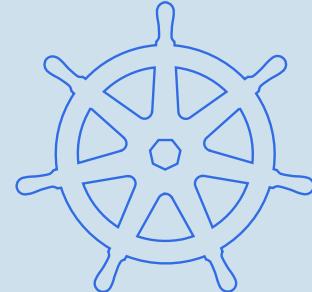
Developing Helm Charts





Demo

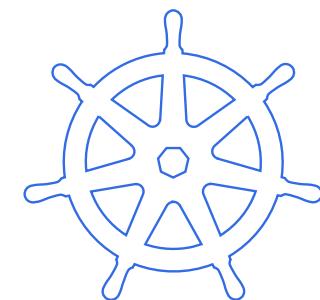
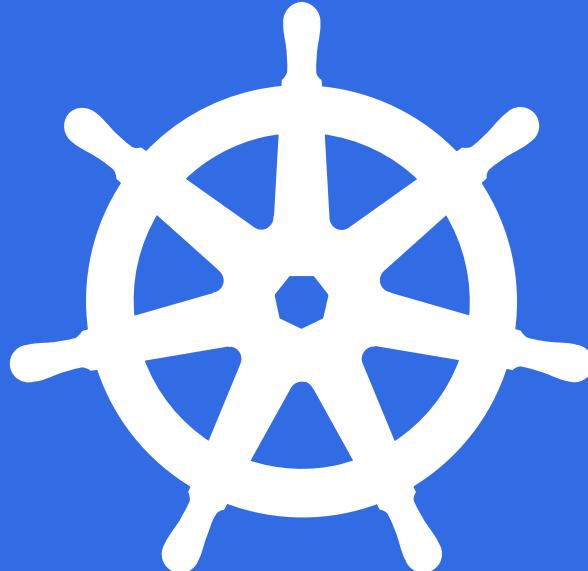
Helm Charts from
Artifact Hub



Summary

Summary

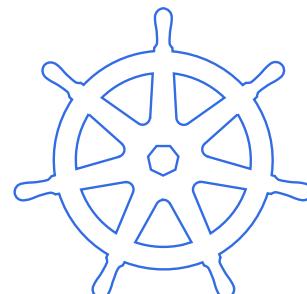
- ✓ Helm
- ✓ Helm Charts
- ✓ Developed Helm Charts
- ✓ Artifact Hub



Conclusion

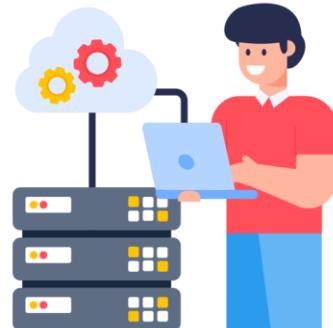


Practical Kubernetes - Beyond CKA & CKAD Hands-on





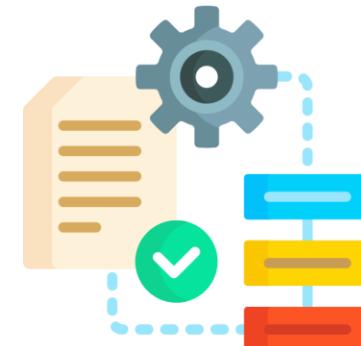
Deploy



Manage



Troubleshoot



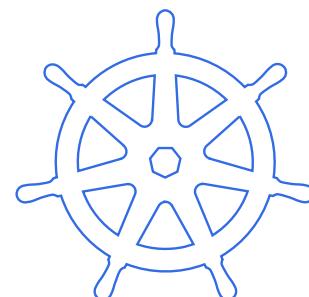
Advanced
Configurations

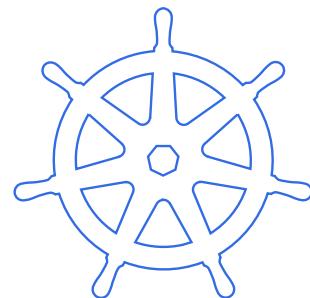
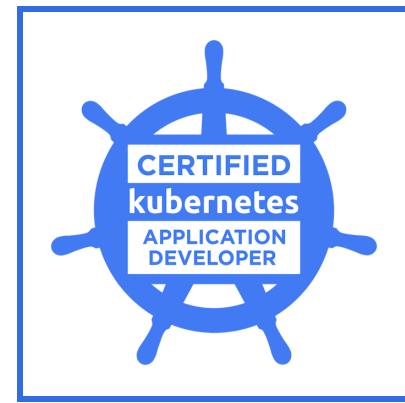


Security



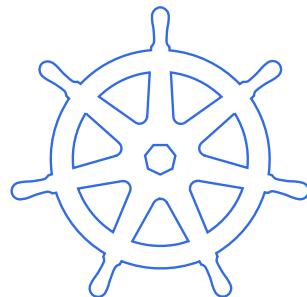
Ingress







kubernetes





Follow us on:

 @thinknyx

 @thinknyx

 @thinknyx-technologies

 @thinknyx

 @thinknyx-technologies