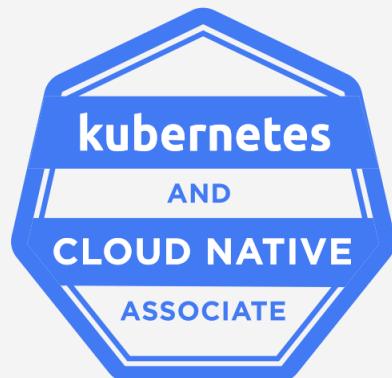




KCNA

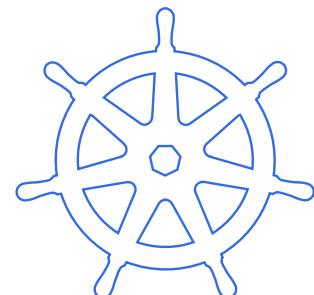
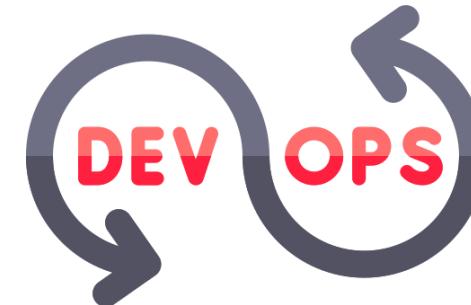
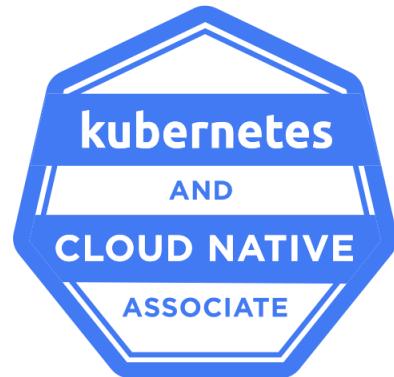
(Kubernetes and Cloud Native Associate)

Certification Preparation Course!





Yogesh Raheja





Puppet for the Absolute
Beginners – Hands-On



Generative AI Essentials -
Practical Use Cases



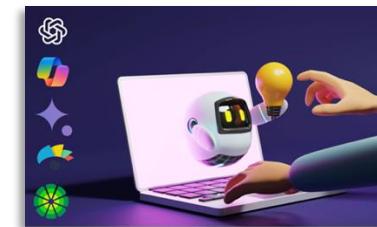
SaltStack for the Absolute
Beginners – Hands-On



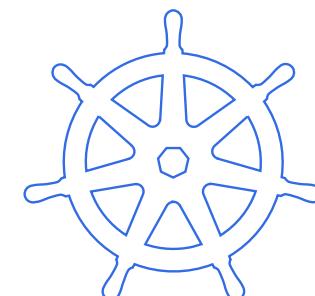
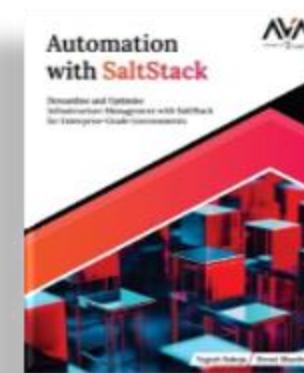
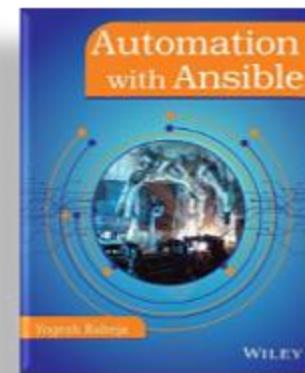
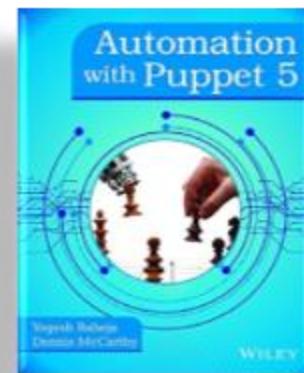
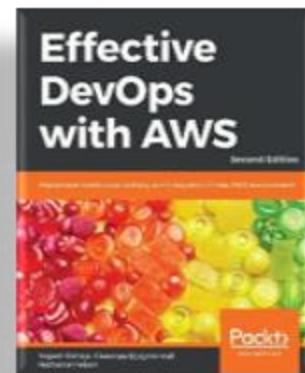
Infrastructure Automation
with OpenTofu – Hands-On



AI Ecosystem for the Absolute
Beginners - Hands-On



Mastering Prompt
Engineering for GenAI

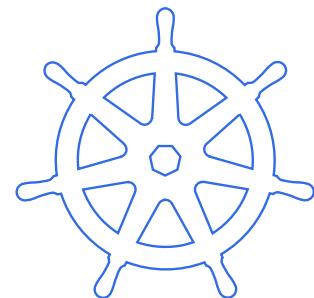




Yogesh Raheja



Thinknyx Technology Team



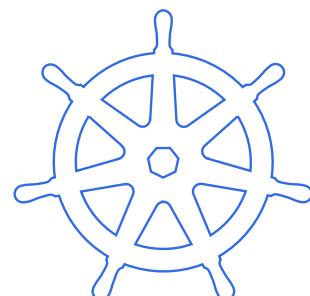
Containers &
Container Orchestration

Kubernetes Fundamentals

Cloud Native Architecture &
Observability

Cost Management for Cloud
Native Environments

Cloud Native Application
Delivery Principles





Comprehensive Learning Modules



kubernetes

Architecture

Scheduling

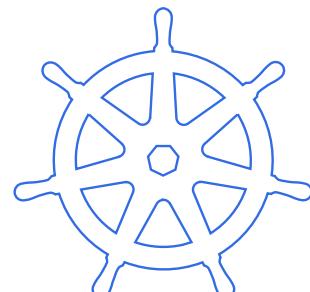
Networking

Storage

Security

Observability

Application Delivery



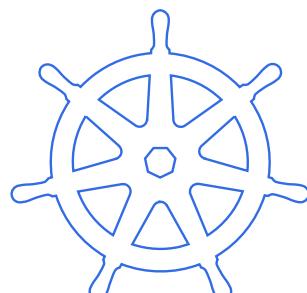


Comprehensive Learning Modules



Hands-On Labs & Demos

```
root@master:~#  
root@master:~# kubectl get nodes  
NAME     STATUS   ROLES      AGE     VERSION  
master   Ready    control-plane   6d20h   v1.30.1  
worker   Ready    worker       6d20h   v1.30.1  
root@master:~#  
root@master:~#  
root@master:~# kubec
```





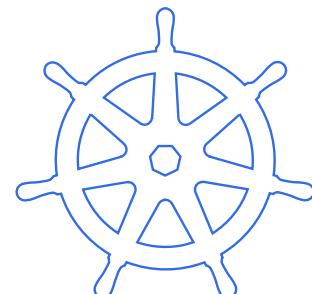
Comprehensive Learning Modules

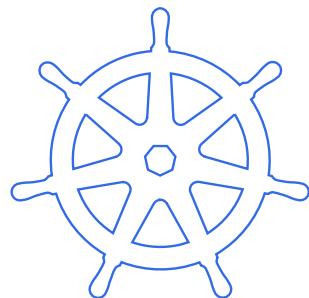


Hands-On Labs & Demos



Practice Tests and Exam Preparation Tips





KCNA Exam Overview

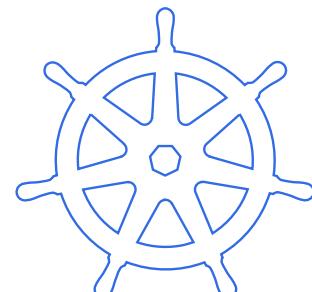
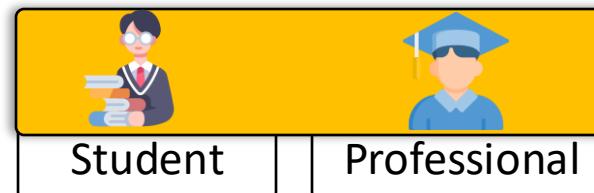
KCNA Exam Overview



kubernetes

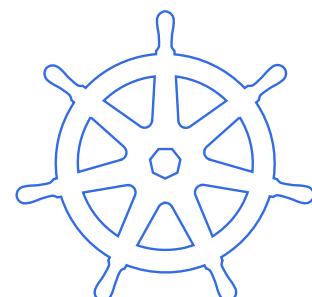
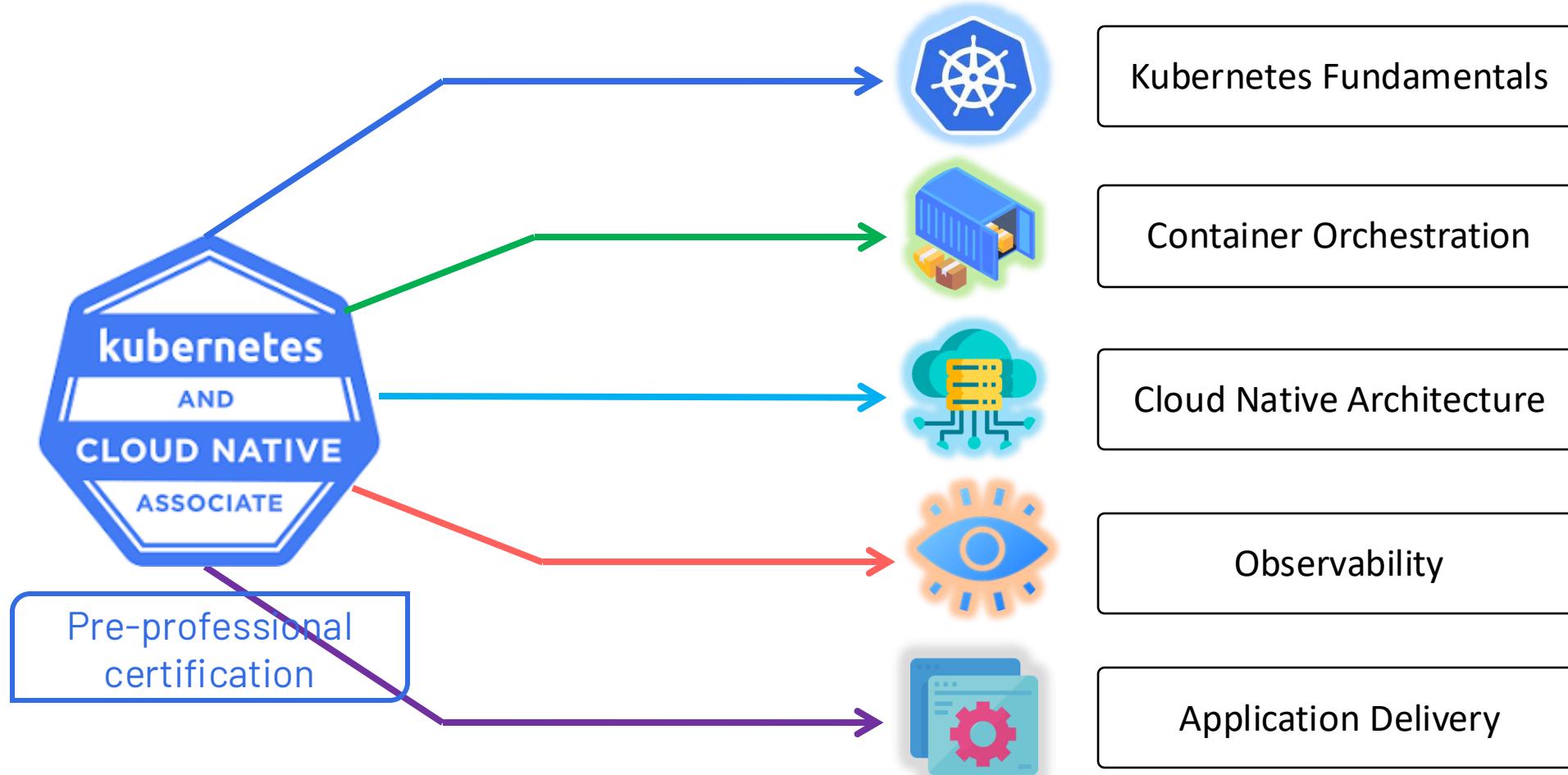


Cloud Native
Technologies



What is KCNA

What is KCNA



What is KCNA

Essential concepts



Kubernetes architecture



API



Containers



Networking



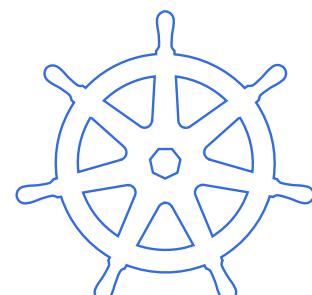
Security



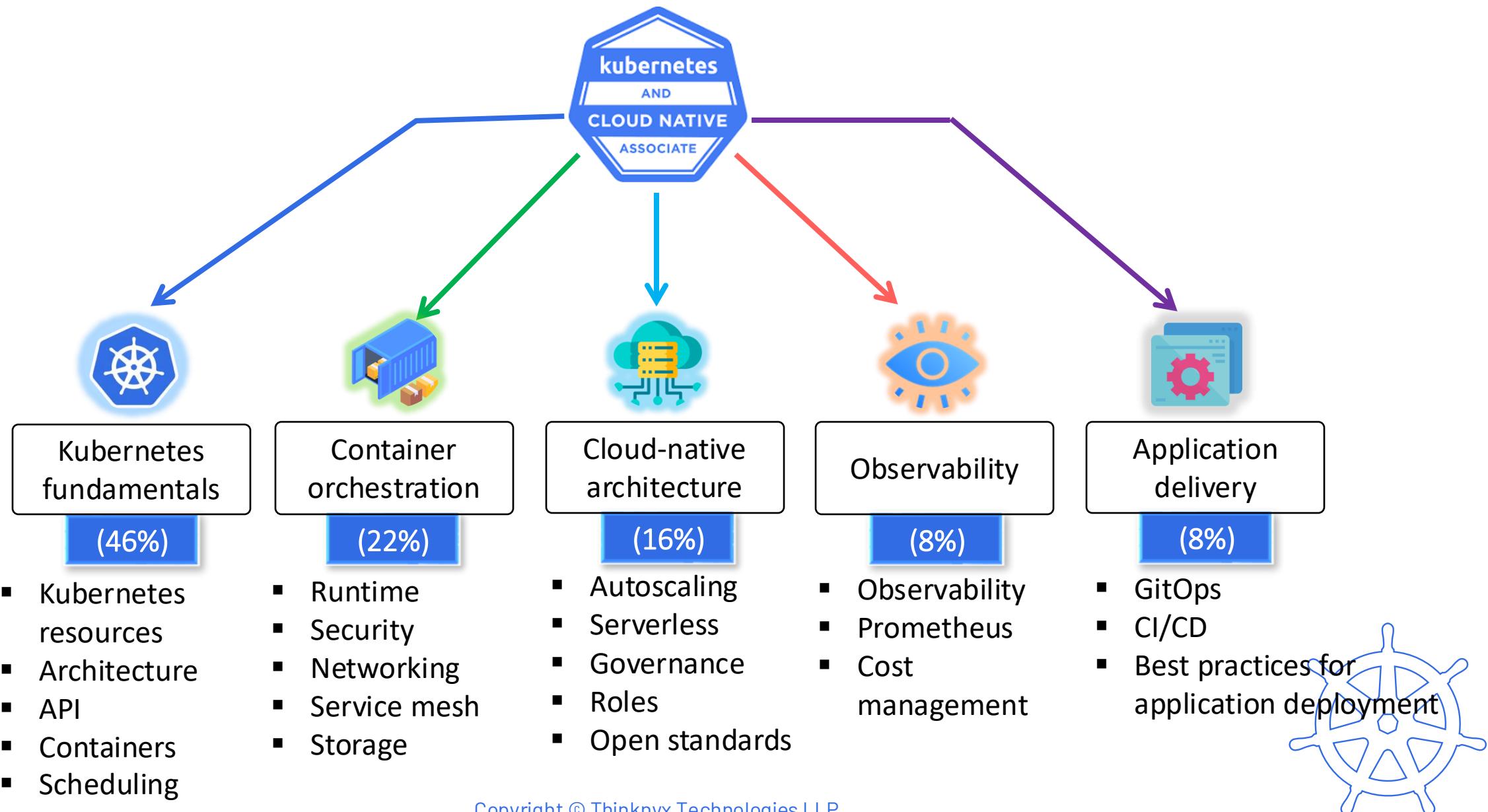
GitOps



CI/CD



Exam Domains



Exam Domains

Important note



A B C
○ ● ○



CKA
(Certified Kubernetes Administrator)



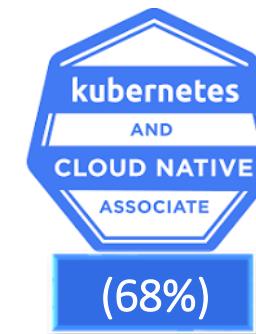
CKAD
(Certified Kubernetes Application Developer)



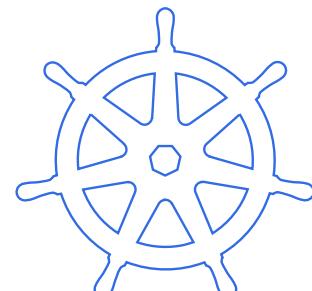
Kubernetes
fundamentals
(46%)



Container
orchestration
(22%)



(68%)



Exam Domains



No strict
prerequisites



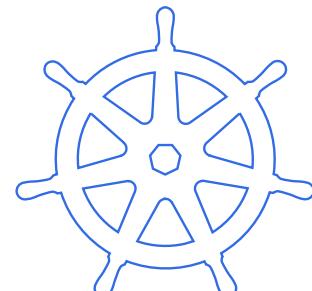
Basic understanding
of Linux



kubernetes

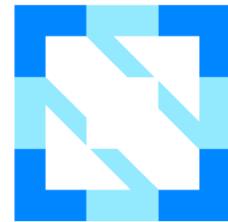


Cloud-native
Tools



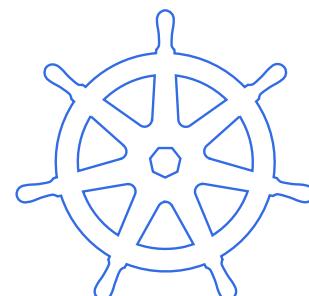
Overview of CNCF

Overview of CNCF



**CLOUD NATIVE
COMPUTING FOUNDATION**

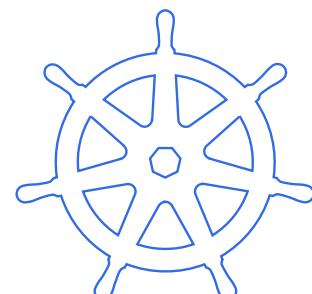
2015



Overview of CNCF

Cloud-native computing

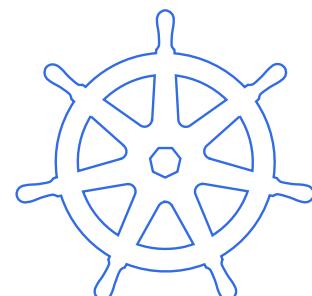
Design, development, & deployment of applications that leverage cloud computing models to build scalable, resilient, & agile systems



Overview of CNCF



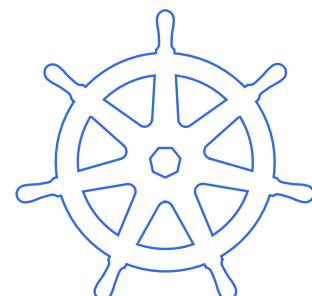
Neutral home for open-source projects



Mission and Vision

Mission and Vision

- CNCF supports the development of open-source projects for scalable cloud-native applications
- Its vision is a world where these technologies are accessible, interoperable, and secure



Key Focus Areas

Key Focus Areas

Containerization



Orchestration



Microservices



Service Mesh



Observability



Prometheus



JAEGER

Key Focus Areas

Security



Application Definition & Image Builds



kustomize.io



CI/CD



argo



TEKTON



Database

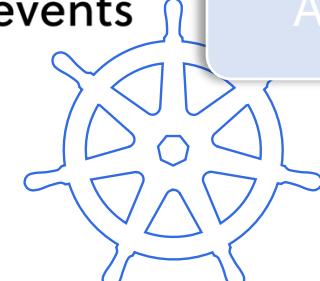


Vitess



TiKV

Streaming & Messaging



Key Focus Areas

Scheduling & Orchestration



Cloud Native Storage



Container Runtime



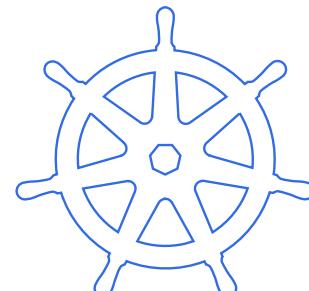
Security & Compliance



Automation & Configuration



kubectl

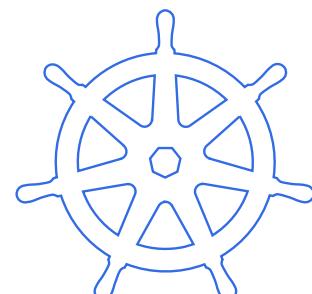


Technical Collaboration

Technical Collaboration

Technical Advisory Groups
(TAGs)

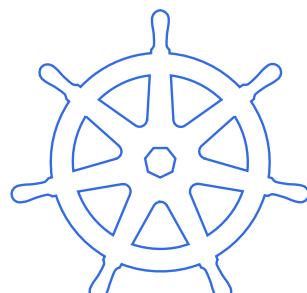
TAGs help grow contributions from both technical and user communities while ensuring quality and integrity for CNCF's mission



Technical Collaboration

Technical Oversight Committee
(TOC)

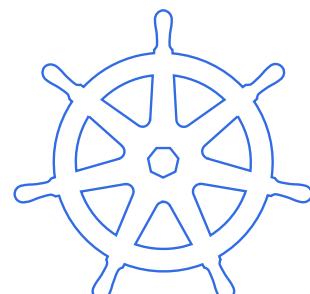
Attend TOC meetings and subscribe to its mailing list to stay informed about CNCF developments and decisions



Technical Collaboration

CNCF Projects

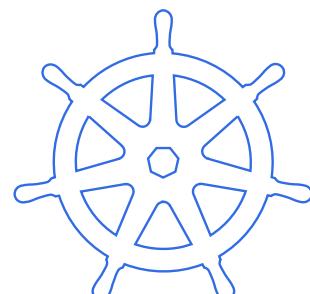
CNCF supports and guides rapidly growing cloud-native projects and offers a way for organizations to donate projects on GitHub



Technical Collaboration

Special Interest Groups
(SIGs)

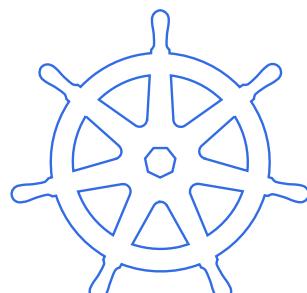
SIGs are community groups focused on specific topics within a larger project, like Kubernetes, to drive innovation and solve technical challenges



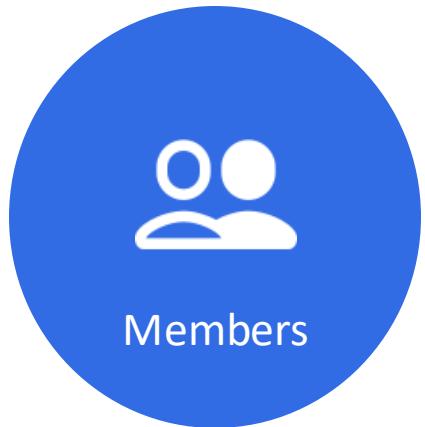
Technical Collaboration

Kubernetes Enhancement Proposals
(KEPs)

KEPs are formal proposals for feature enhancements in Kubernetes, outlining design, rationale, and implementation details



Technical Collaboration



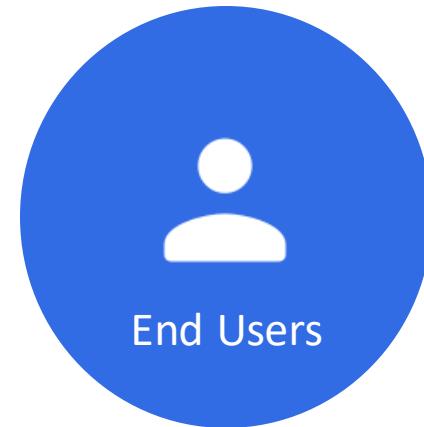
Members

Organizations that support the foundation with funding, resources, and advocacy to help guide its projects



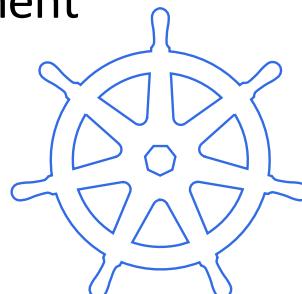
Contributors

Individuals or groups who actively help develop and improve CNCF projects through collaboration and leadership



End Users

People or organizations that use CNCF technologies to build and manage cloud-native applications, providing valuable feedback for improvement



CNCF Landscape

Filters

GROUP:

Projects and products

Members

Certified partners and providers

Serverless

Wasm

CNAI

VIEW MODE:

Grid

Card

ZOOM:



App Definition and Development

Application Definition & Image Build

CNCF GRADUATED	CNCF GRADUATED	CNCF INCUBATING	CNCF INCUBATING	CNCF INCUBATING	KUBEVELA	CARVEL
CNCF INCUBATING	CNCF INCUBATING	Devfile.io	DevSpace	Eclipse Che™	fabric8	Cyclops
itopia	Kaniko	Kapeta	ko	kots	KRATOR	KUBERMATIC
lagoon	mia Platform	MICROCKS	mirror	MONOKLE	Okteto	Open Application Model
QUARKUS	radius	rafft	Rig.dev	SCORE	sealer	ServiceComb
TELPUSIONE	TILT	VMware Application Catalog	Walrus			

Continuous Integration & Delivery

CNCF GRADUATED	CNCF GRADUATED	CNCF INCUBATING	CNCF INCUBATING	AKUITY
AppVeyor	AWS CodePipeline	Azure Pipelines	Bamboo	BRIGADE
CARTOGRAPHER	circleci	CloudBees	codefresh	Buildkite
GitHub Actions	GitLab	gitnoss	go	bunnyshell
JENKINS X	K6	Liquibase	Mergify	Bytebase
ozone	Keplor	Pipedream	Northflank	HelmWave
Porter	Razee	Screwdriver.cd	Octopus Deploy	OpenGitOps
TEKTON	terramate	TESTKUBE	Travis CI	psMx
unleash	unleash	unleash	Werf	TeamCity
spacelift	semaphore	spacelift	kubeburner	Spinnaker

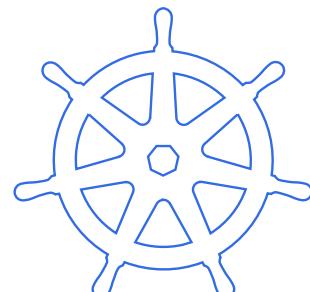
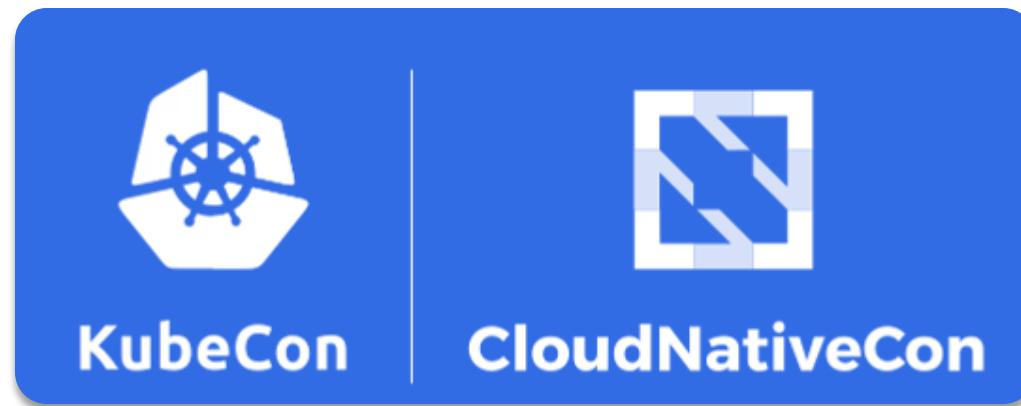
Database

TiKV	Vitess	CarbonData	Apache Flink	ignite	ArangoDB	BIGCHAINDB	cassandra	ClickHouse	CloudNativePG	Cockroach Labs	Couchbase	CRATE.IO
CNCF GRADUATED	CNCF GRADUATED											
GreptimeDB	hazelcast IMDG	IBM DB2	iguazio	infinispan	InterSystems	KUBEBLOCKS	KubeDB	MariaDB	MongoDB	MySQL	NebulaGraph	neo4j
Apache Spark	Apache Storm	Apache Flink	Apache Beam	Apache NiFi	Apache Beam	Apache Strimzi	Apache Heron	Apache RocketMQ	Apache Flink	Apache Beam	Apache NiFi	Apache Strimzi
Apache Flink	Apache Beam	Apache NiFi	Apache NiFi	Apache Beam	Apache NiFi	Apache Strimzi	Apache Heron	Apache RocketMQ	Apache Flink	Apache Beam	Apache NiFi	Apache Strimzi

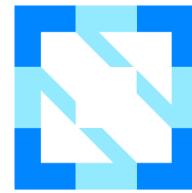
Governance & Community



Operates under a transparent, vendor-neutral governance model, ensuring
that projects are developed collaboratively and without undue influence
from any single organization



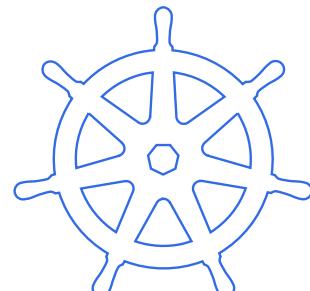
Impact & Adoption

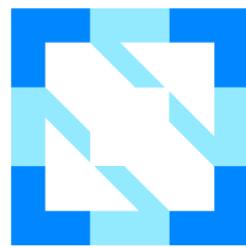


**CLOUD NATIVE
COMPUTING FOUNDATION**

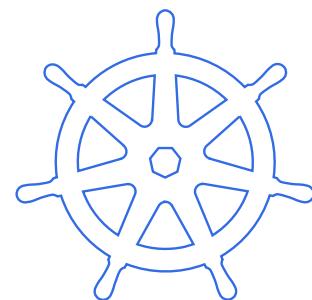


kubernetes





**CLOUD NATIVE
COMPUTING FOUNDATION**

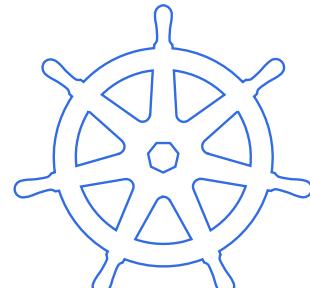


Section: 1

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

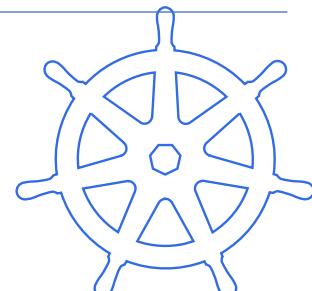
Data Centre Infrastructure

Special buildings that hold
the physical servers



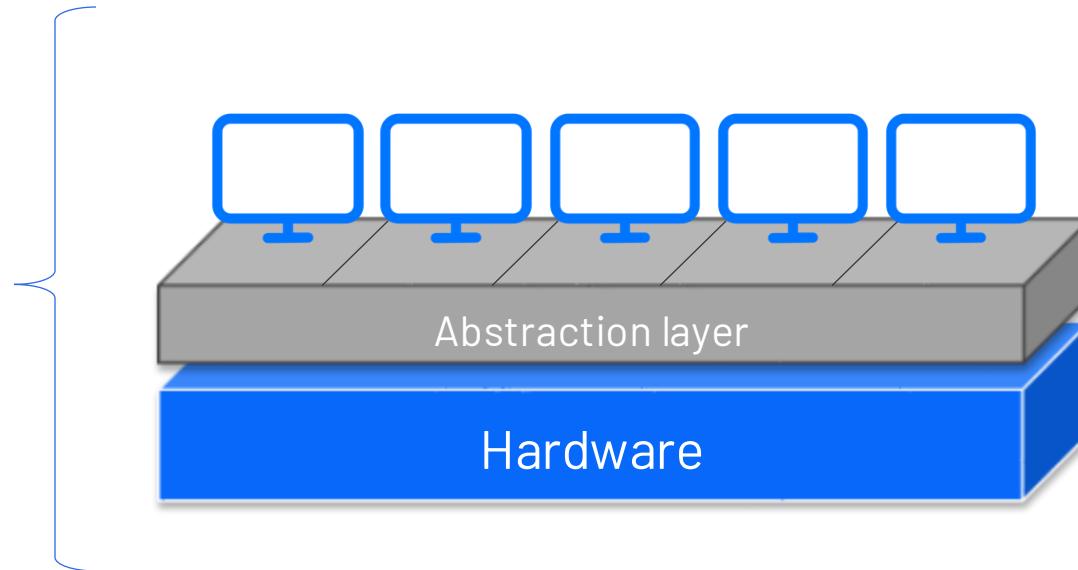
Data Centre

Servers were underutilized
and resulting in wasted
resources



Virtualization

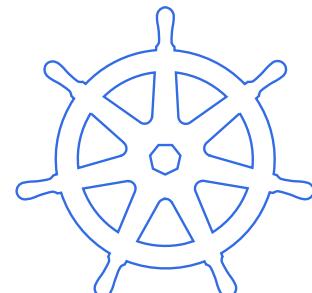
Improves resource utilization by running multiple workloads



- Processors
- Memory
- Storage

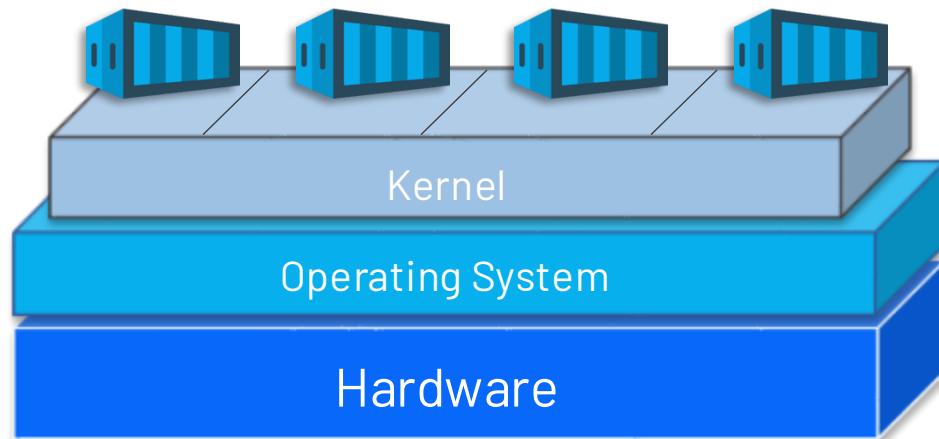


- Time consuming
- Resource intensive

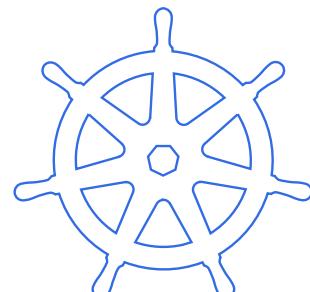


Containers

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

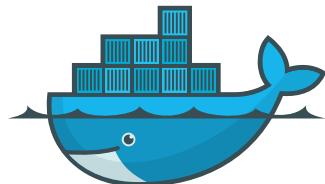


- ✓ Reduce deployment time & resource requirements
- ✓ Run containers in any cloud infrastructure



Container Runtime Tools

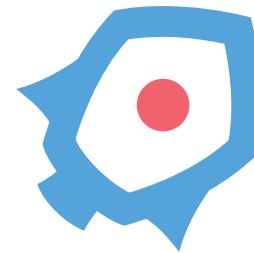
Container Runtime Tools



Docker



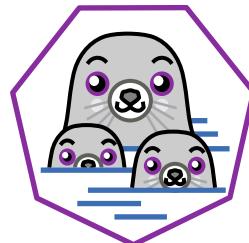
containerd



rkt



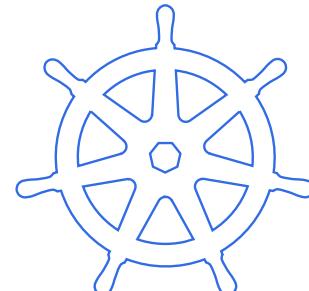
cri-o



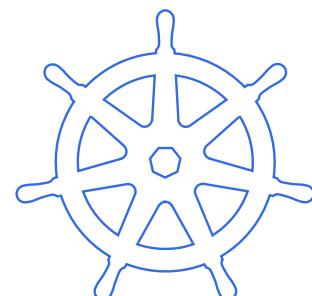
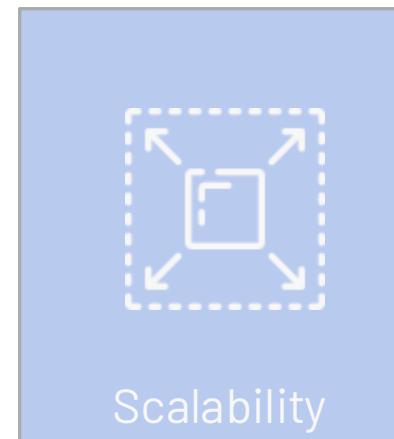
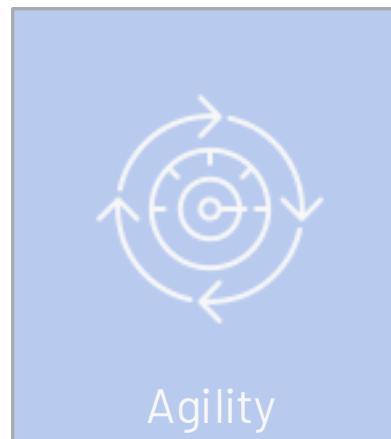
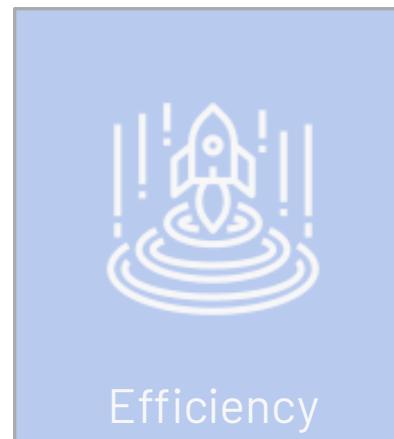
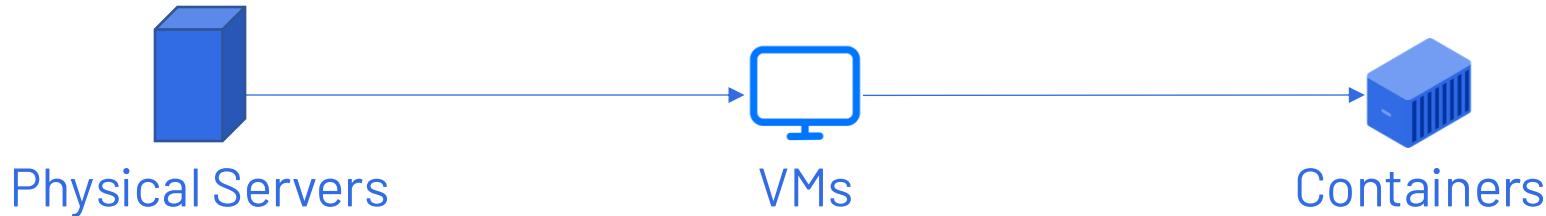
podman



Kata Container



Evolution of Containers



VMs vs Containers

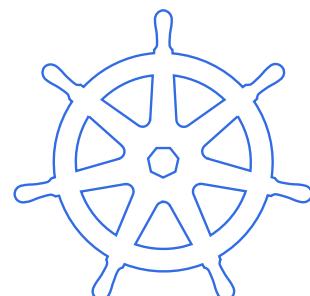
Feature	Virtual Machines(VMs)	Containers
Virtualization	Hardware Virtualization	Operating System Virtualization
Definition	Full operating system environment running on a hypervisor	Lightweight, portable, and efficient way to package and run applications
Relevance	Suitable for workloads requiring a full 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



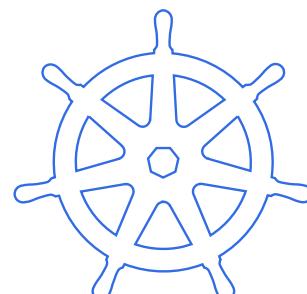
Containers provide lightweight, portable, and efficient way to package and run applications



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:

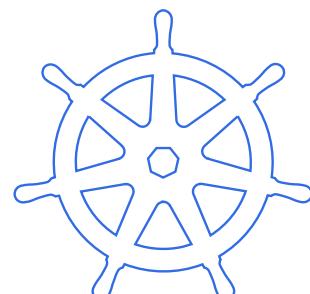
Mitigating Limitations with Container Orchestrator



Containers

Container Orchestration

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



Container Orchestration

Container Orchestration

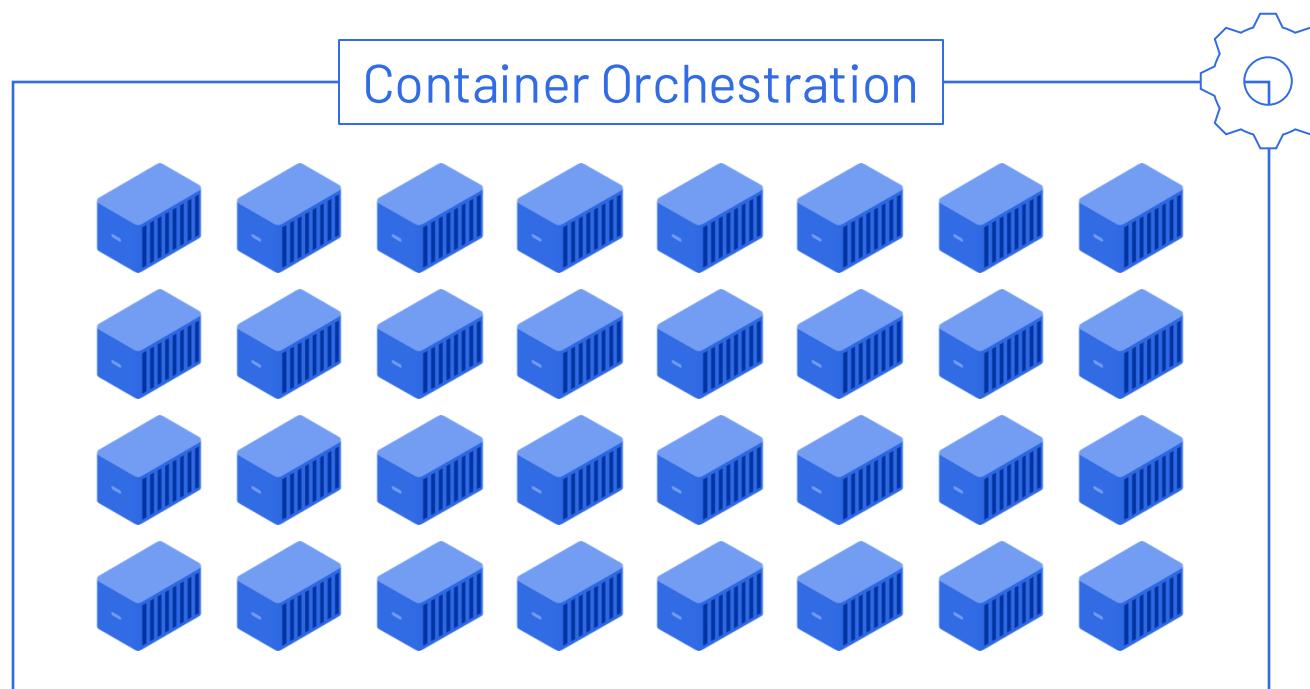


Agility
Efficiency

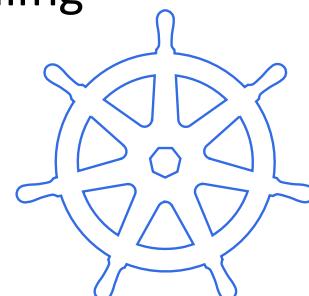


Manages lifecycle
of containerized
applications

Address many of
limitations inherent
to containers



- Automate Tasks like:
- ✓ Deployment
 - ✓ Scaling
 - ✓ Health Checks
 - ✓ Self-healing

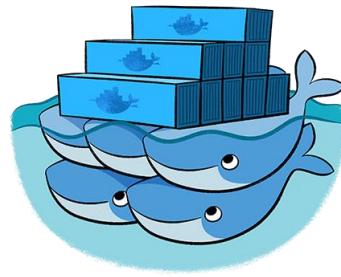


Container Orchestration



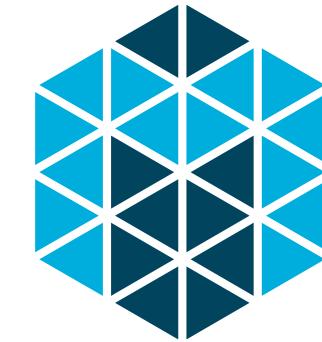
Kubernetes

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



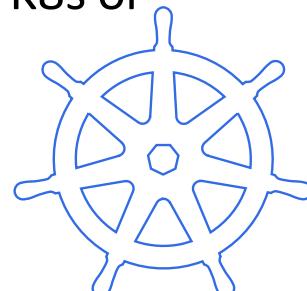
Docker Swarm

- ✓ 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

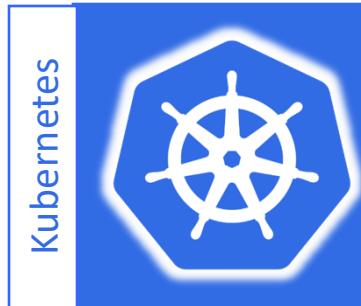
- ✓ Acts as a cluster kernel
- ✓ Flexible and resource optimization
- ✓ Requires more configuration effort compared to K8s or Swarm



Choosing Right Orchestrator

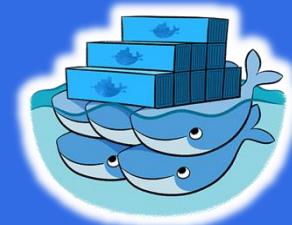
Choosing Right Orchestrator

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



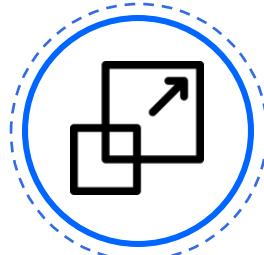
Features of Container Orchestrator

Features of Container Orchestrator

High Availability & Self-Healing



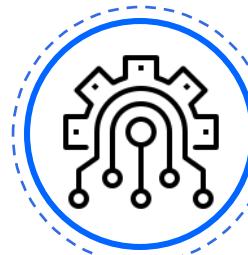
Scaling



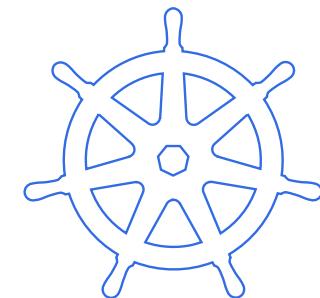
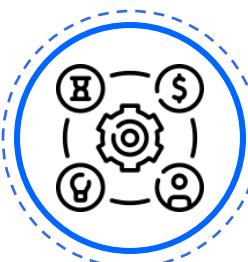
Zero Downtime Application Upgrades



Integration with External Tools



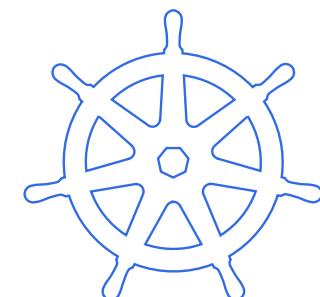
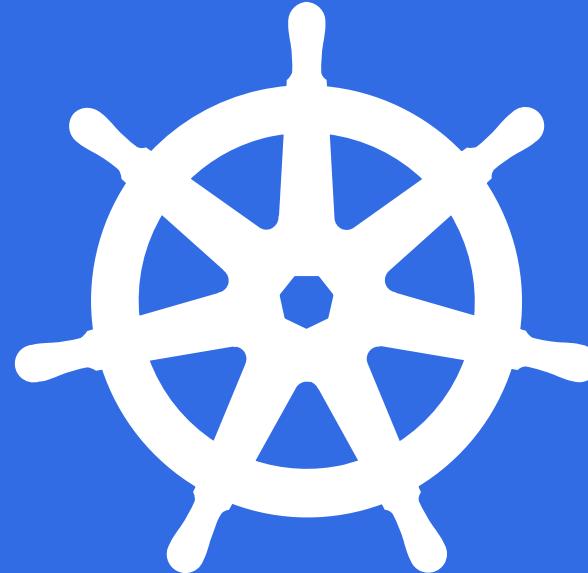
Resource Management & Scheduling



Summary

Summary

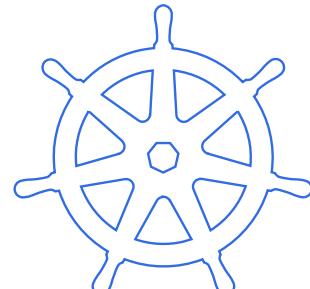
- ✓ Evolution of containers
- ✓ Container limitations
- ✓ How orchestrators address container limitations
- ✓ Key features of orchestrators



Kubernetes at a Glance

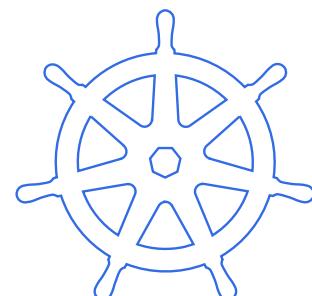
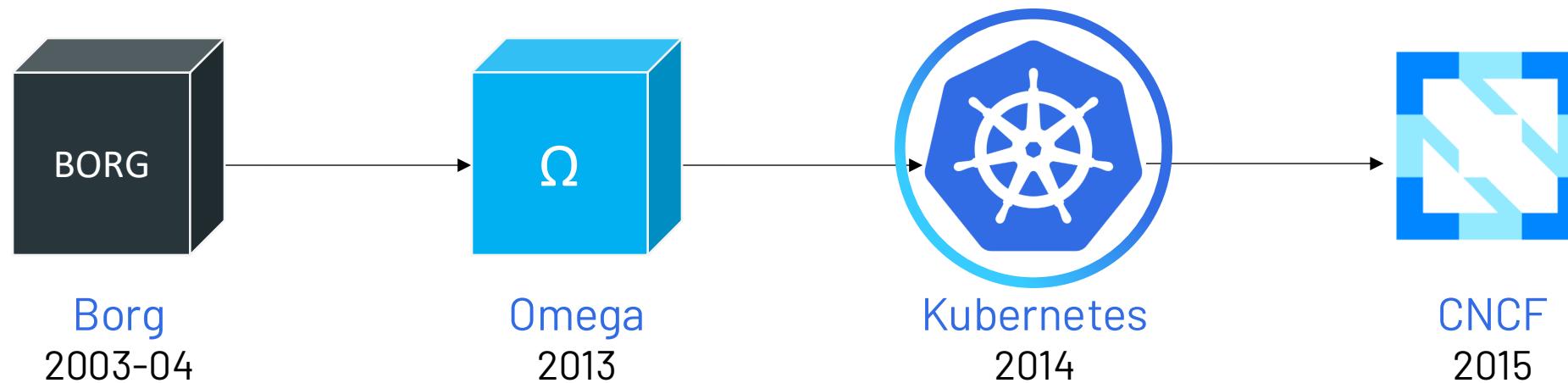
Section Overview

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



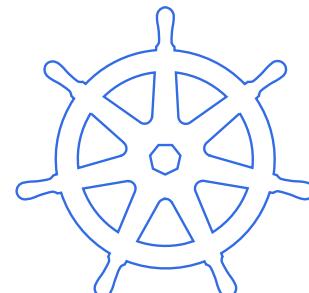
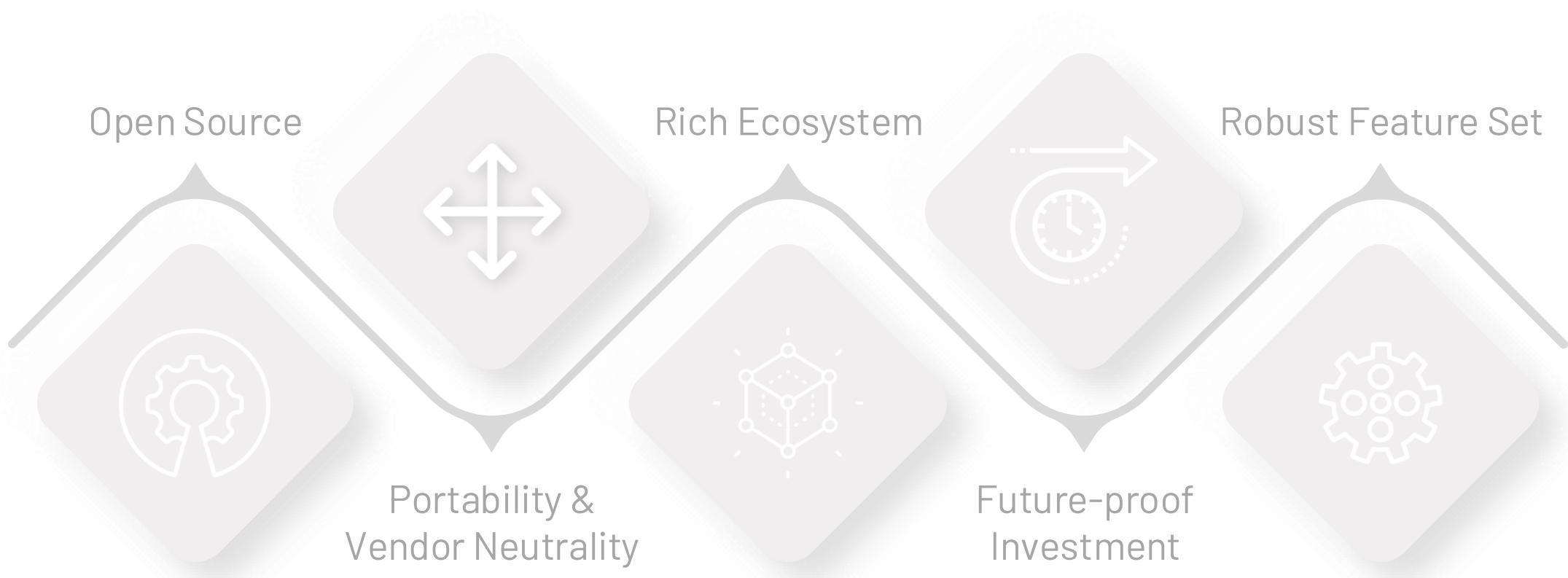
Introduction to Kubernetes

Introduction to Kubernetes

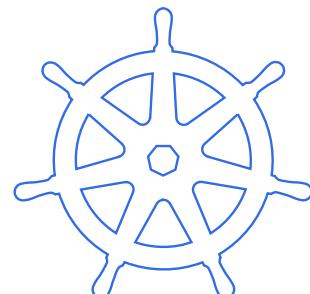


Why Kubernetes?

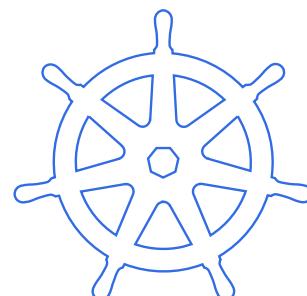
Why Kubernetes?



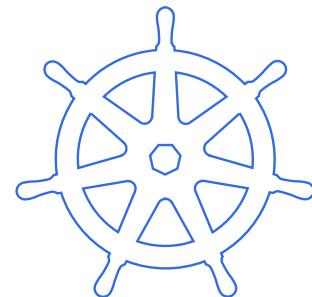
Why Kubernetes?



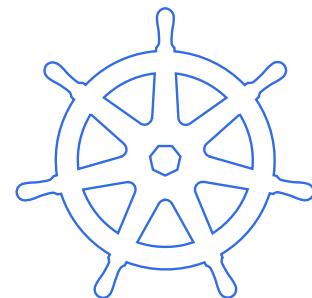
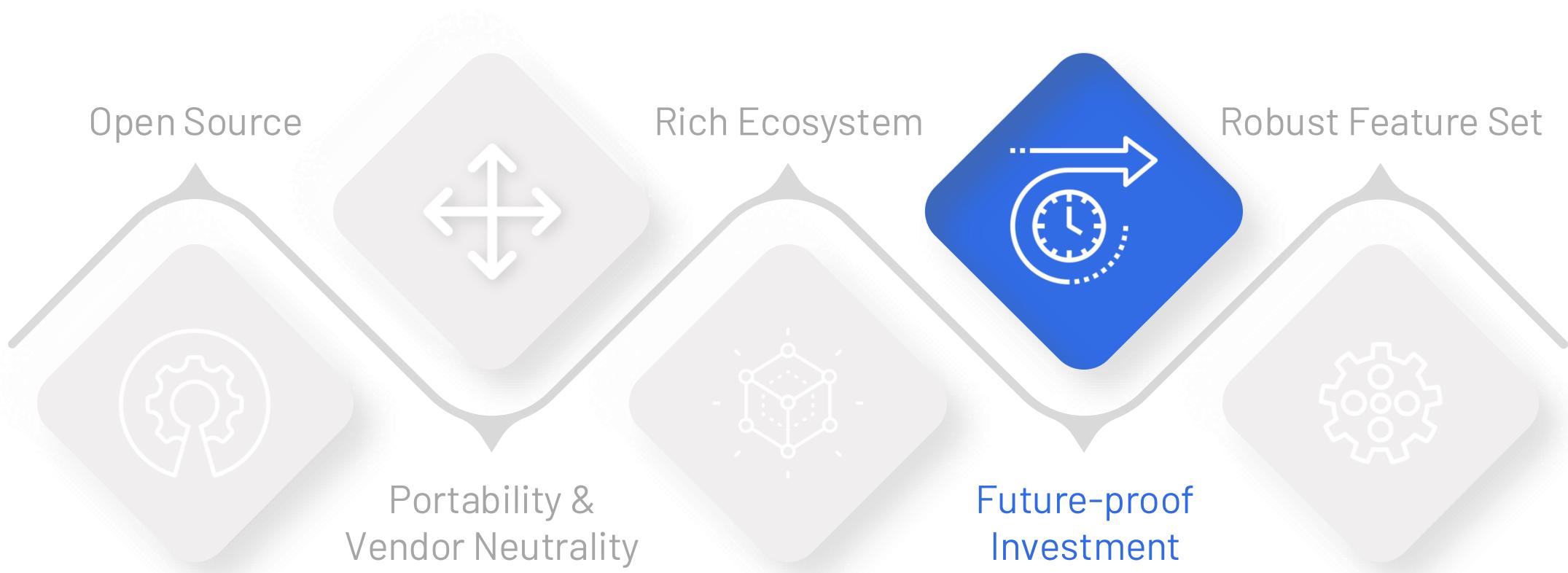
Why Kubernetes?



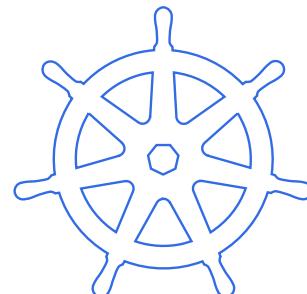
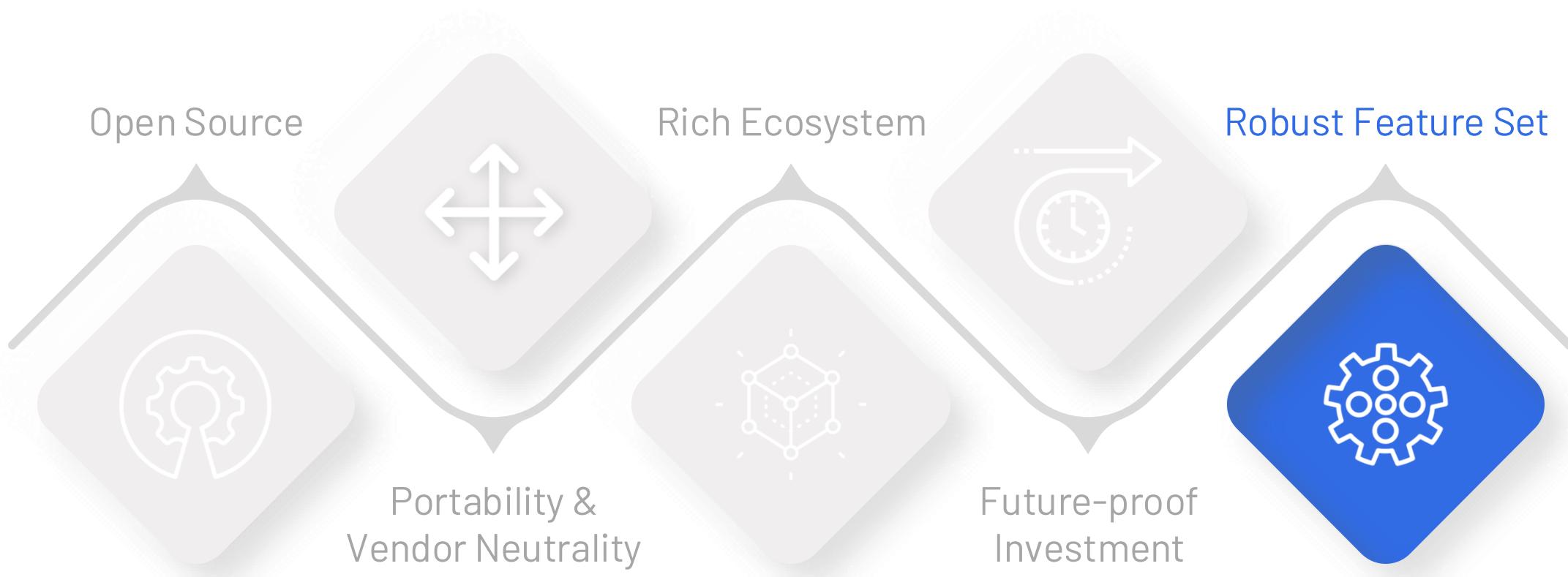
Why Kubernetes?



Why Kubernetes?



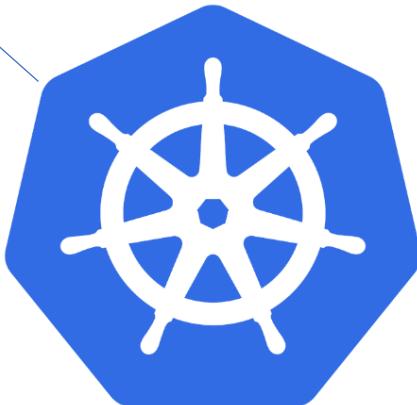
Why Kubernetes?



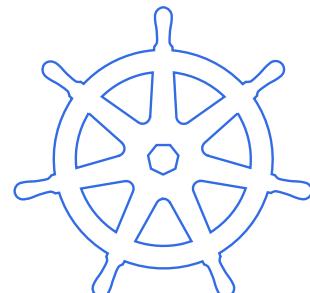
Why Kubernetes?

Modular Architecture

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



Offers compelling combination of features, flexibility, and community support



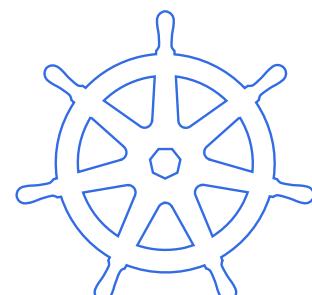
Setup Options

Setup Options

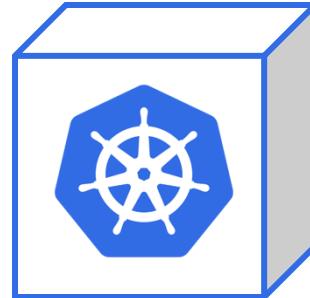


Native Kubernetes

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

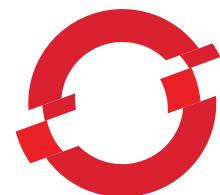


Setup Options

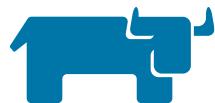


Enterprise Kubernetes

- ✓ 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



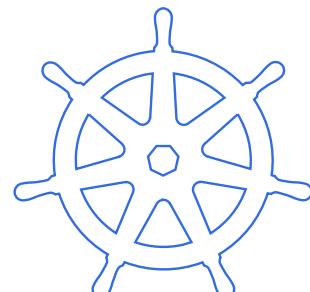
Rancher



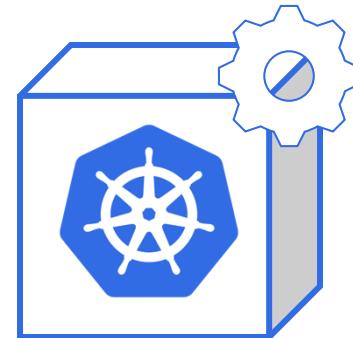
Tanzu



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



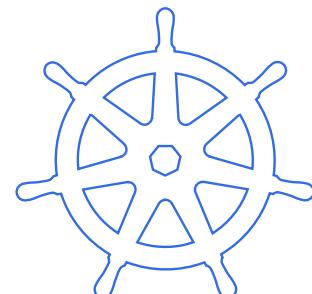
EKS



AKS

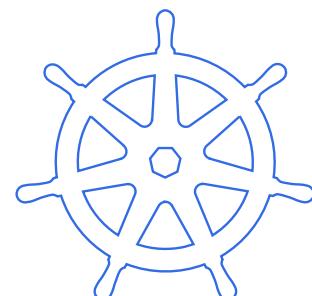
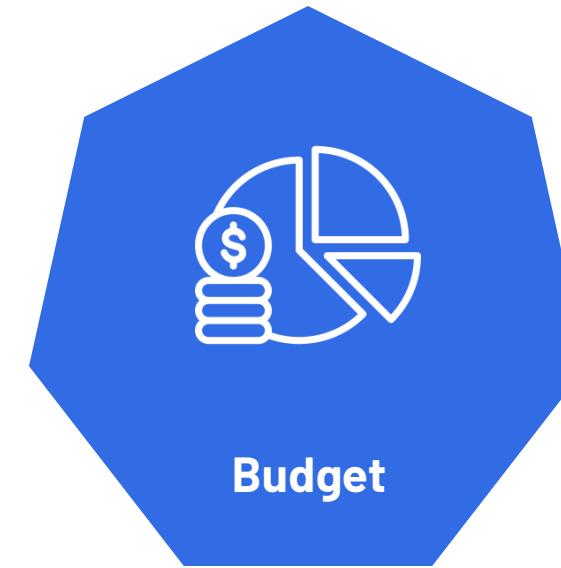
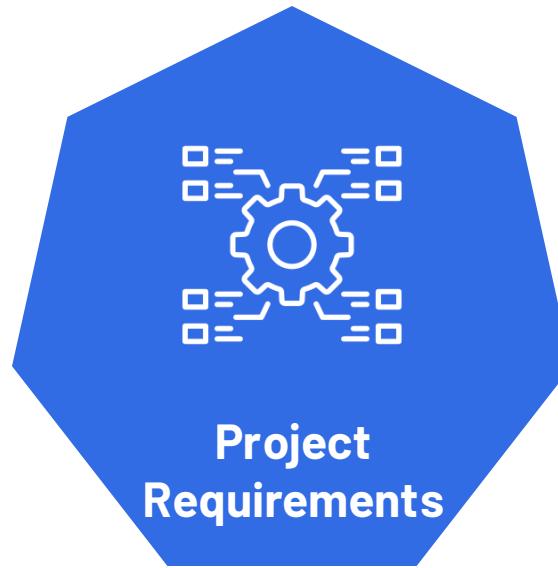


GKE



Choosing the Right Option

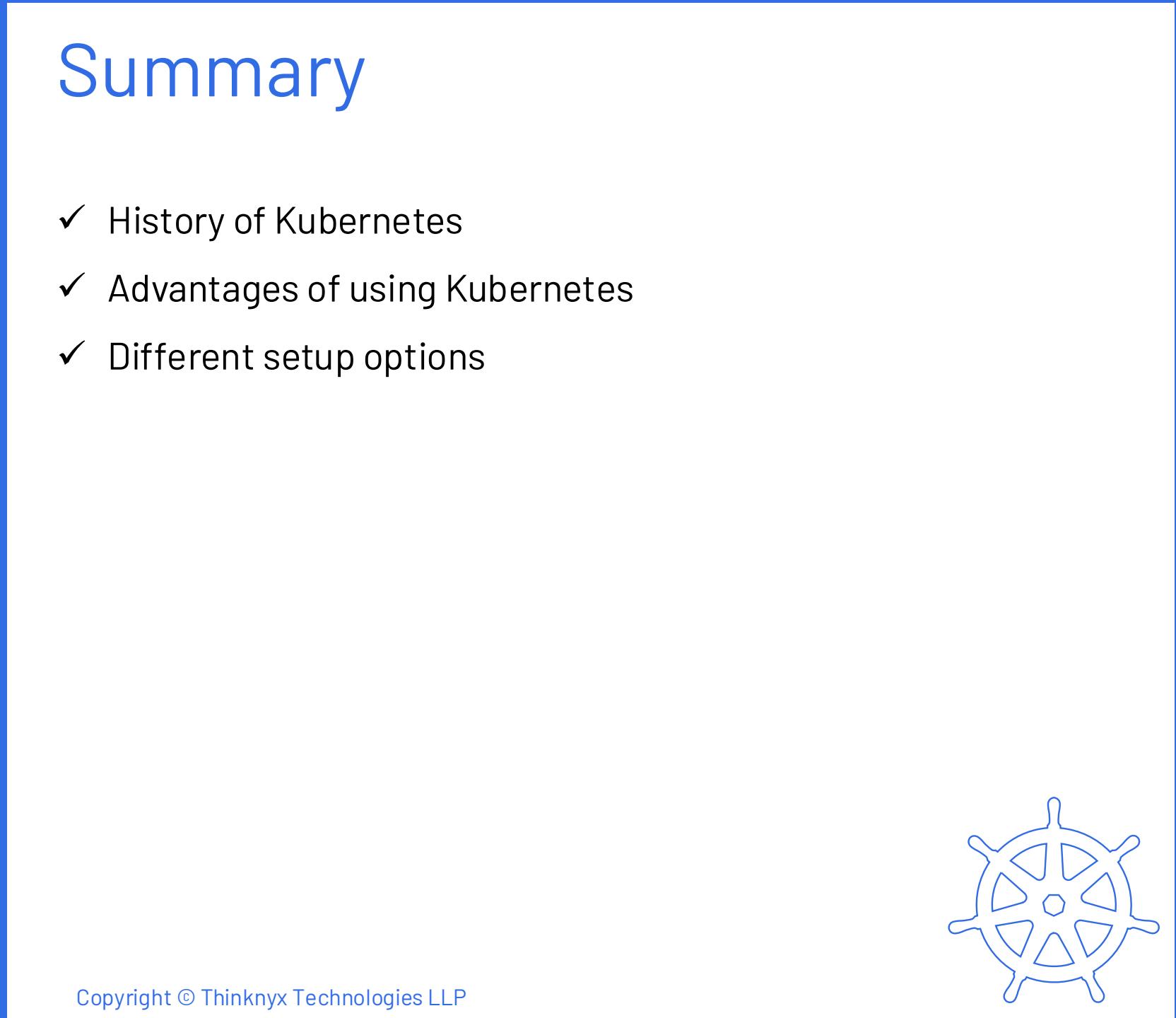
Choosing the Right Option



Summary



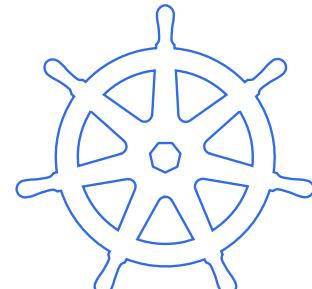
Summary

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

Understanding Kubernetes Architecture

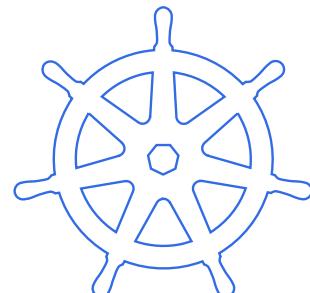
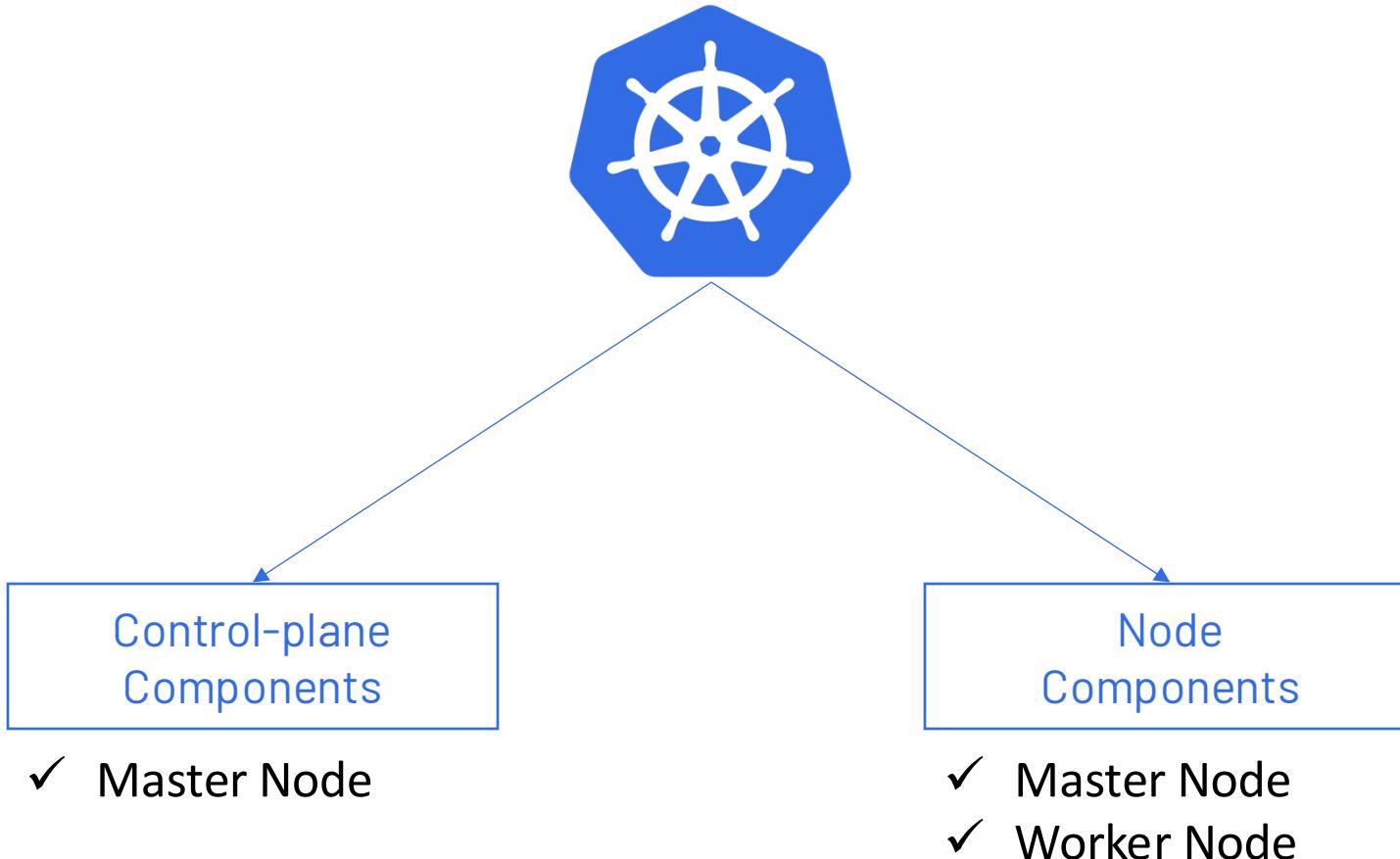
Section Overview

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



Overview of Kubernetes Components

Overview of Kubernetes Components



Control-plane Components

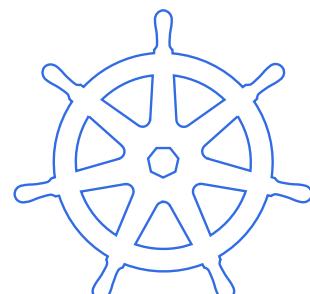
Control-plane Components



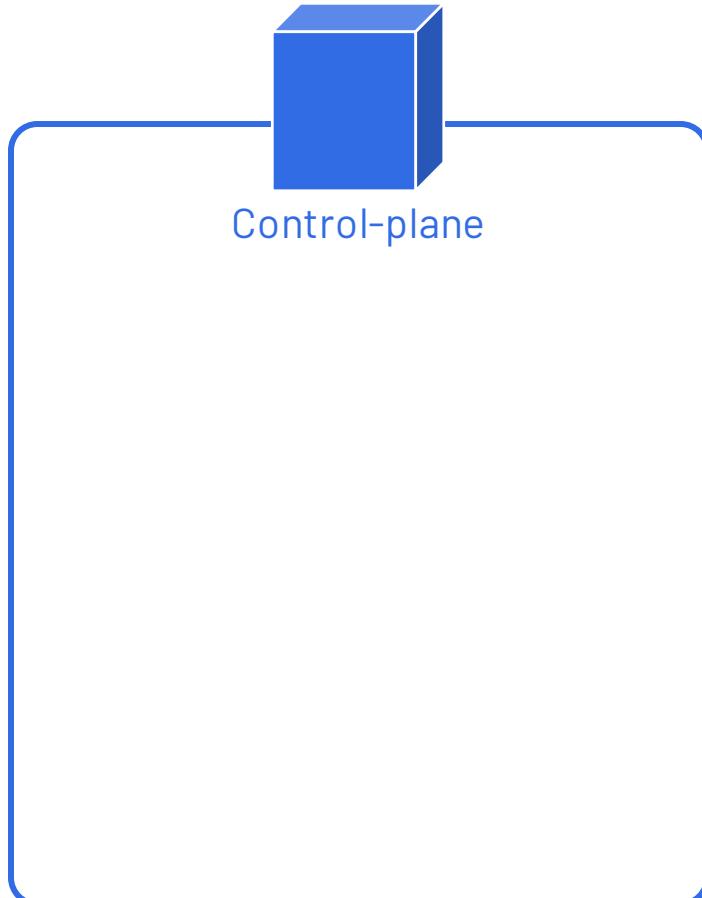
Control-plane

Central nervous system of the cluster

Manage, plan, schedule and monitor nodes and application containers

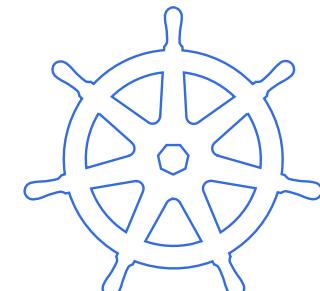


Control-plane Components

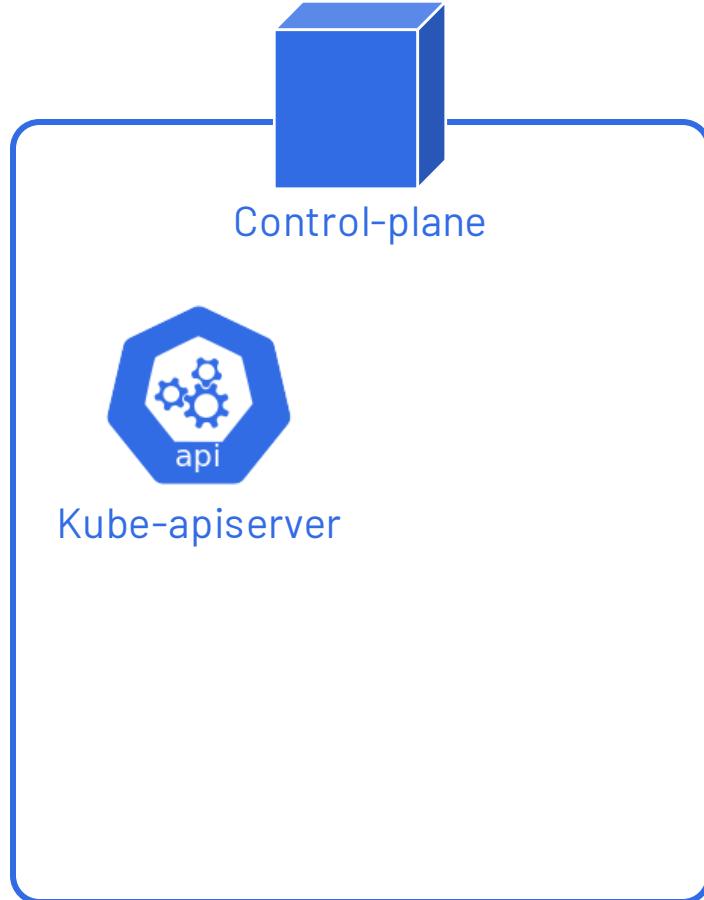


Kube-apiserver

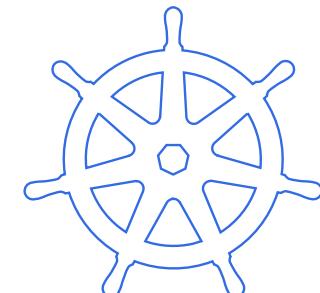
- ✓ Acts as a central point of communication & management
- ✓ Responsible for exposing K8s API's & orchestrating all operations within the cluster
- ✓ Serves as single point of entry for all user requests & interactions with the cluster



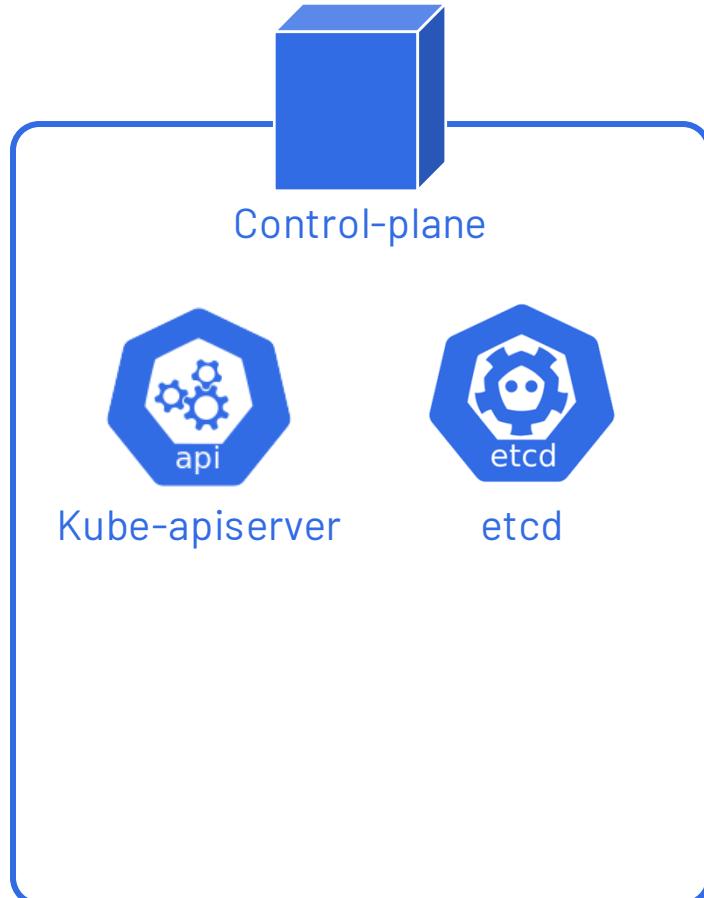
Control-plane Components



- ✓ 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

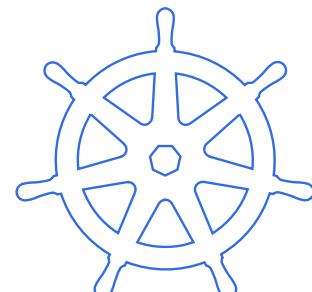


Control-plane Components

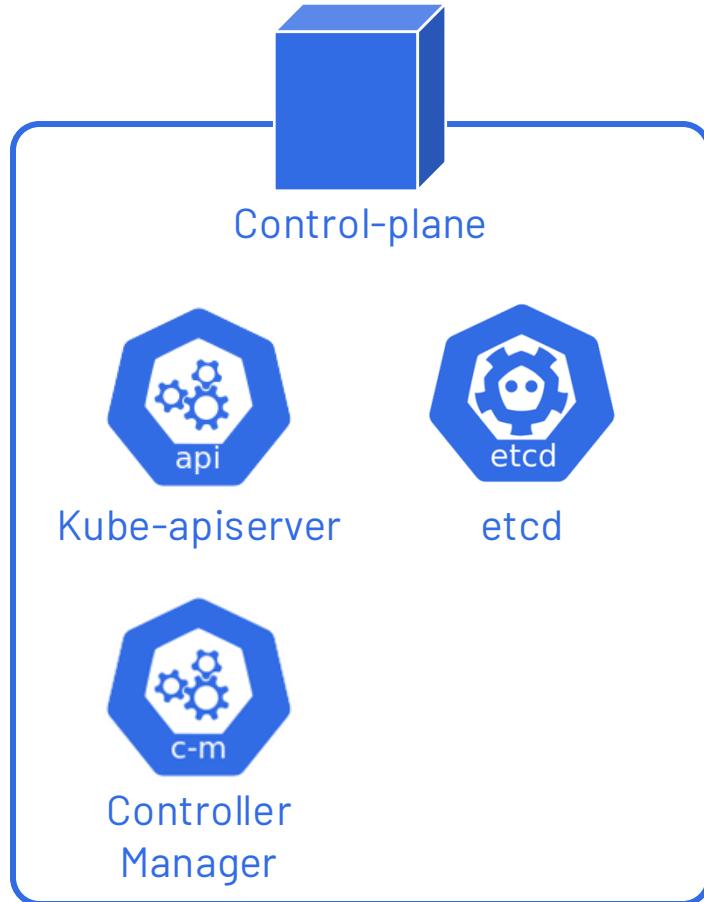


Controller Manager

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

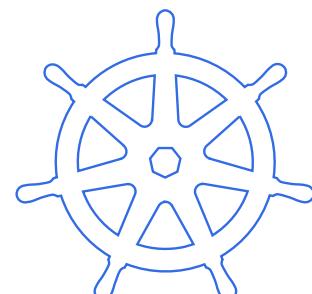


Control-plane Components

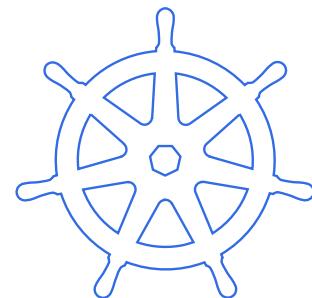
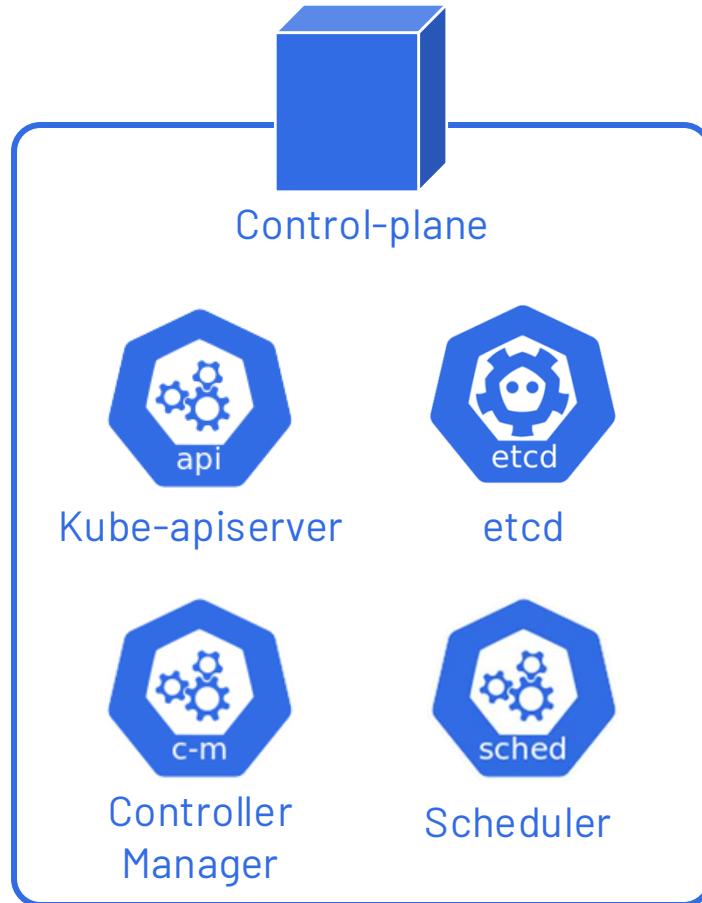


Scheduler

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

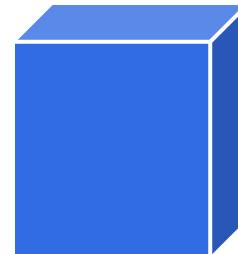


Control-plane Components



Node Components

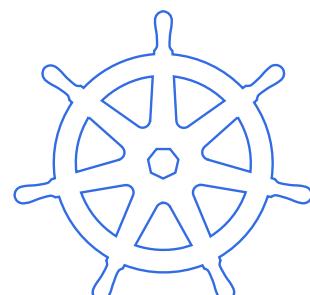
Node Components



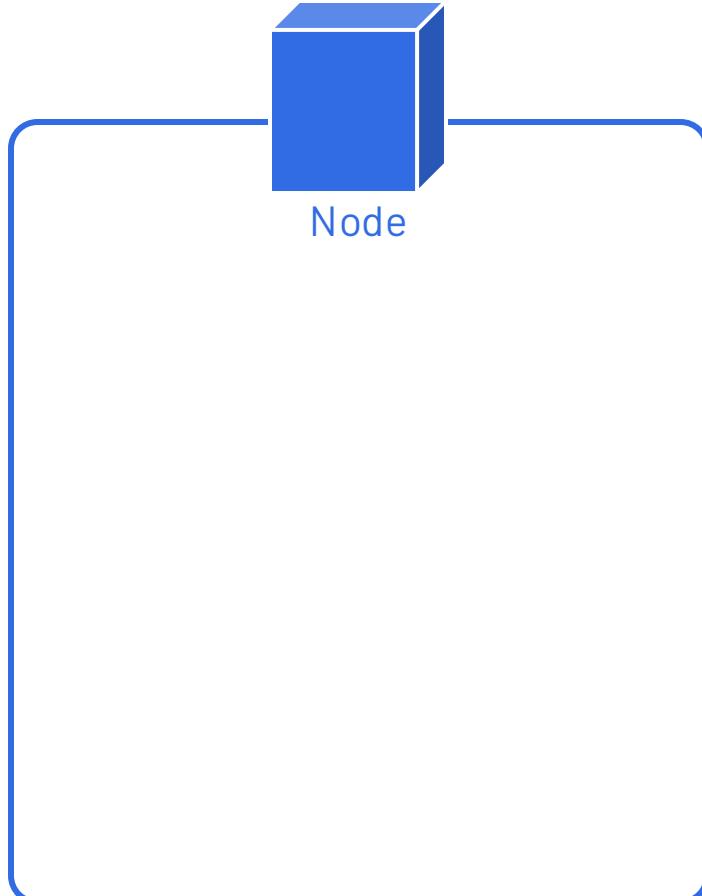
Node

Node components reside on every node in a cluster

Act as the workhorses that execute containerized applications

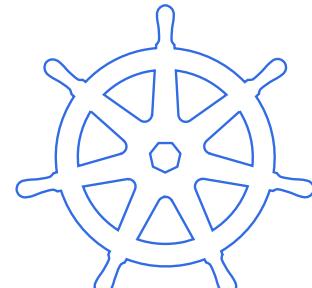


Node Components

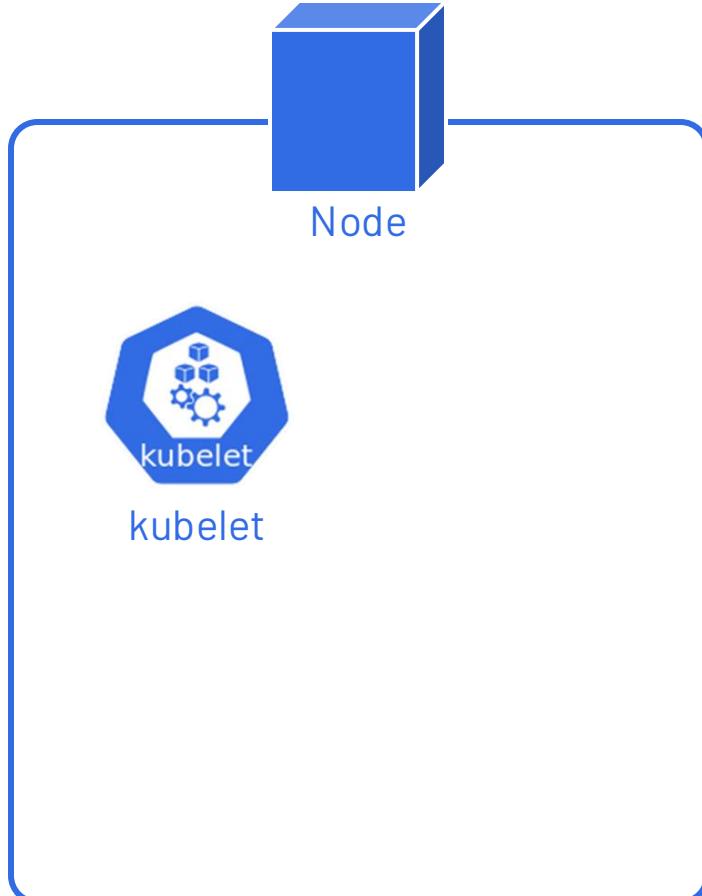


kubelet

- ✓ 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

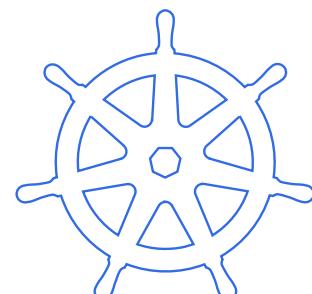


Node Components

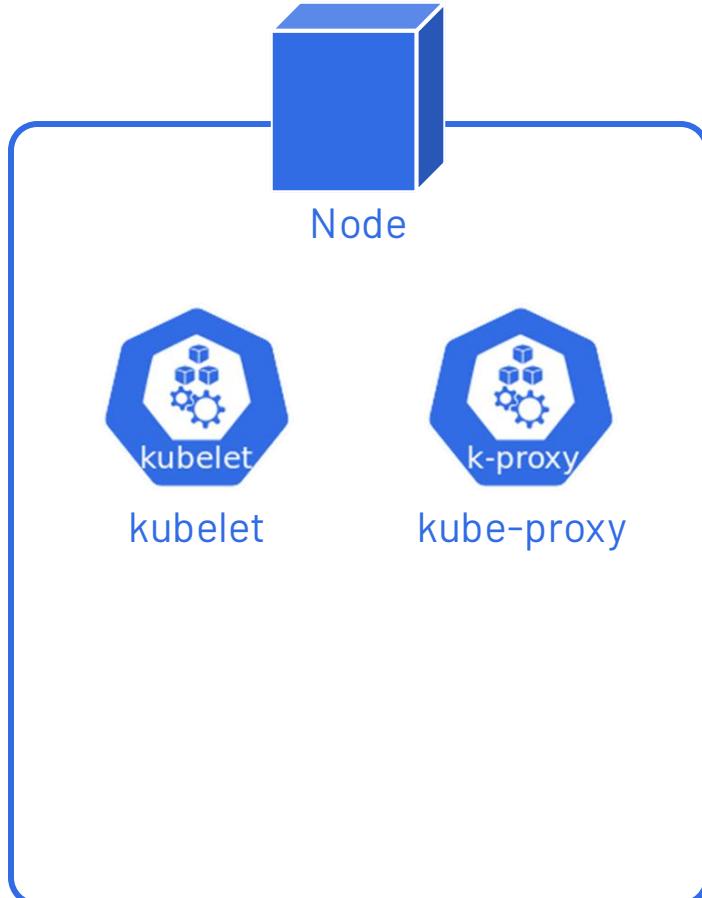


kube-proxy

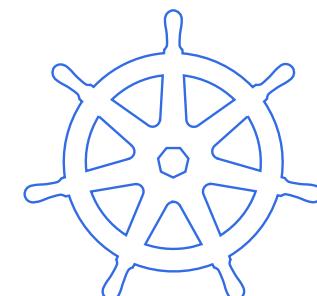
- ✓ Acts as the network traffic director
- ✓ Ensures pods can communicate with each other seamlessly
- ✓ Facilitates service discovery, allowing pods to locate & connect with services running within the cluster



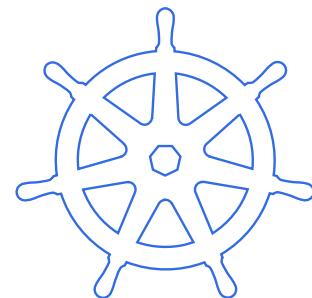
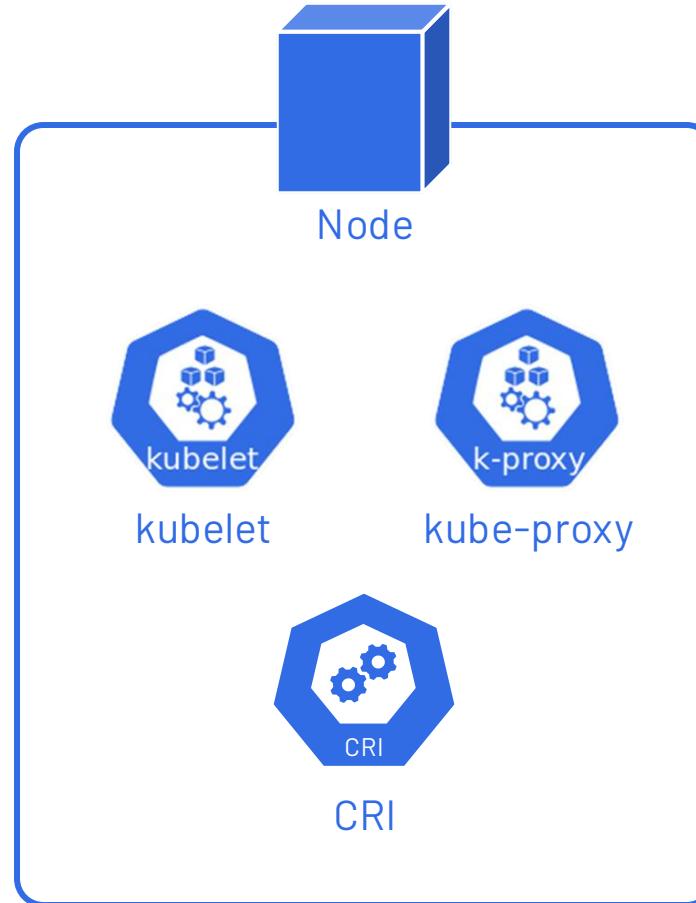
Node Components



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



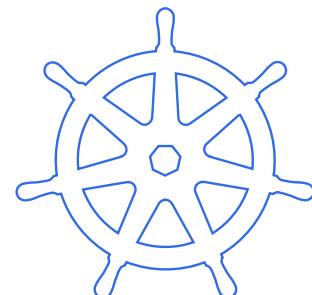
Node Components



Addons



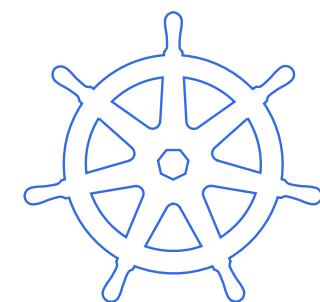
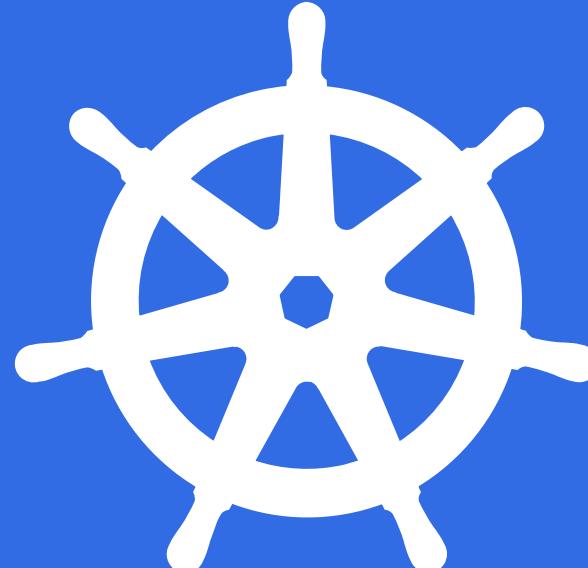
- ✓ Network Plugins
- ✓ Service discovery addons
- ✓ State metrics
- ✓ Kubernetes dashboard



Summary

Summary

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

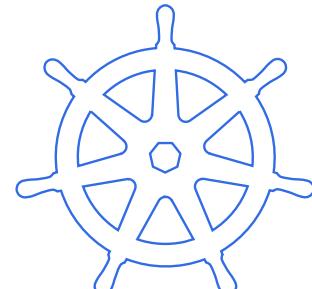


Section: 4

Installation of Kubernetes

Section Overview

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



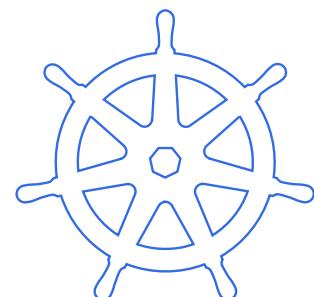
Two Node Cluster



Control Plane/ Master Node



Worker Node



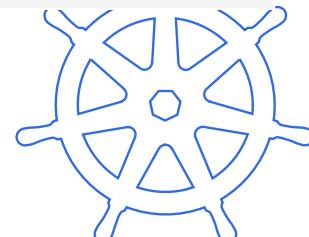
Pre-Requisites (Master Node)

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



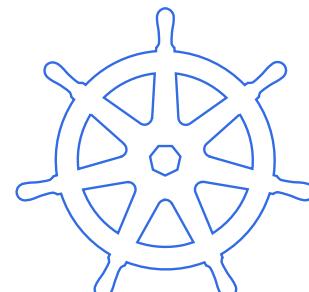
Pre-Requisites (Worker Node)

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

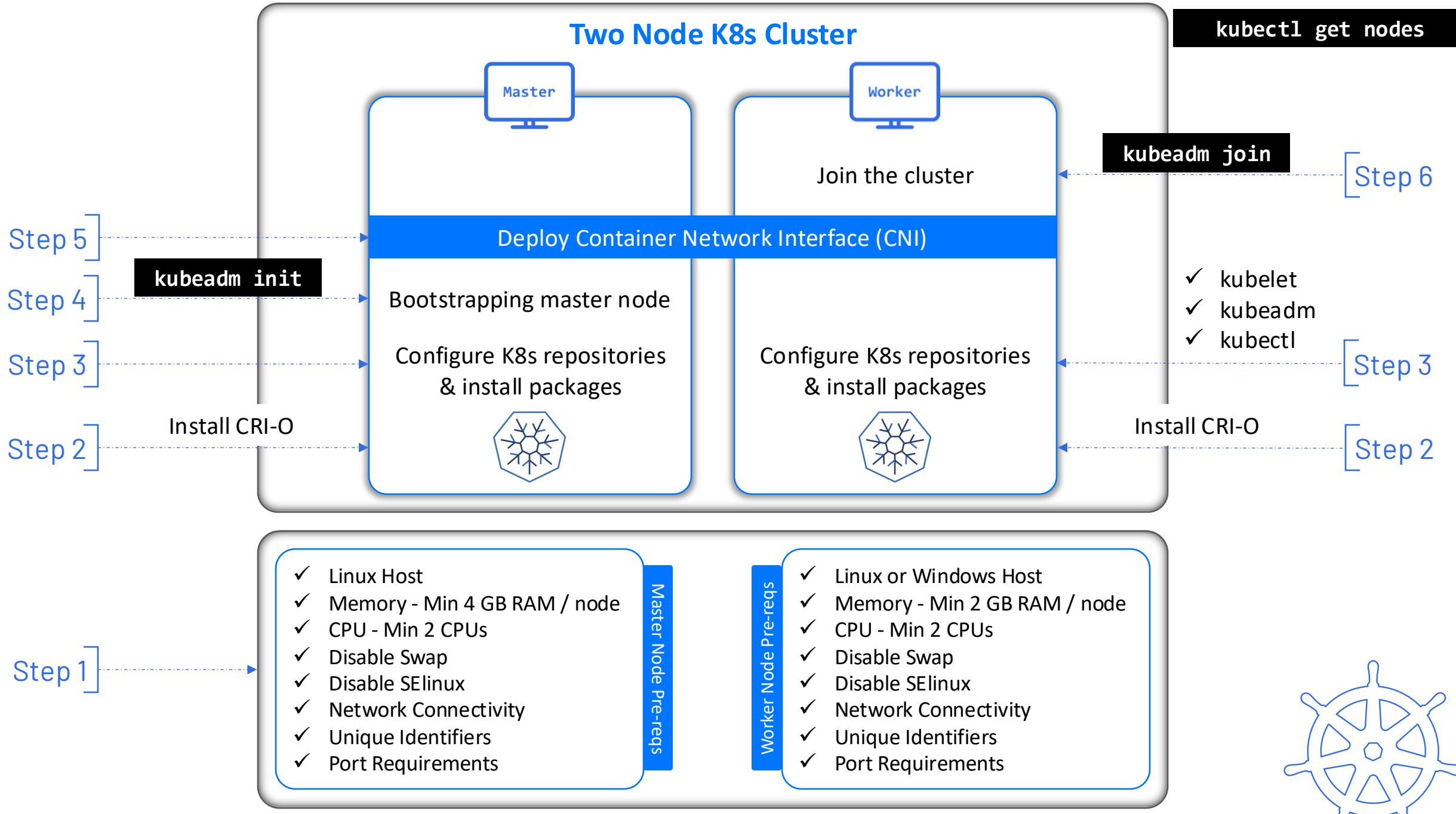


Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10256	kube-proxy	Self, Load balancers
TCP	Inbound	30000-32767	NodePort Services	All



Setting up Two-node Cluster

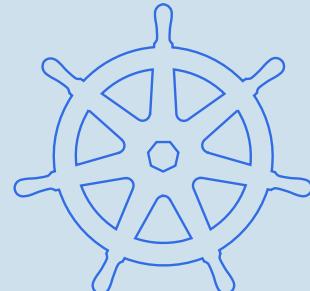
Two Node K8s Cluster





Demo

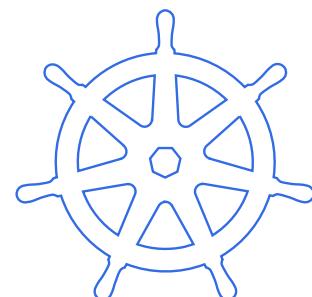
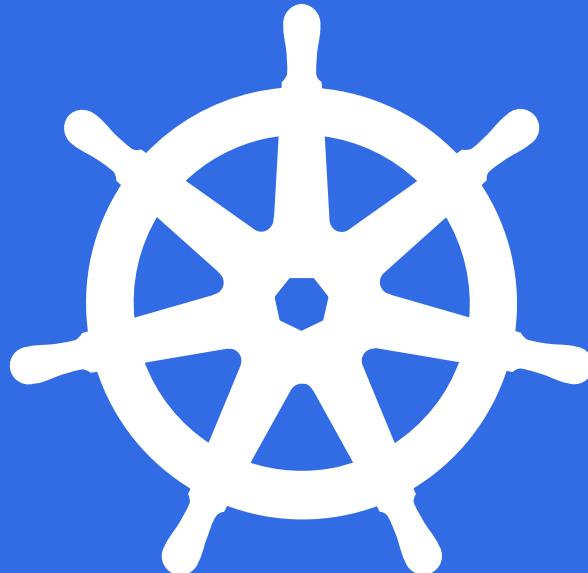
Two Node Cluster Setup



Summary

Summary

- ✓ Two-node Kubernetes cluster installation

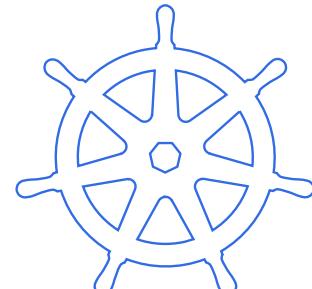


Section: 2

Section Overview

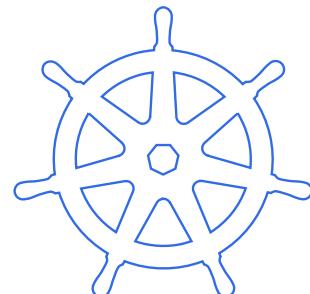
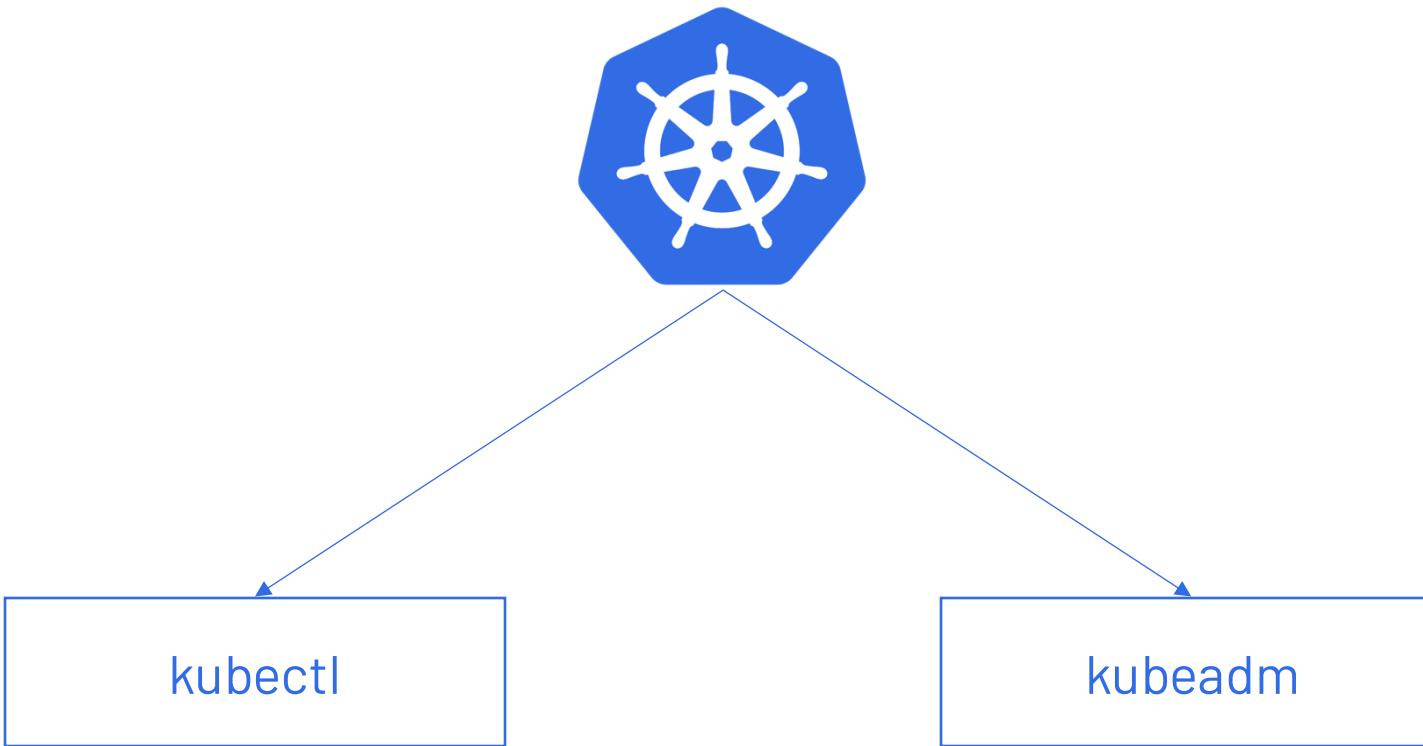
Kubernetes Objects

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



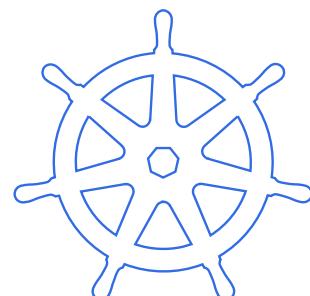
Kubernetes CLI Overview

Kubernetes CLI Overview



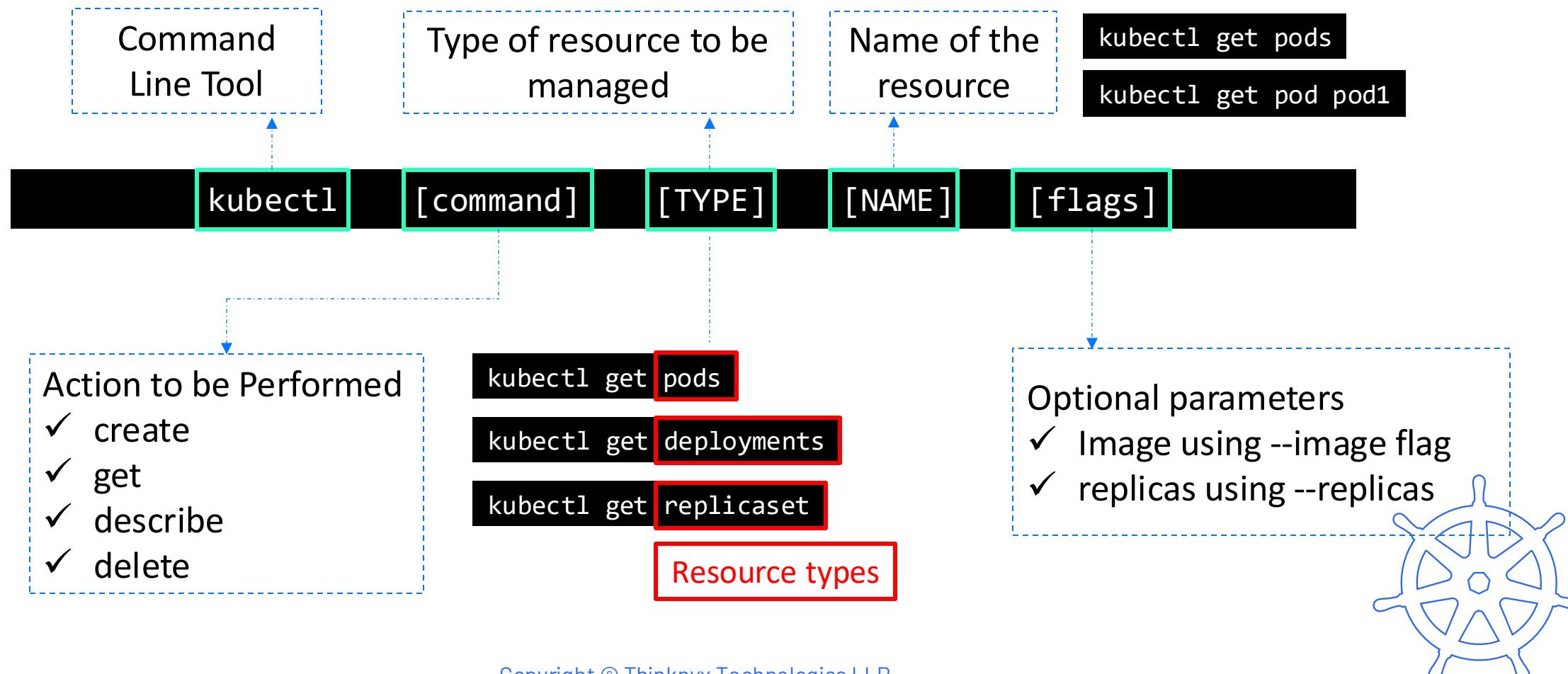
kubectl

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

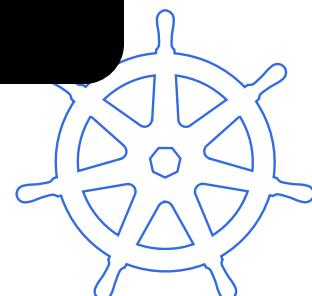
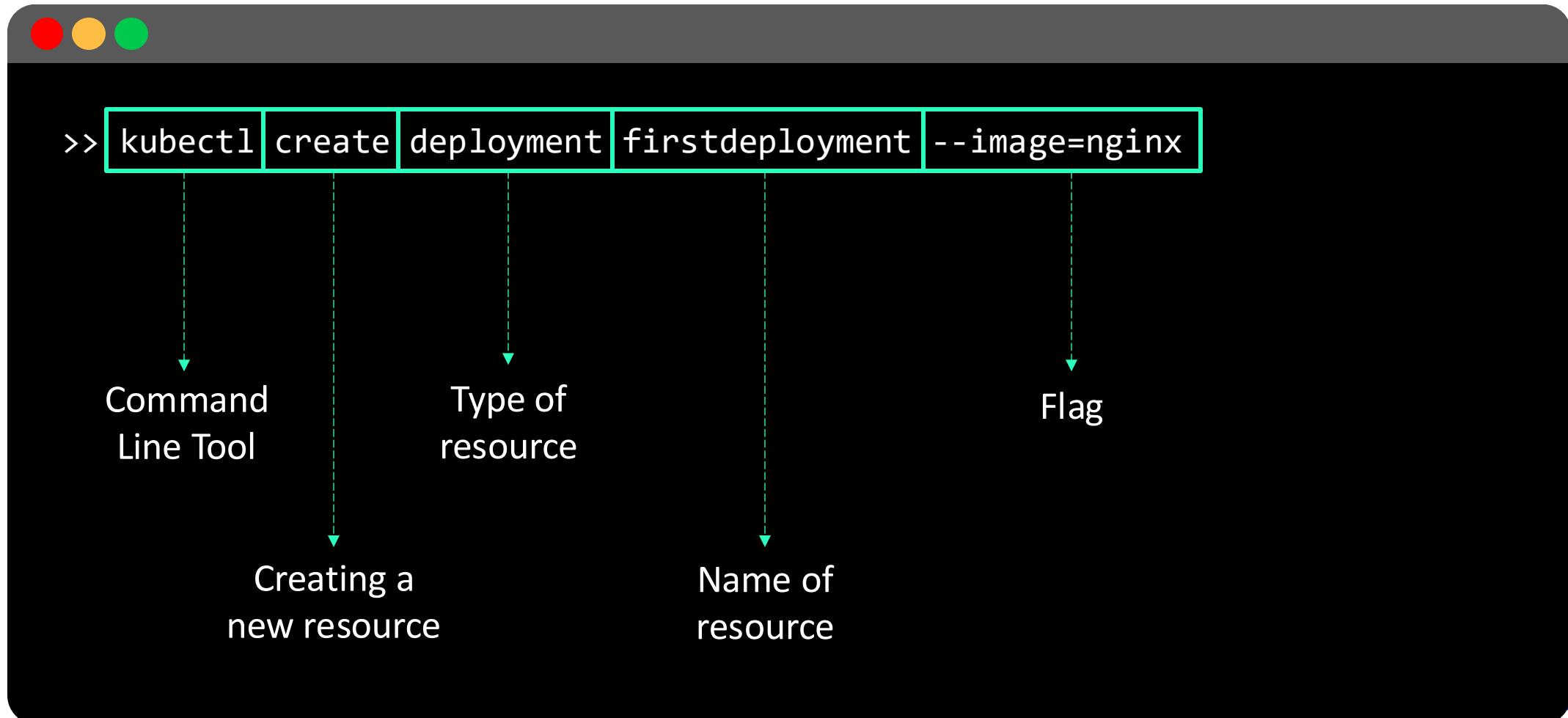


kubectl

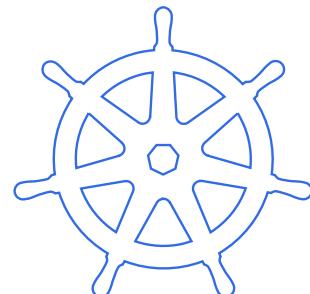
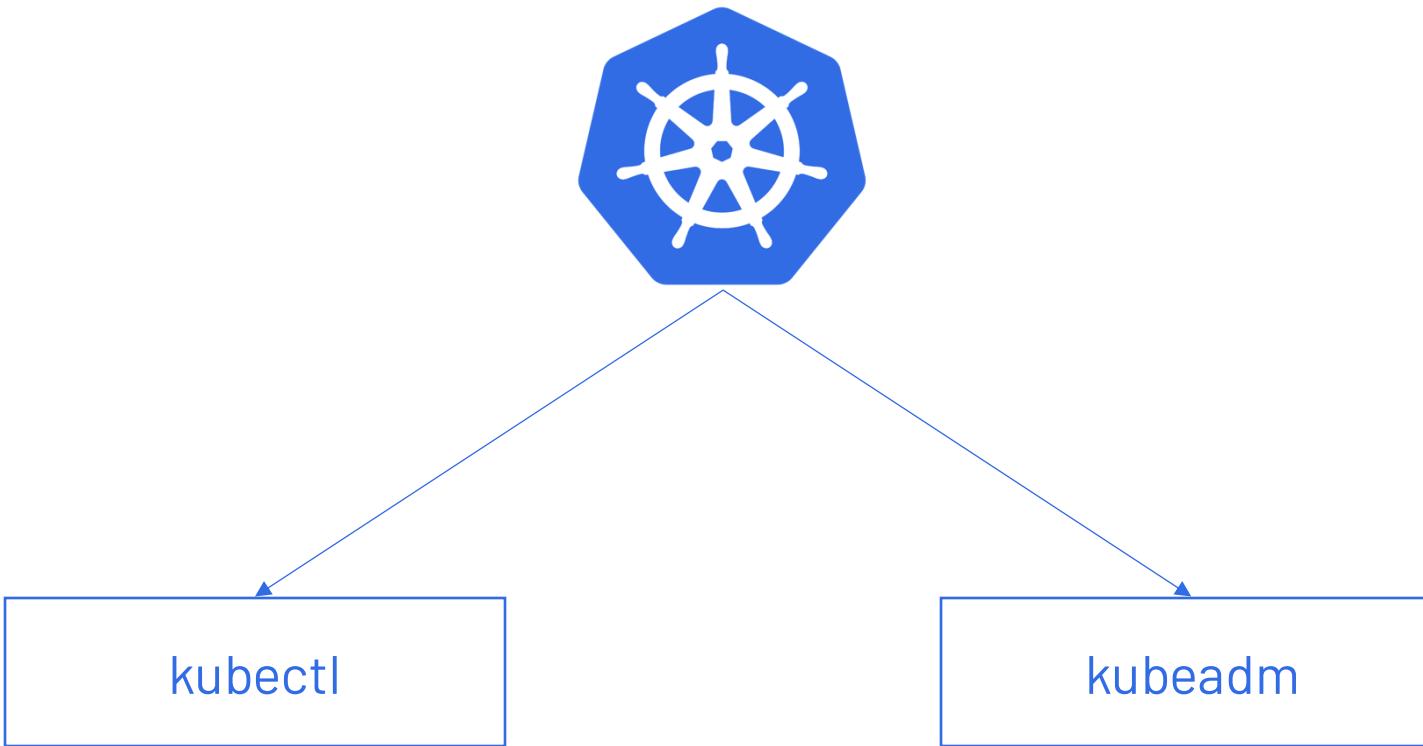
Basic syntax of kubectl command



kubectl

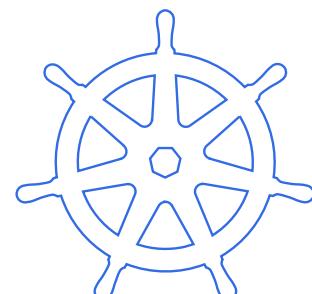


Kubernetes CLI Overview



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

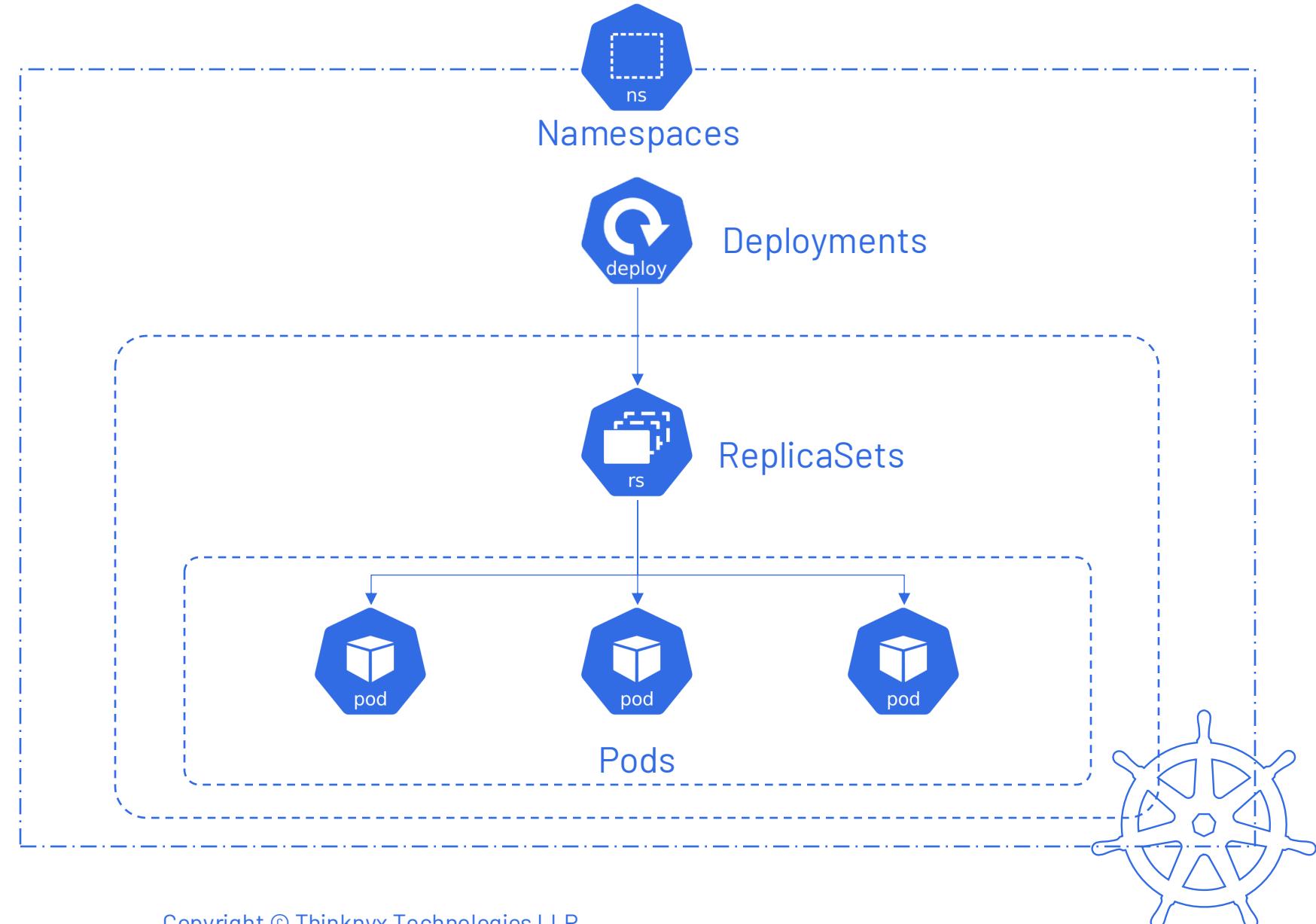
Kubernetes Objects

Provides a way to create logical partitions within a cluster

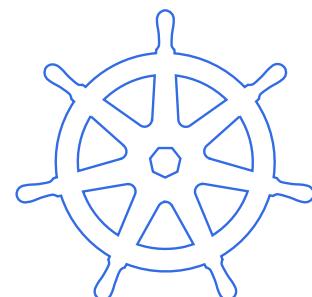
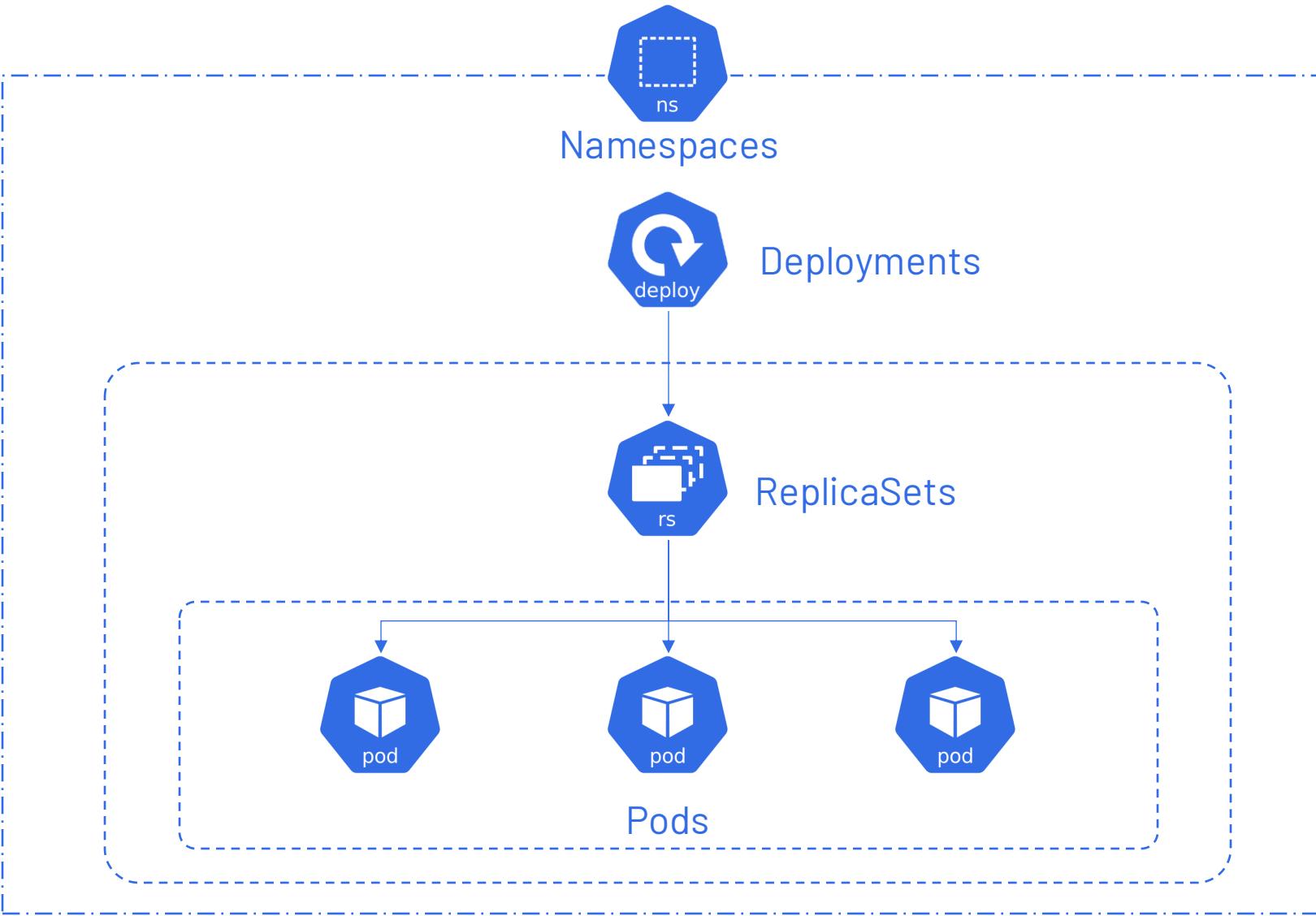
Manage Pod's lifecycle & provide advanced capabilities for application management

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

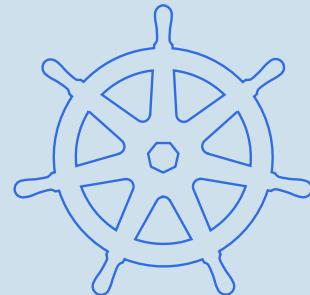
Resource Quotas

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

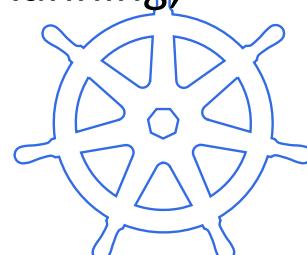


Group of Containers

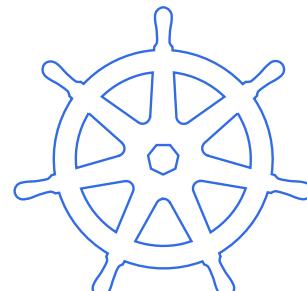
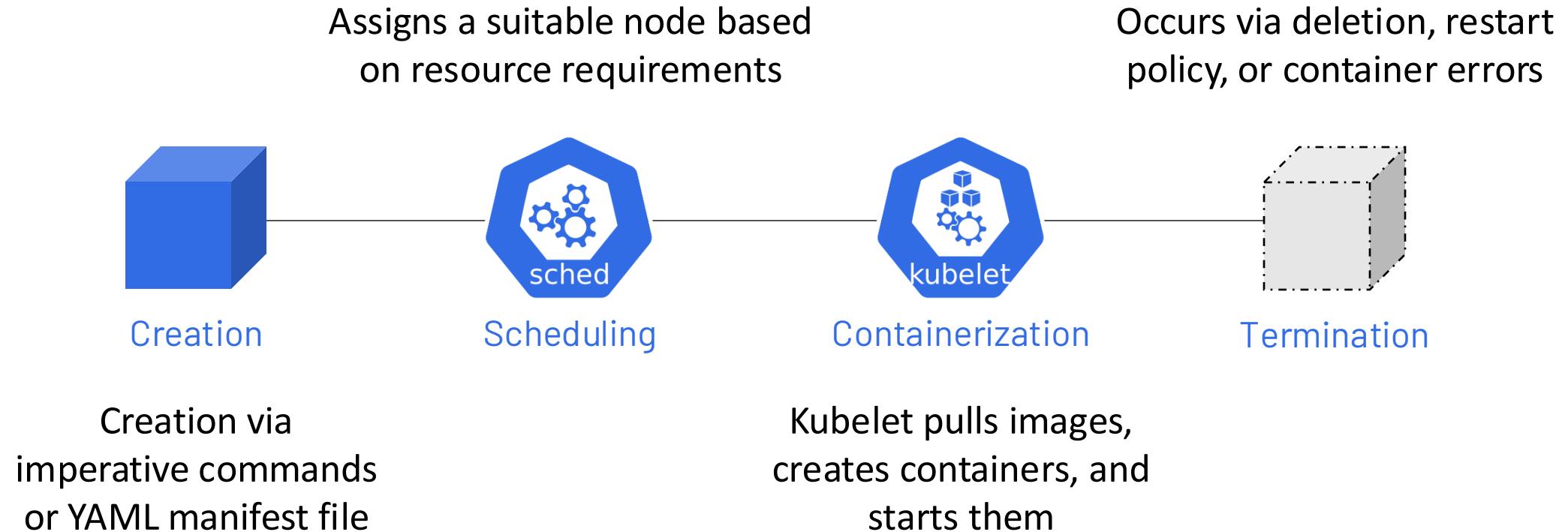
Groups of one or more tightly coupled containers that share the same network namespace and storage

Lifecycle

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

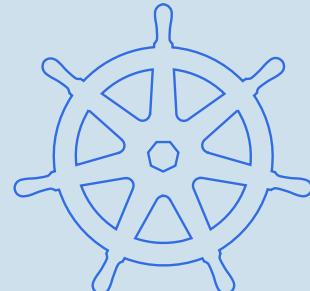


Pods



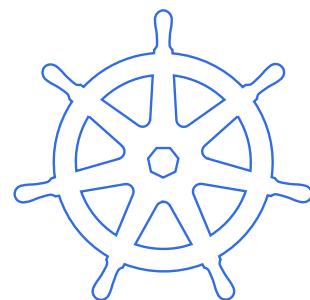
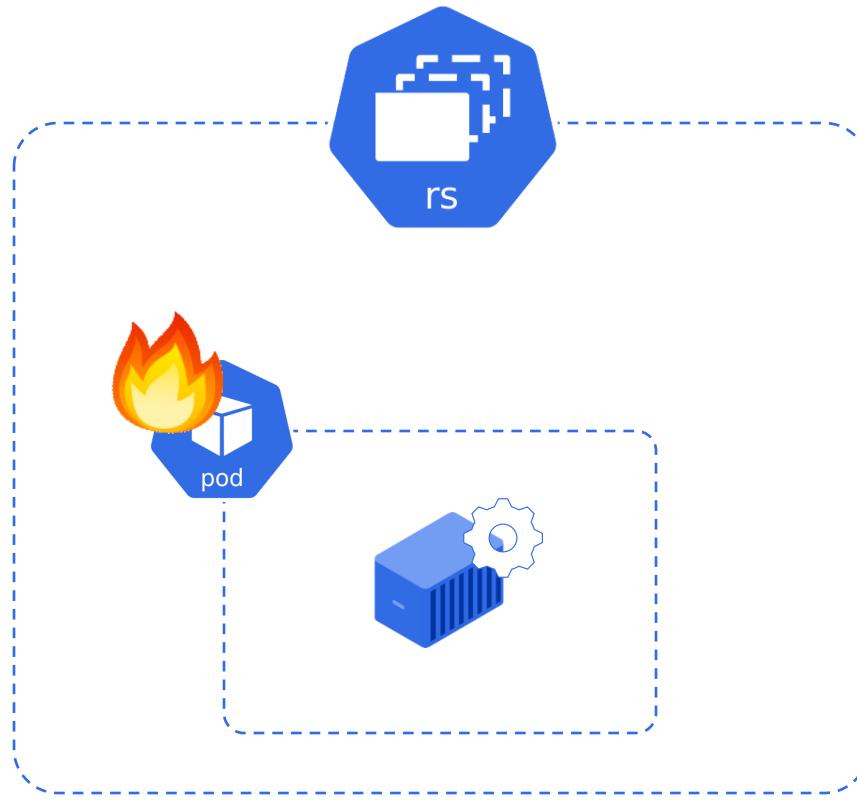


Demo | Pods



ReplicaSets

ReplicaSets

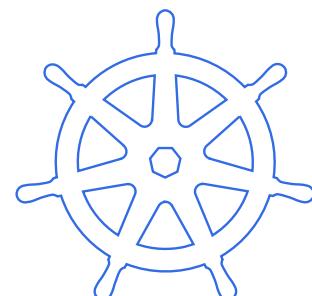


How ReplicaSets Works?

Define
ReplicaSet
Object

Monitor
running Pods

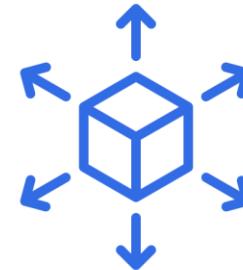
Effortless
Scaling



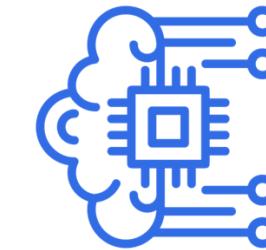
Benefits of ReplicaSets



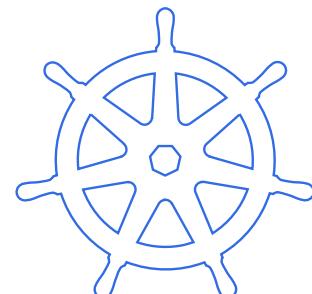
High Availability



Scalability

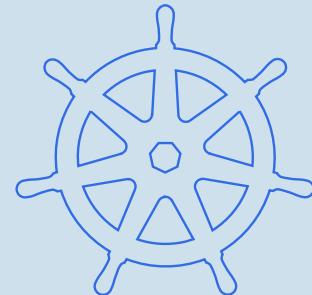


Self-healing





Demo | ReplicaSets



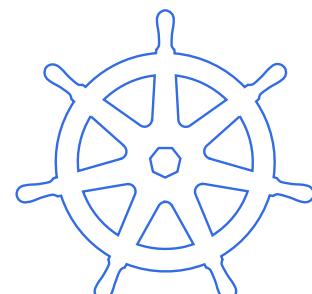
Deployments

Deployments

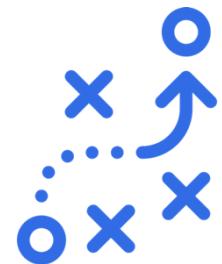
Offers advanced features such as rolling updates & rollbacks, makes it easier to manage the application lifecycle



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



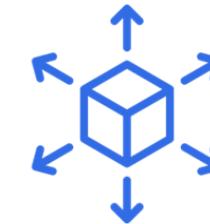
Benefits of Deployments



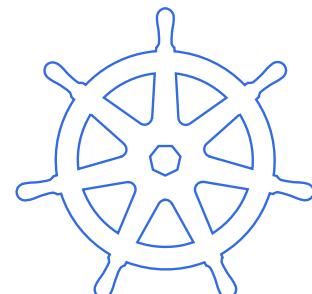
Update Strategies



Rollbacks



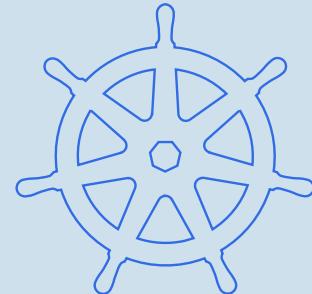
Scaling





Demo

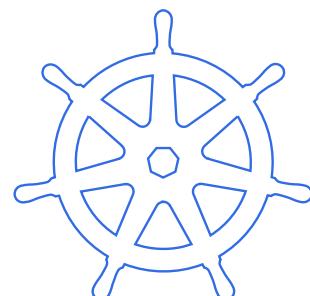
Deployments



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



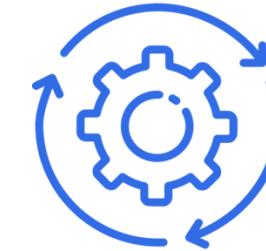
Benefits of Labels & Selectors



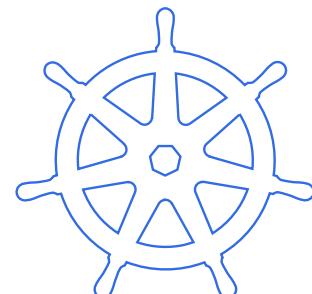
Categorization
/Organization



Selection



Automation



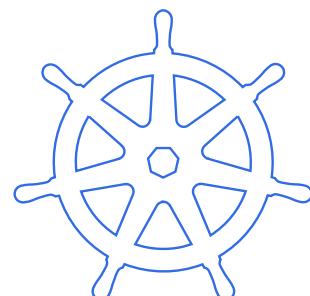
Key Concepts

Key-value pairs for classifying
Kubernetes resources

Labels

Selectors

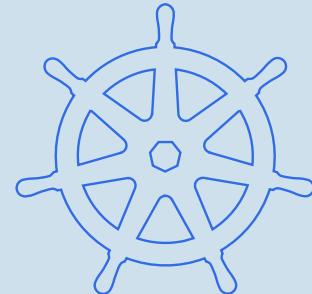
Filter resources based on labels
key-value comparisons





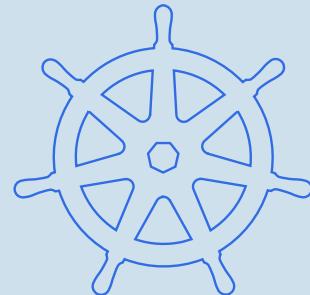
Demo

Labels &
Selectors





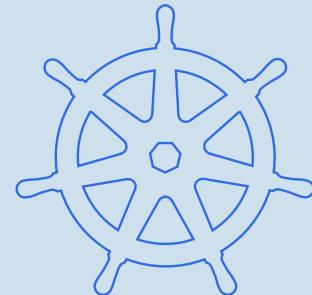
Demo | Inbuilt Labels





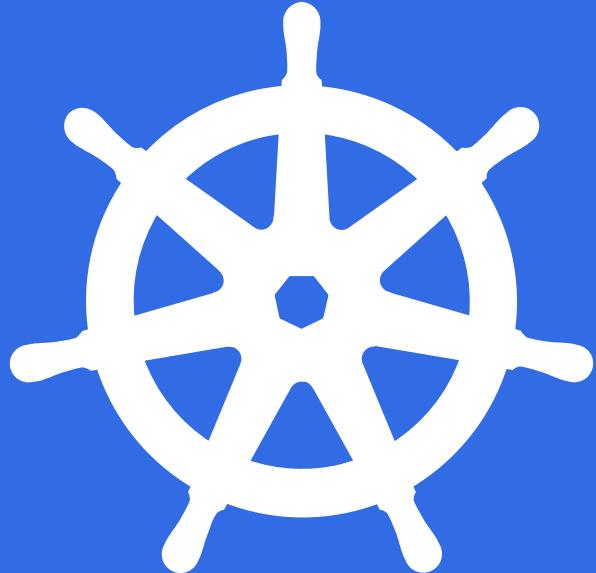
Demo

Useful kubectl
Tips

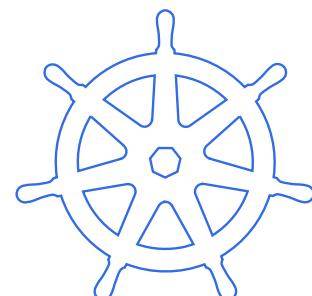


Summary

Summary



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

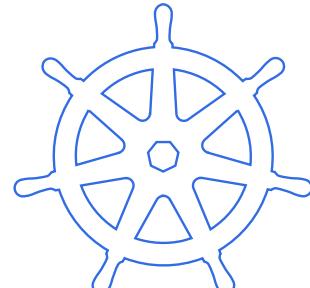


Section: 3

Kubernetes Networking

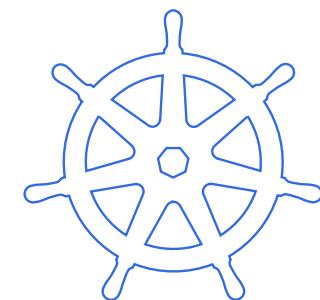
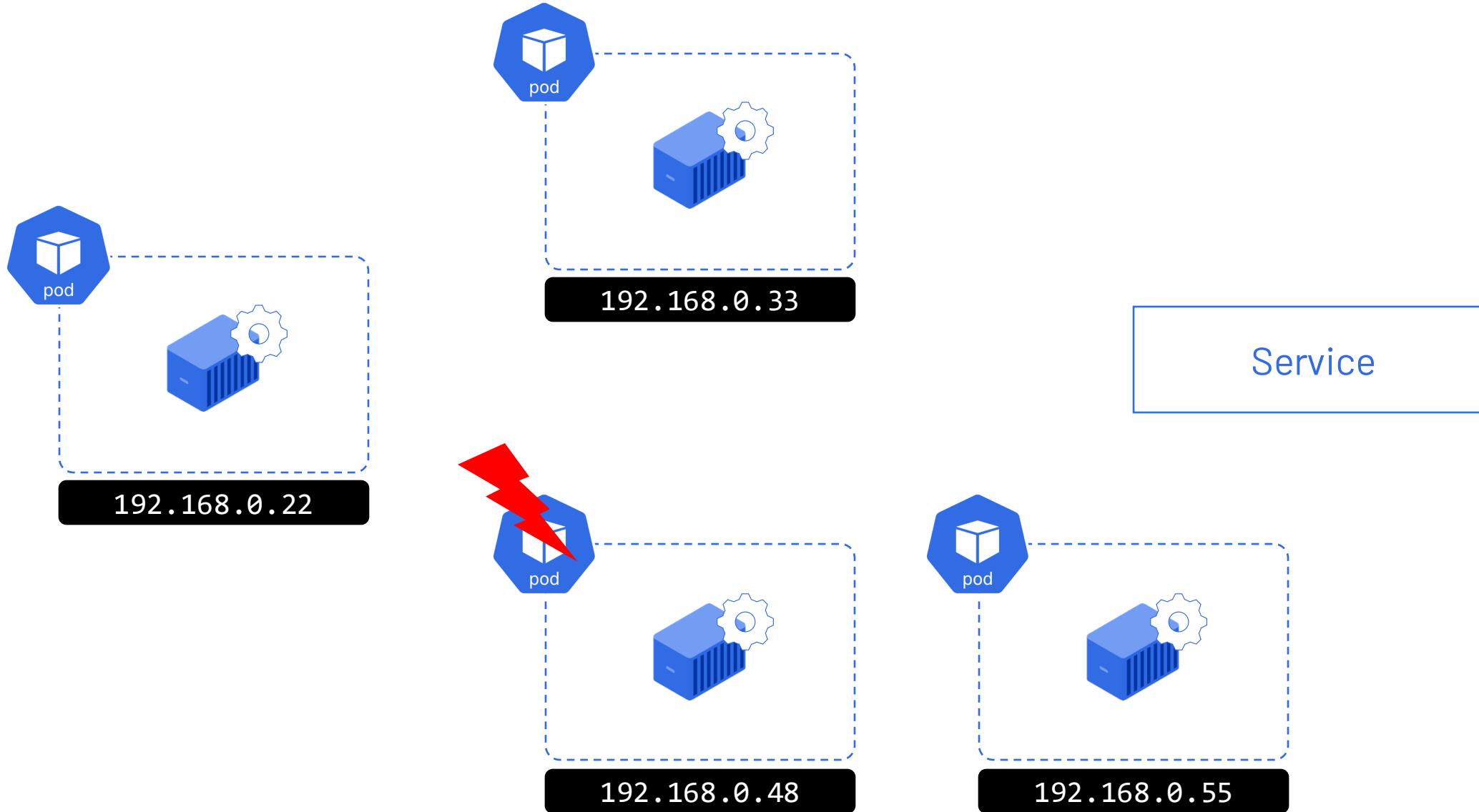
Section Overview

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

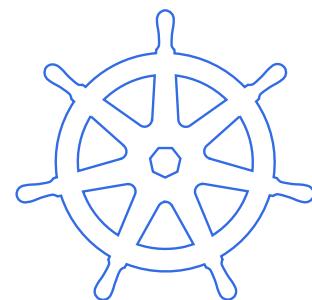
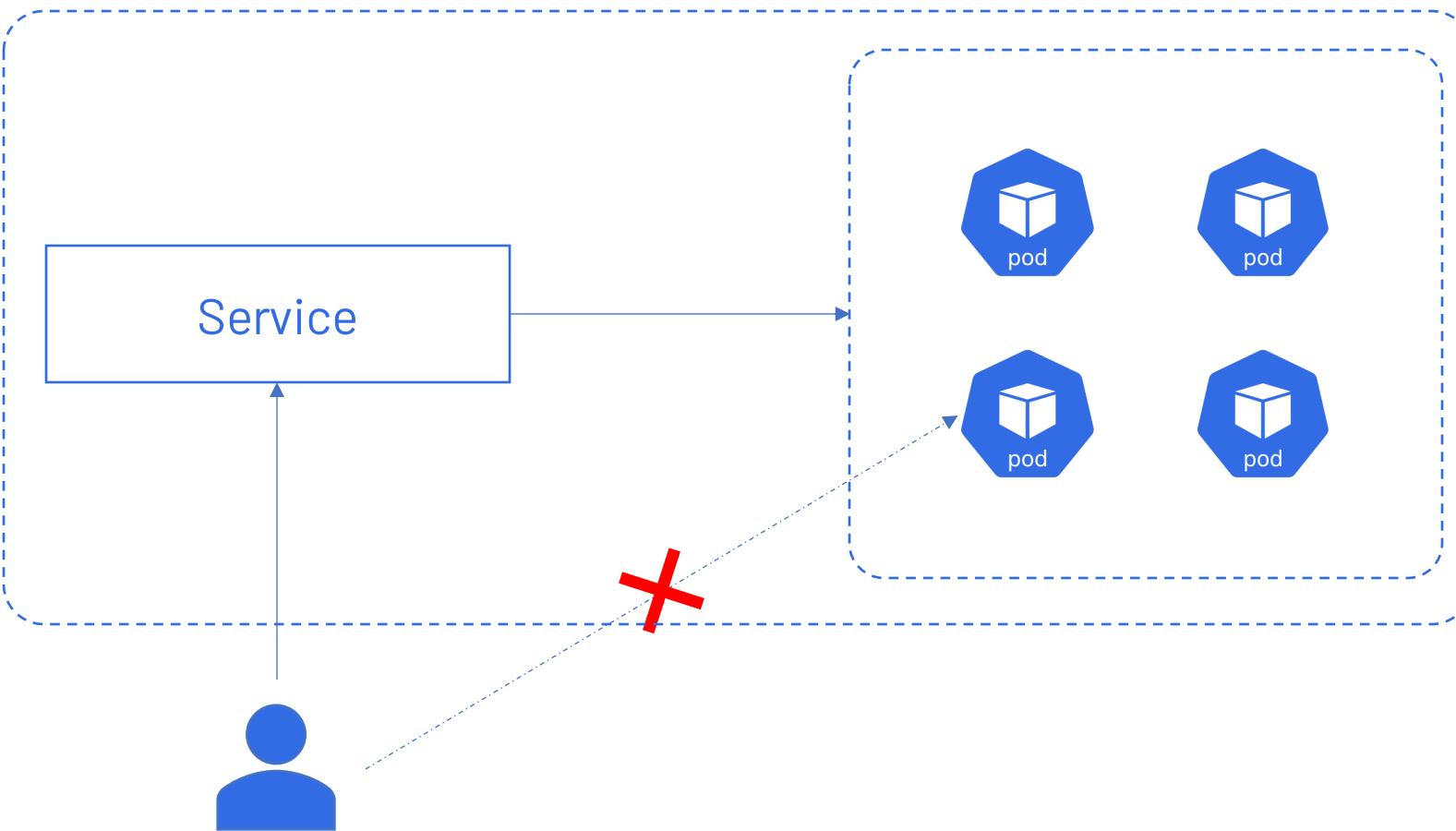


Services

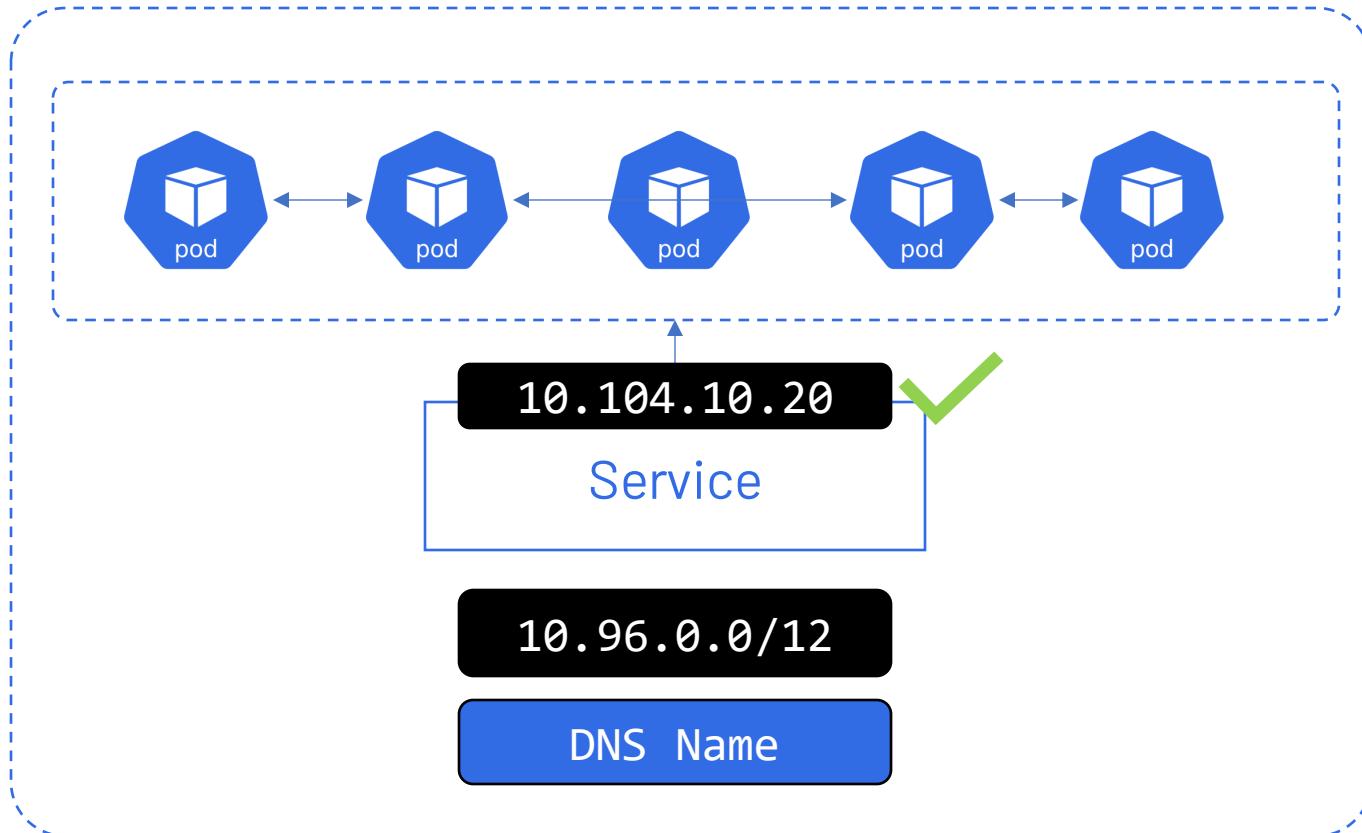
Services



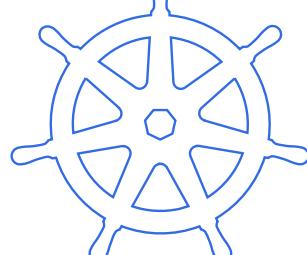
Services



Services

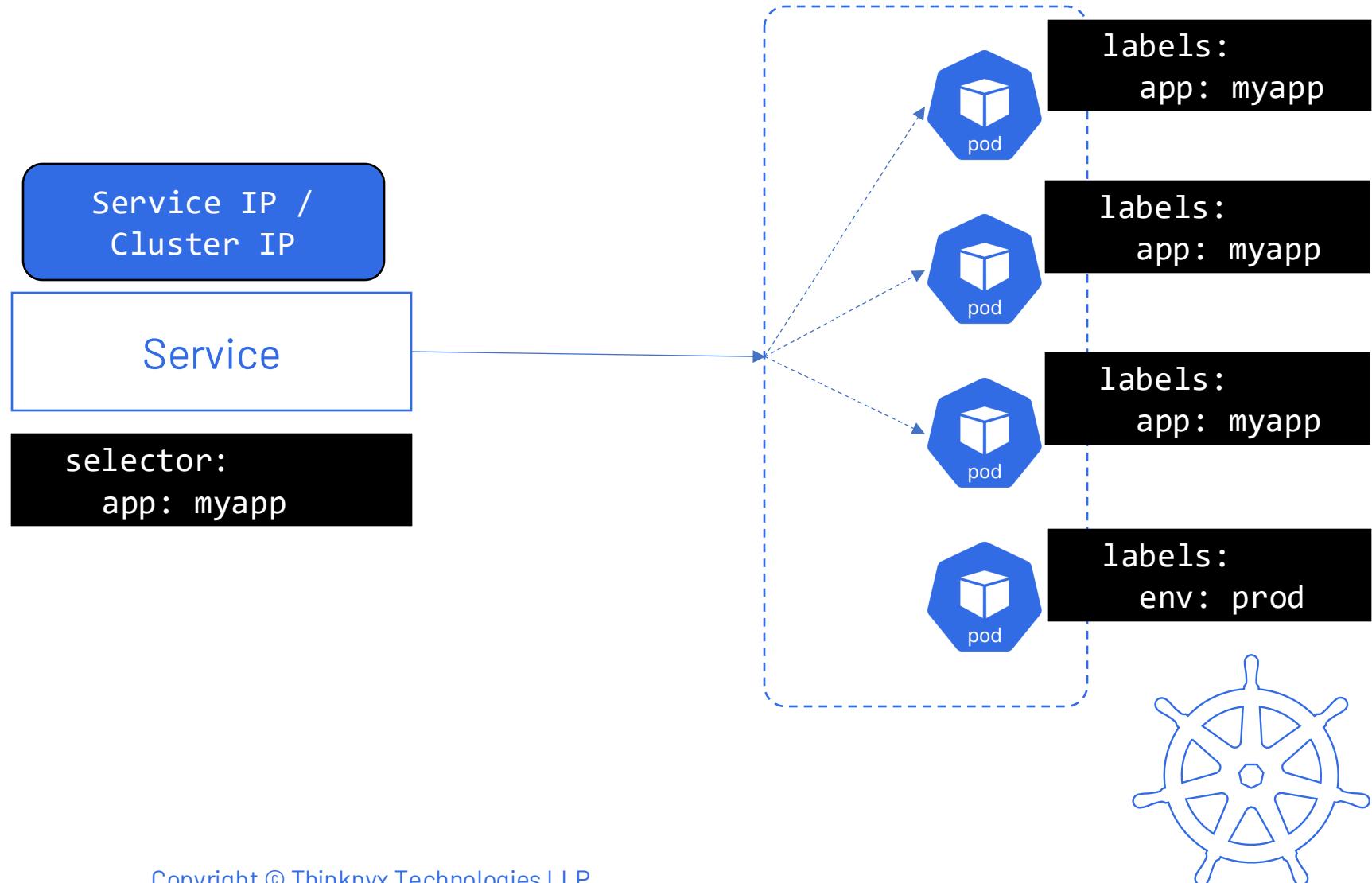


- ✓ ClusterIP
- ✓ NodePort
- ✓ LoadBalancer



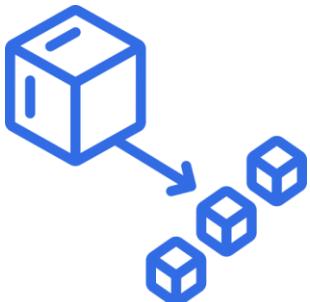
Services

Services can
perform Internal
load balancing

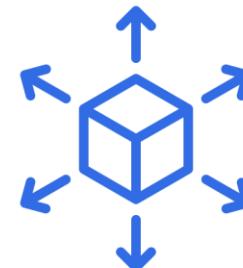


Benefits of Service Objects

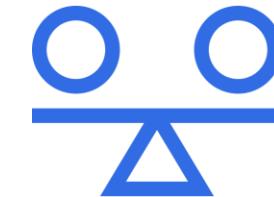
Benefits of Service Objects



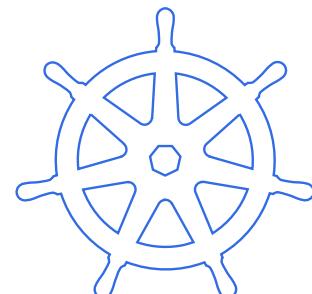
Decoupling



Scalability



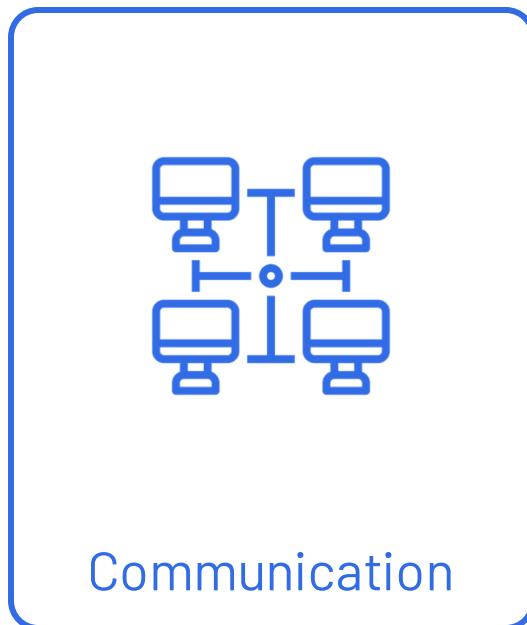
Stability



ClusterIP

ClusterIP

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



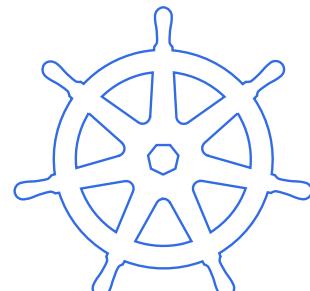
Communication



Security

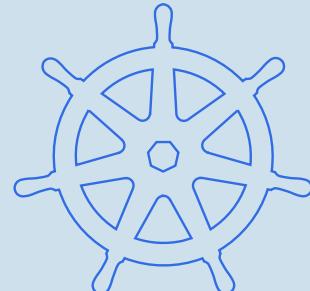


Limitations





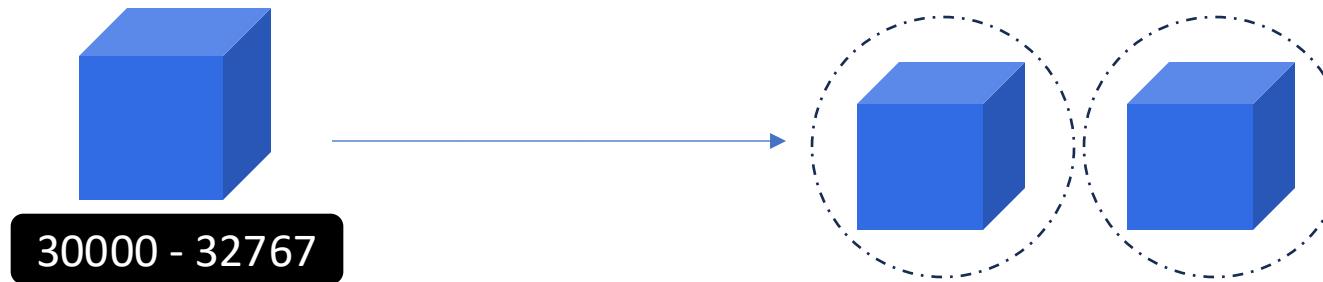
Demo | ClusterIP
Service Type



NodePort

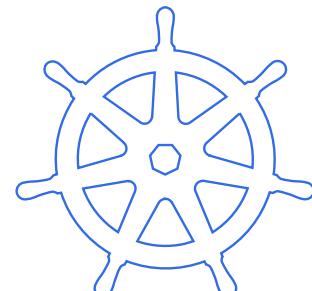
NodePort

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



NodePort Benefits

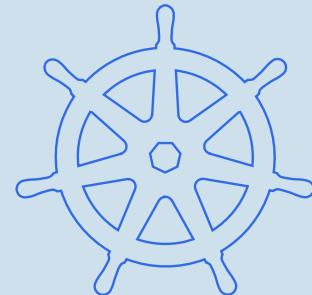
- ✓ Simple External Access
- ✓ No Load Balancer Required





Demo

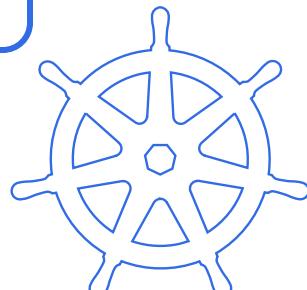
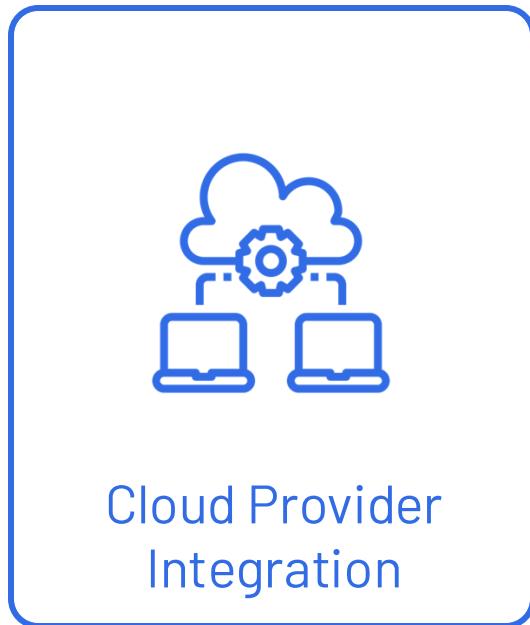
NodePort
Service Type



LoadBalancer

LoadBalancer

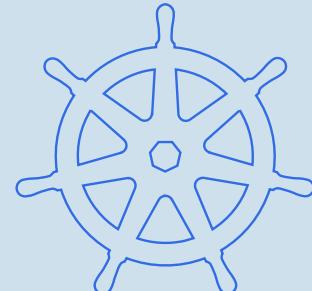
- Leverages external or cloud provider-specific load balancers to distribute incoming traffic across pods
- Highly available & scalable solution for exposing application to the public internet





Demo

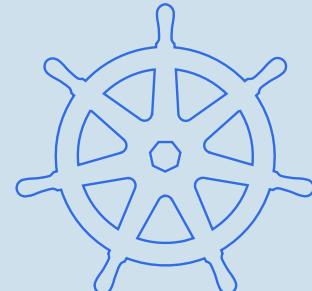
LoadBalancer
Service Type





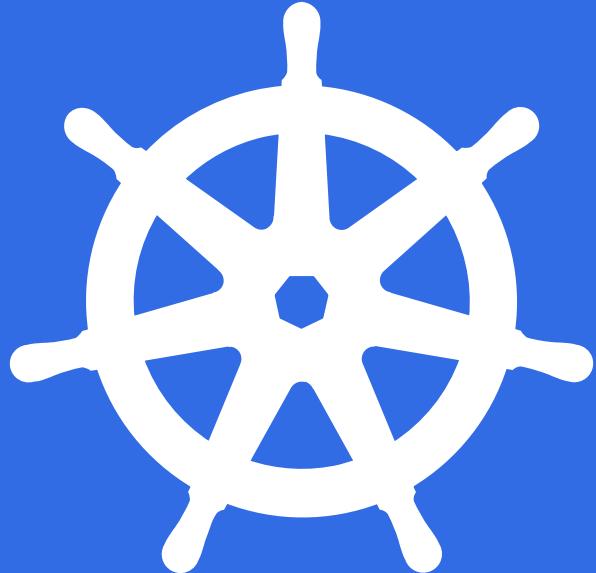
Demo

Troubleshooting
Deployment and Service

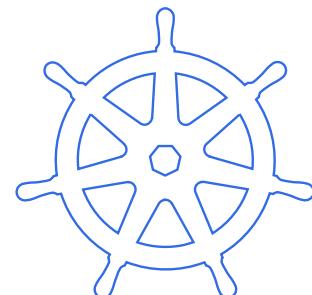


Summary

Summary



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

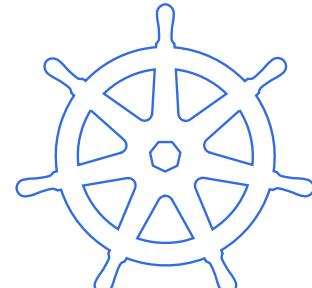


Section: 4

Kubernetes Manifest Files

Section Overview

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

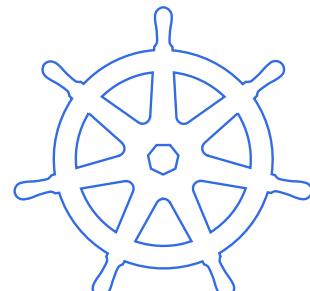
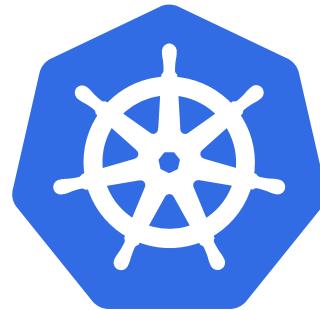


Understanding Kubernetes API Ecosystem

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

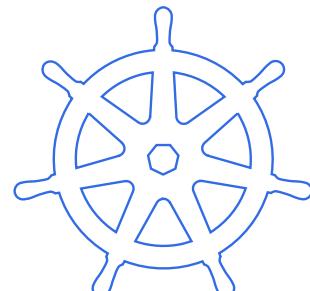


Section: 5

Section Overview

Useful Kubernetes Objects

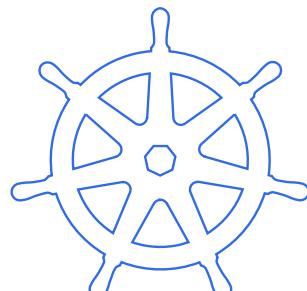
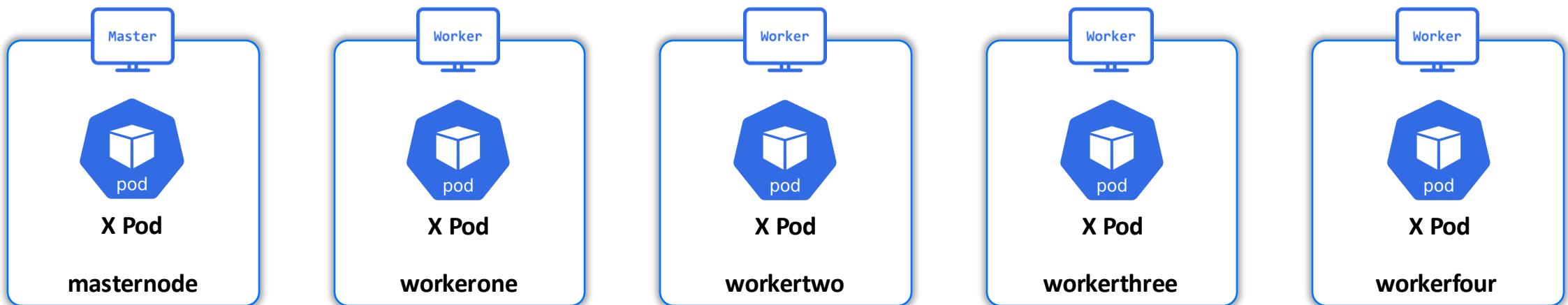
- ↳ **DaemonSet**
- ↳ **Jobs & CronJob**
- ↳ **Static Pods**
- ↳ **ConfigMaps & Secrets**



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

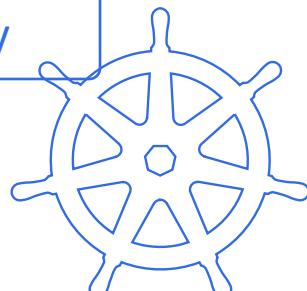
Uniform Deployment

Automatic Updates

Node Affinity>Selectors

Scalability

Resource Efficiency



DaemonSet



Logging



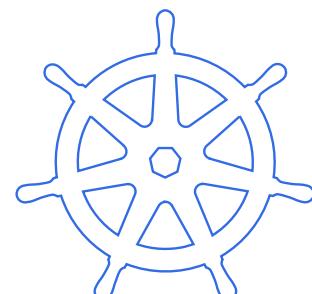
Monitoring



Networking

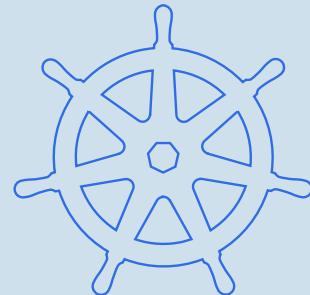


Security





Demo | DaemonSet

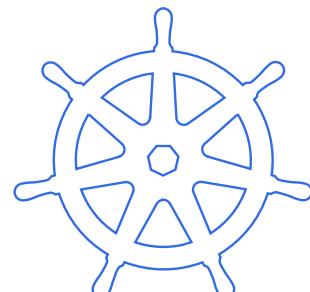
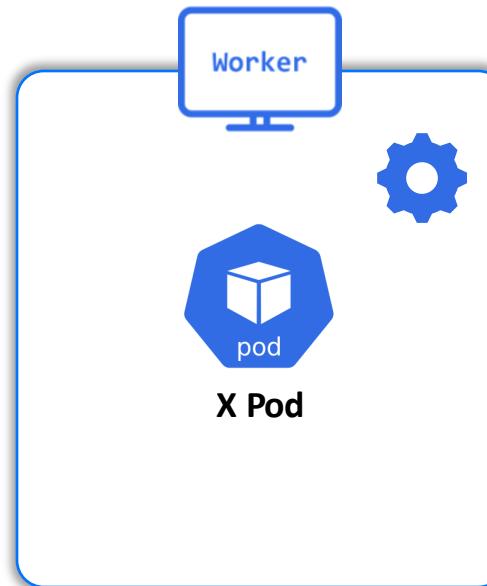


Jobs

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

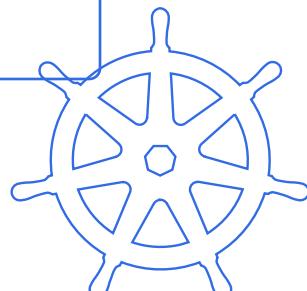
Key Features

Finite
Execution

Pod
Management

Parallelism

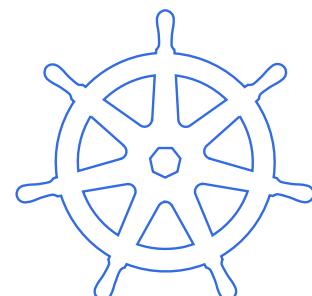
Completion
Tracking



Jobs

Types of Jobs

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



Jobs

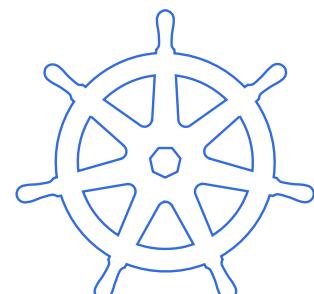
Use Cases

Data Processing

Database
Migrations

Scheduled
Maintenance

Event-Driven
Tasks



Jobs

Resource
Management

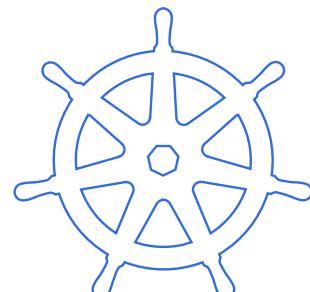
Job Cleanup

Best
Practices

Retries &
Backoff Limit

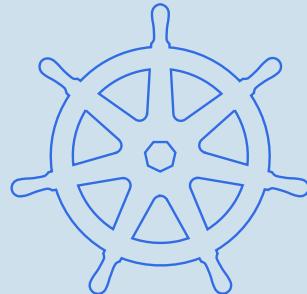
Idempotency

Monitoring and
Logging





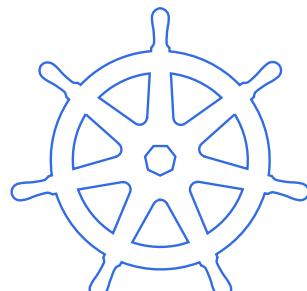
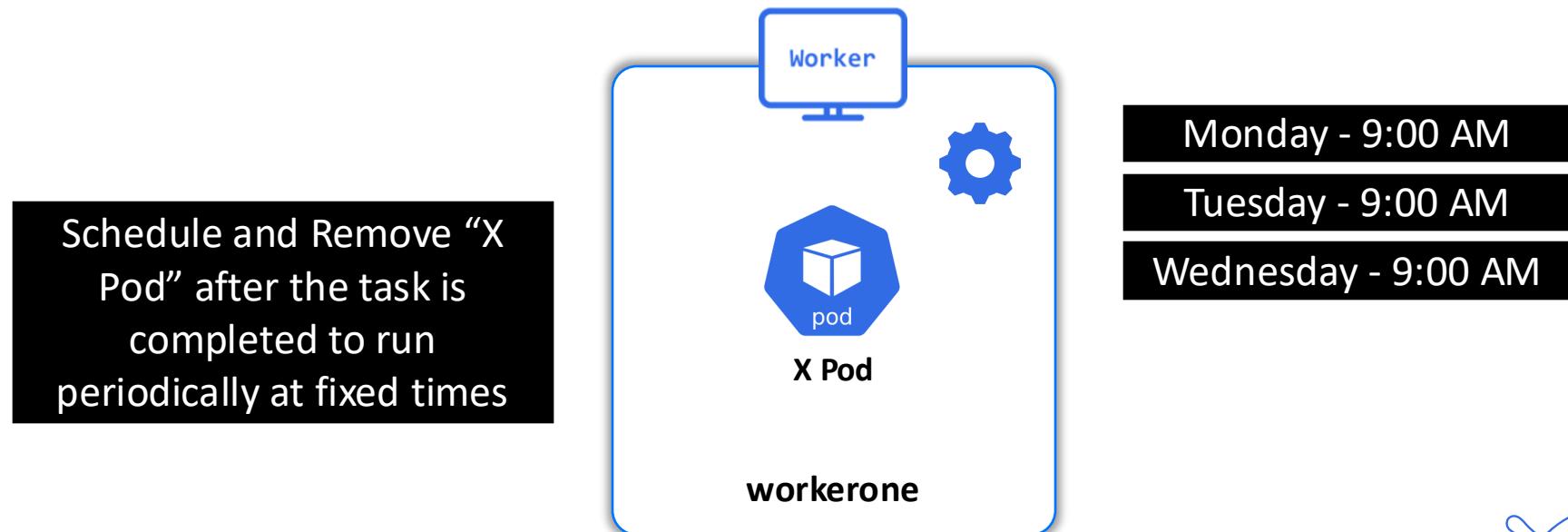
Demo | Jobs



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

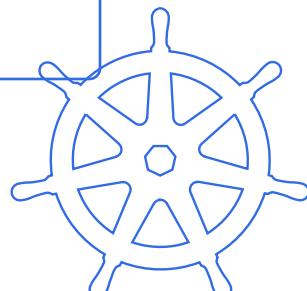
Key Features

Scheduled
Execution

Job
Management

Time-Based
Triggers

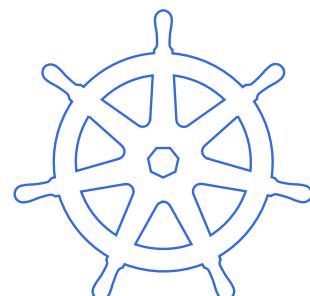
Concurrency
Policy



CronJob

Types of CronJob

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



CronJob

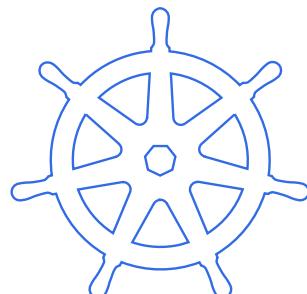
Use Cases

Regular Backups

System
Maintenance

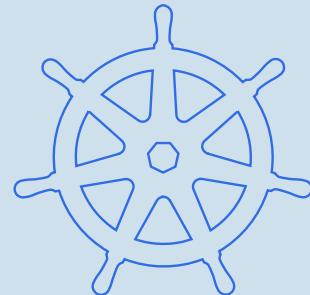
Report
Generation

Periodic Data
Processing





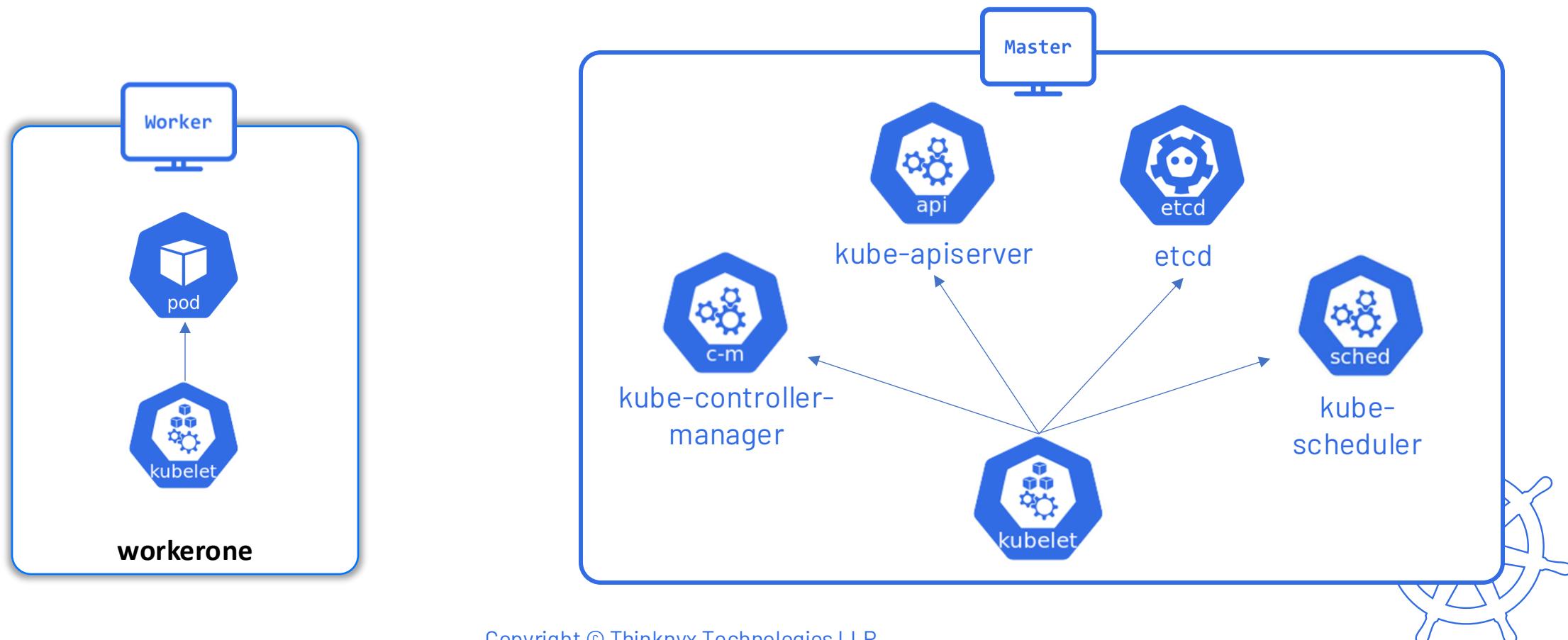
Demo | CronJob



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

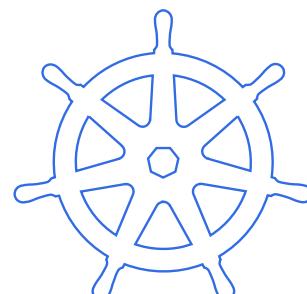
- Static Pods are managed directly by the kubelet on individual nodes
- Provide a way to run containers on nodes without relying on the Kubernetes control plane for management

Key Features

Node-Specific
Management

Configuration
Through Files

Self-Healing

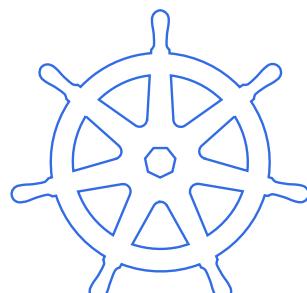


Static Pods

Use Case

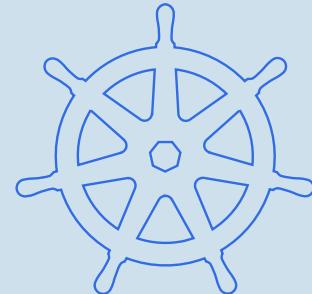
Critical Cluster Components

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





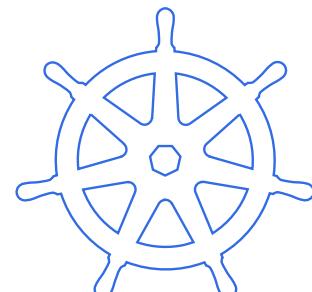
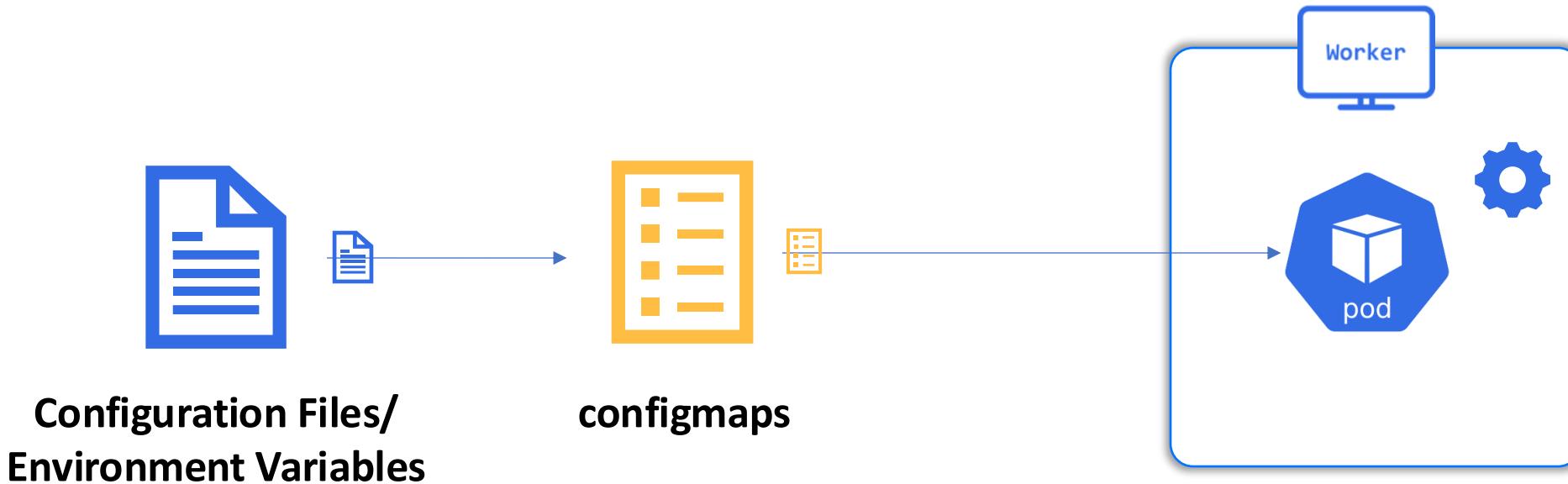
Demo | Static Pods



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

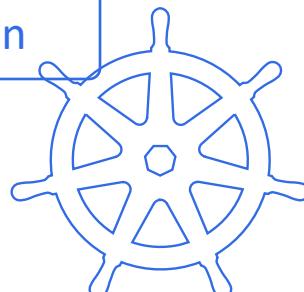
Key Features

Centralized
Management

Dynamic
Updates

Versatile Data
Sources

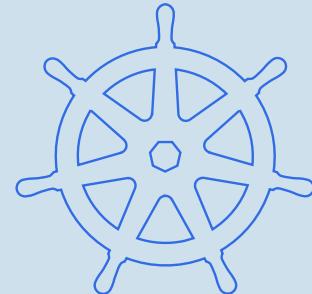
Seamless
Integration





Demo

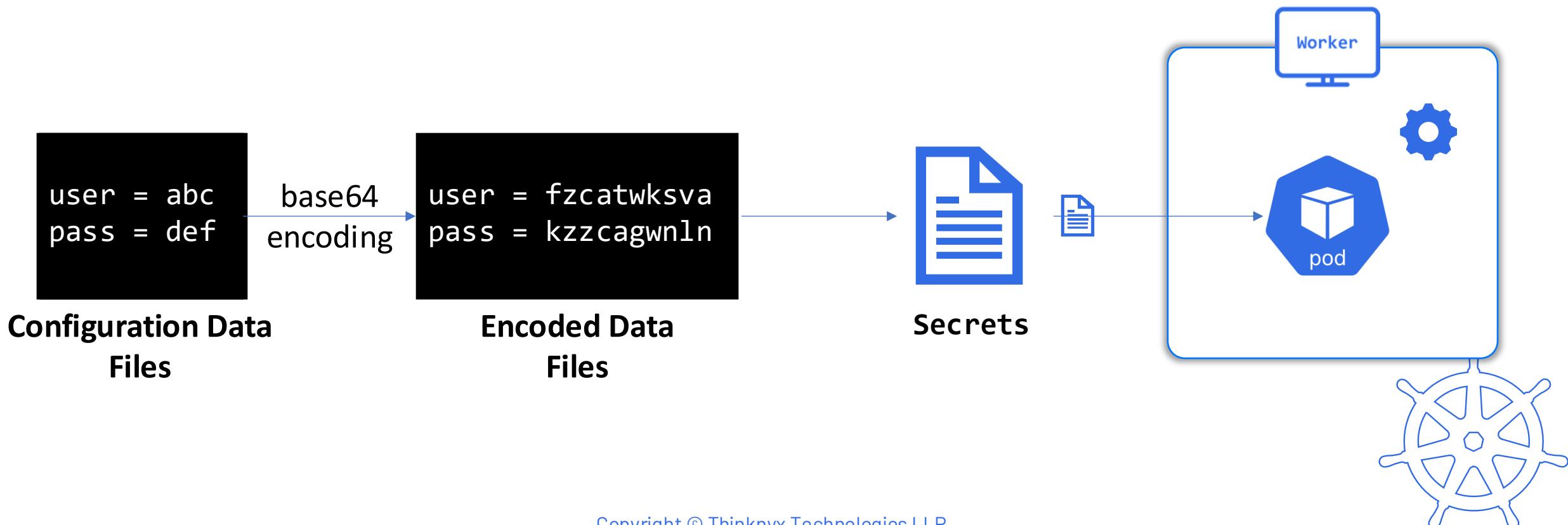
ConfigMaps



Secrets

Secrets

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

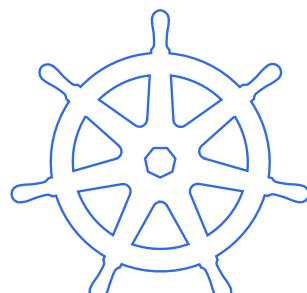


Secrets

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

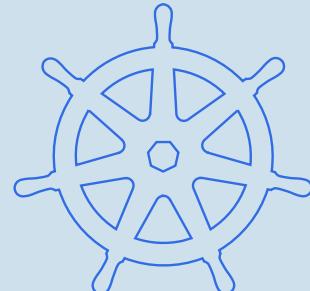
Types of Secrets

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





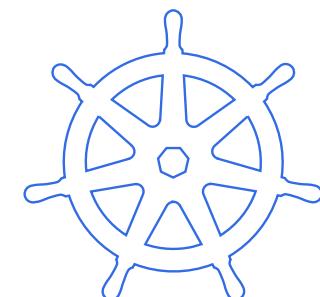
Demo | Secrets



Summary

Summary

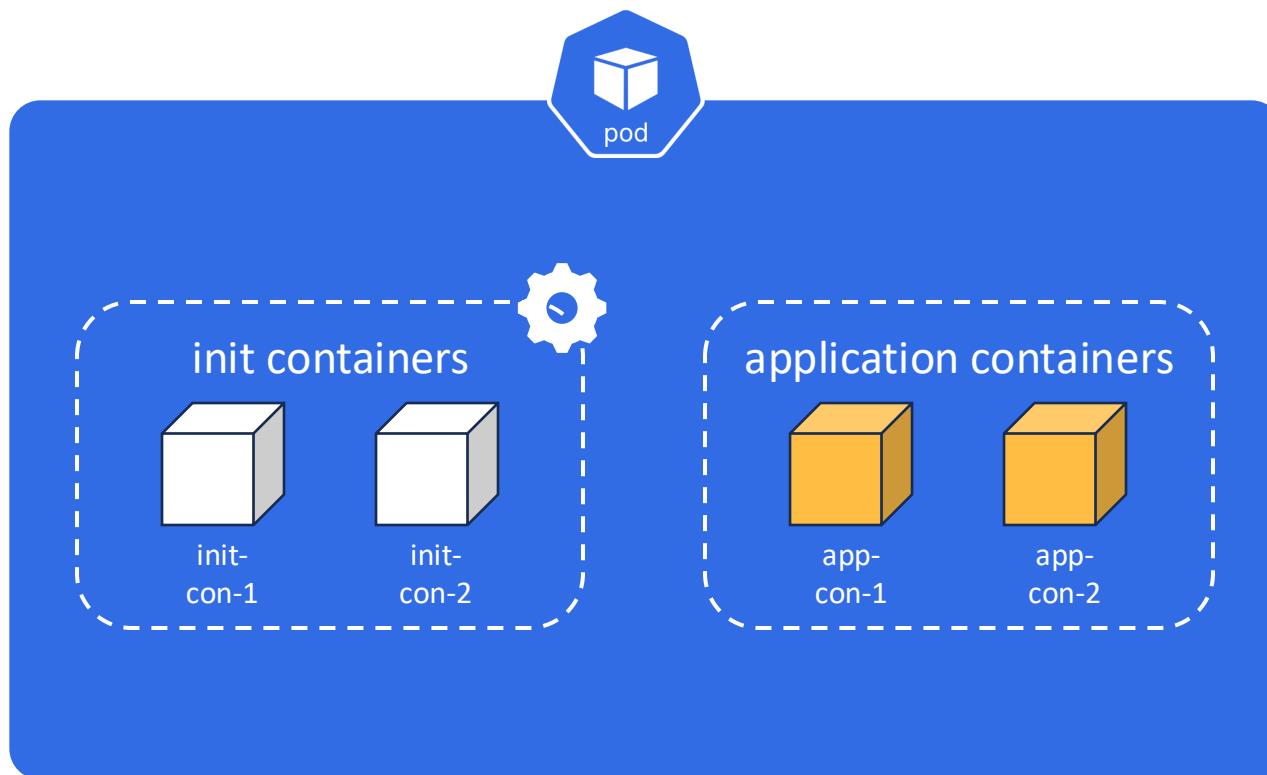
- ✓ DaemonSet
- ✓ Jobs & CronJob
- ✓ Static Pods
- ✓ ConfigMaps & Secrets



Init Containers

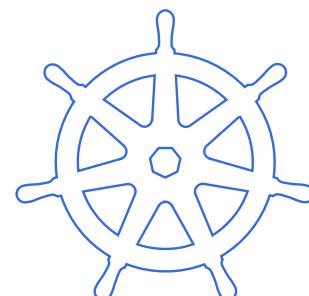
Init Containers

- 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

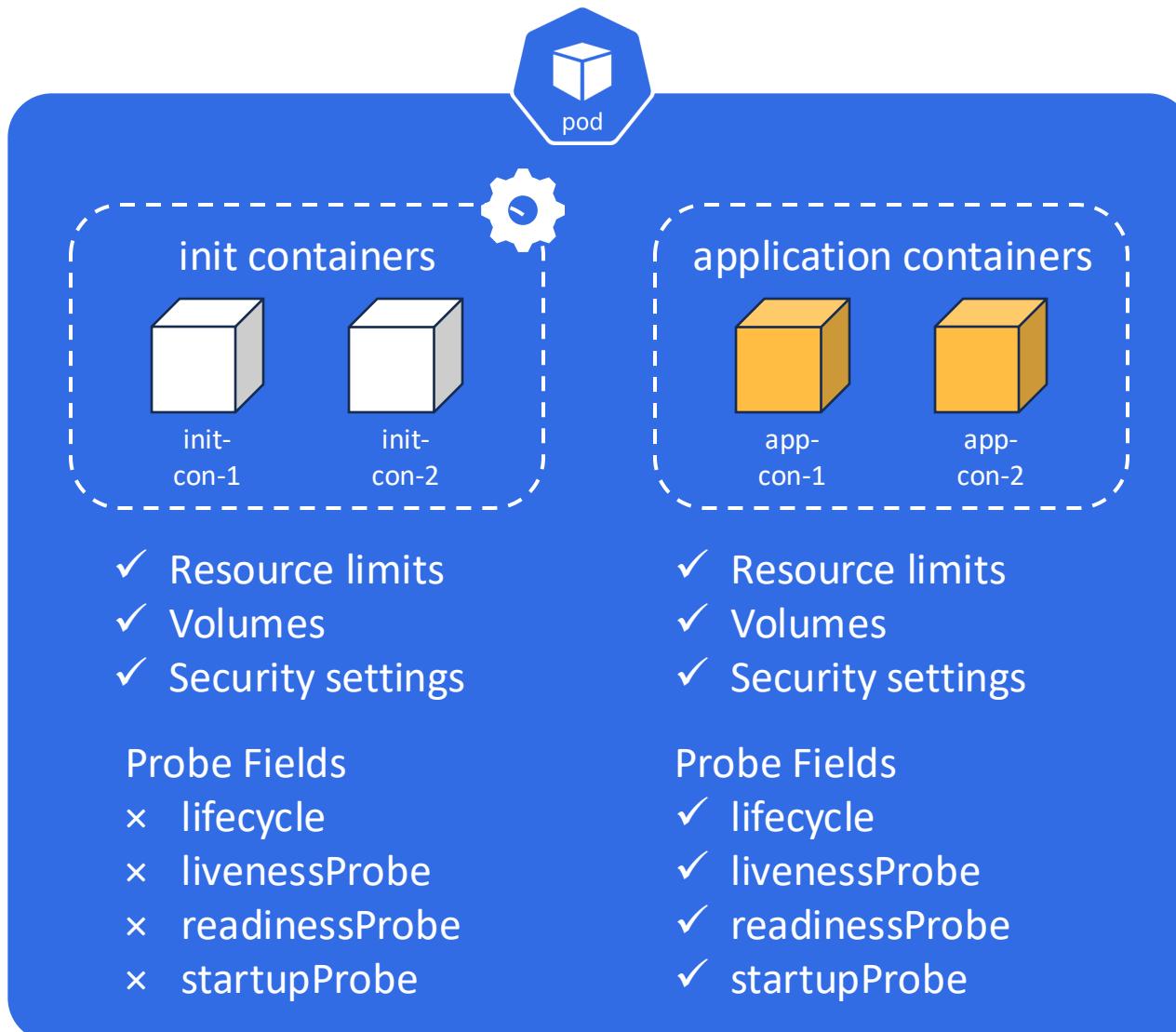


Key characteristics:

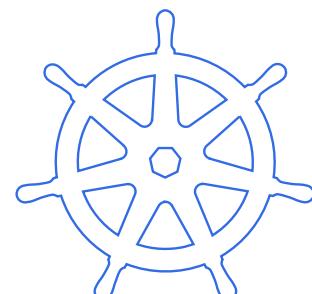
- ✓ Run to Completion
- ✓ Sequential Execution
- ✓ Failure Handling



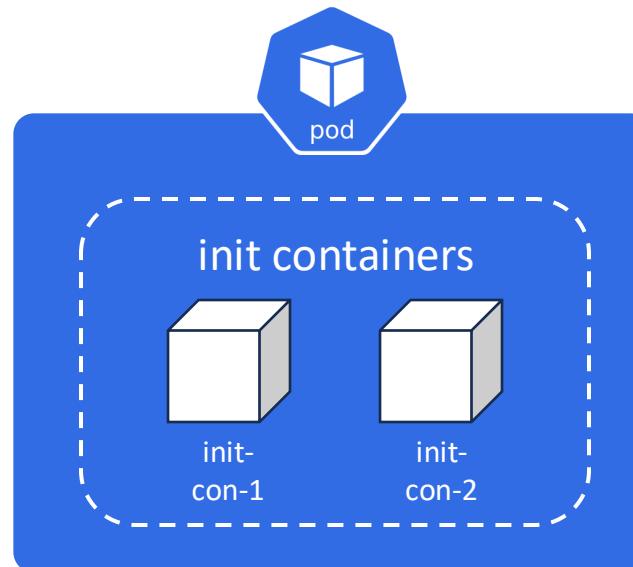
Init Containers



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

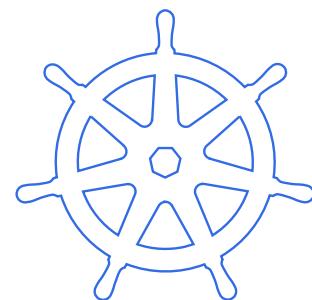


Init Containers



Separate
Images

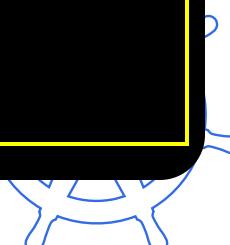
Startup
Control



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"]
```



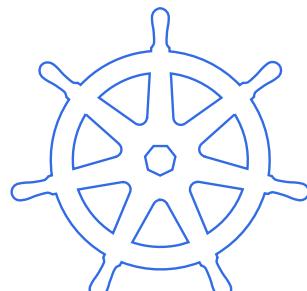
Resource Limits & Requests

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



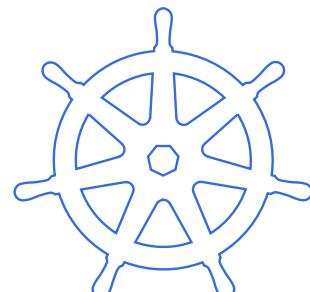
Resource Limits & Requests

Set the minimum amount of CPU or memory a container needs

Resource Requests

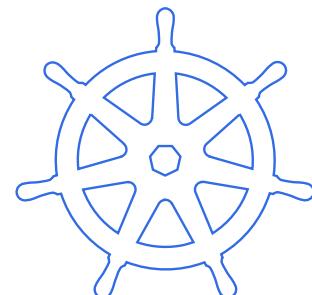
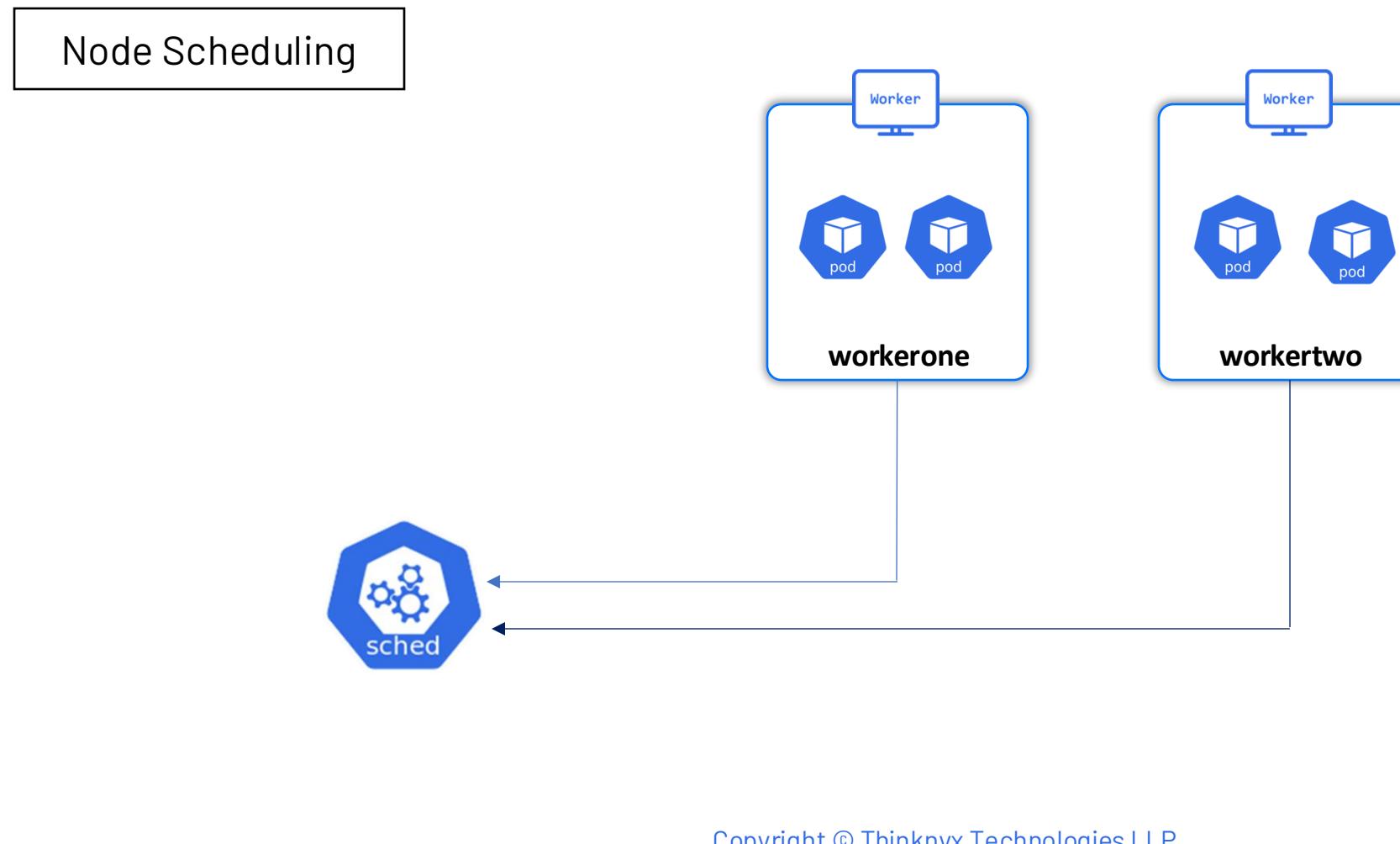
Resource Limits

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



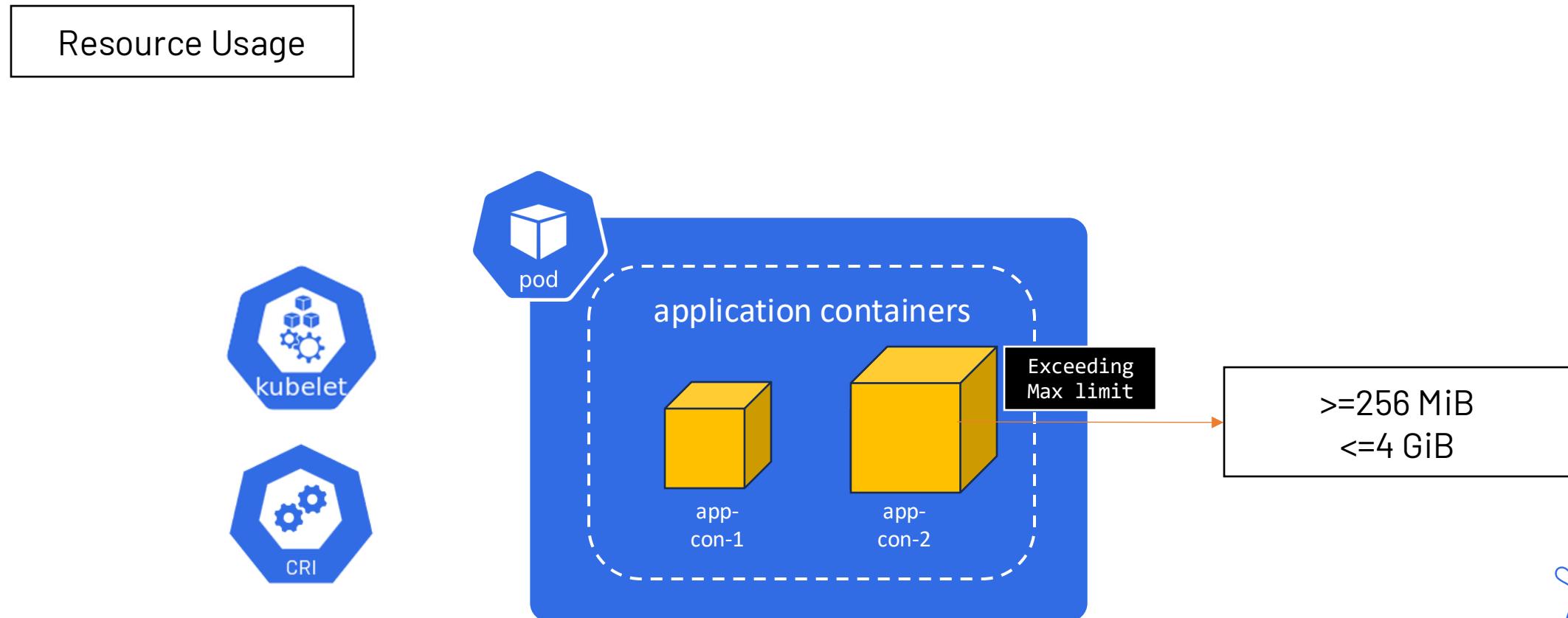
Resource Limits & Requests

How requests & limits work?



Resource Limits & Requests

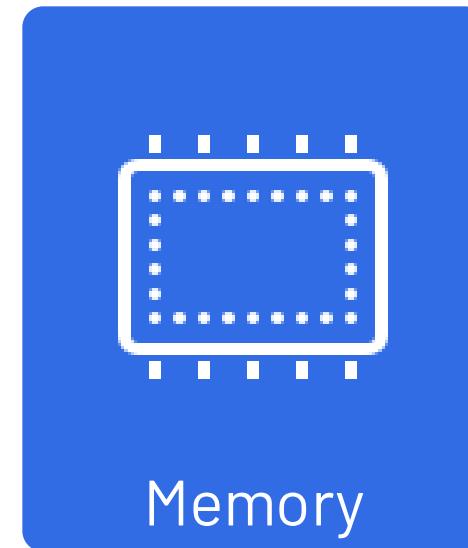
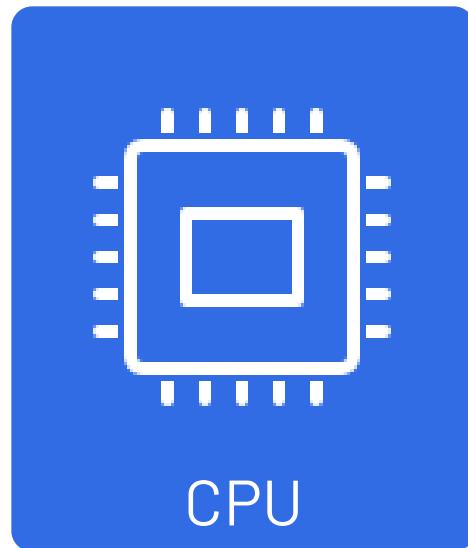
How requests & limits work?



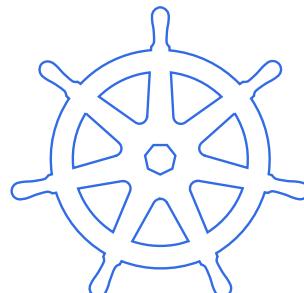
Resource Limits & Requests

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).





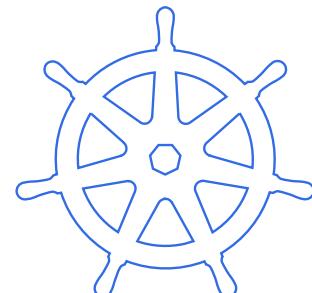
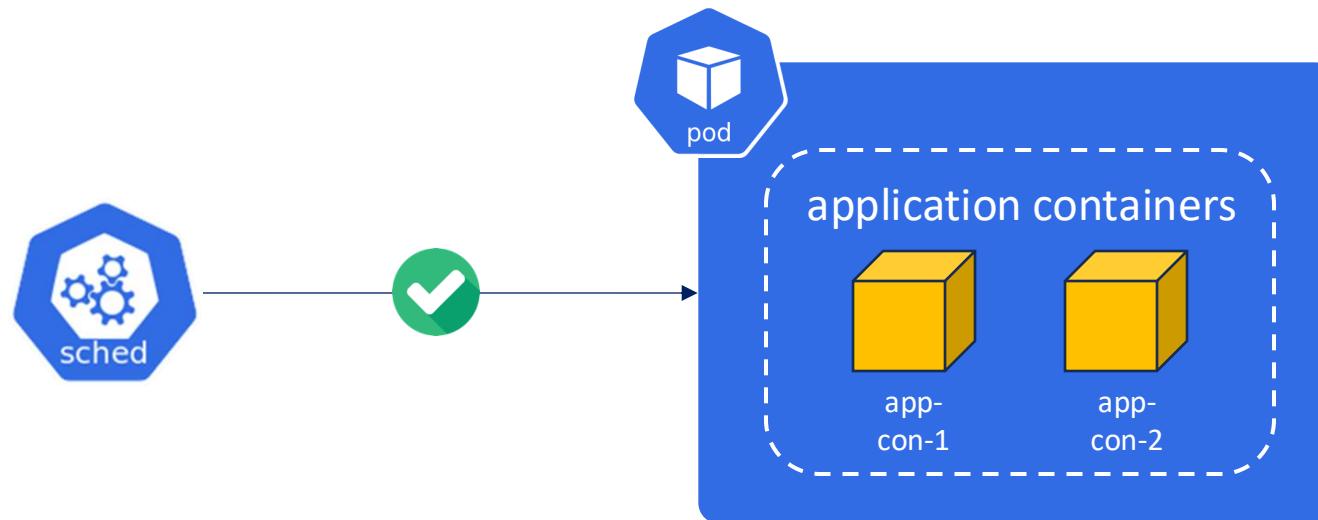
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
    - name: app
      image: httpd:v1
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
    - name: redis
      image: redis:v1
   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



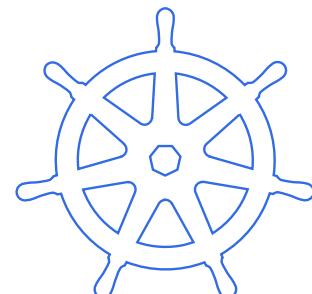
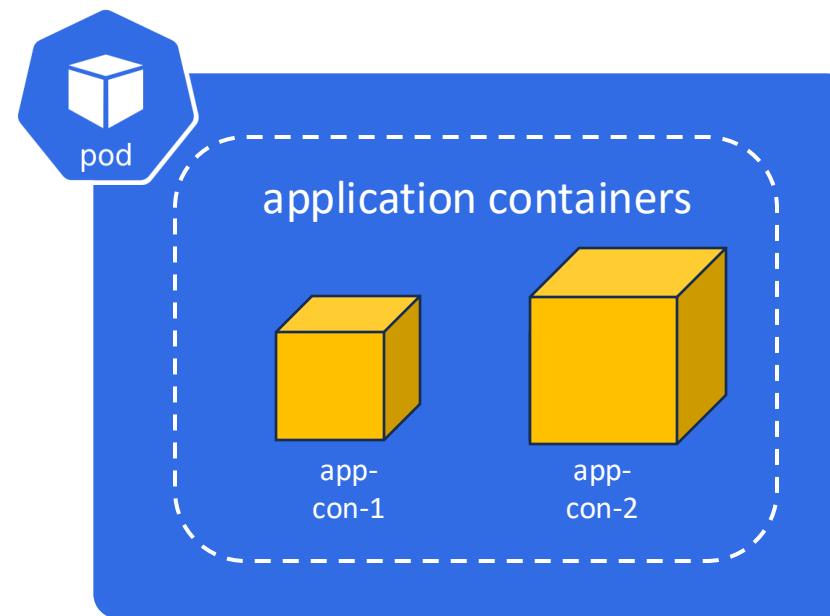
Resource Limits & Requests

How Kubernetes Applies Requests and Limits

During Execution



cgroups

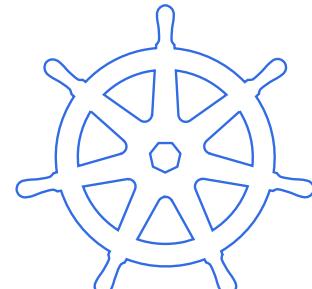


Section: 6

Scheduling

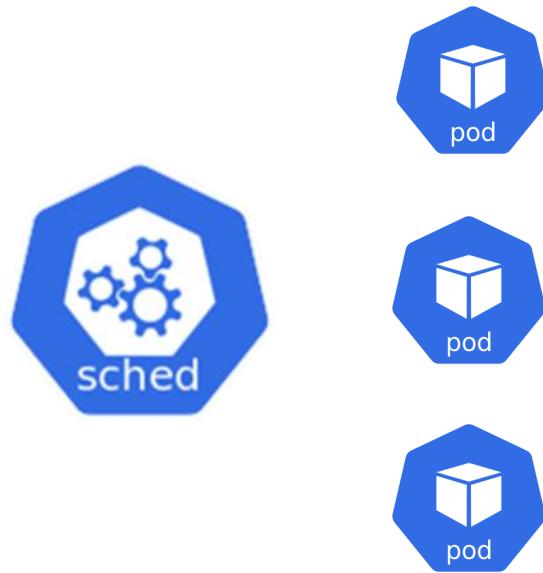
Section Overview

- ↳ **nodeName**
- ↳ **nodeSelector**

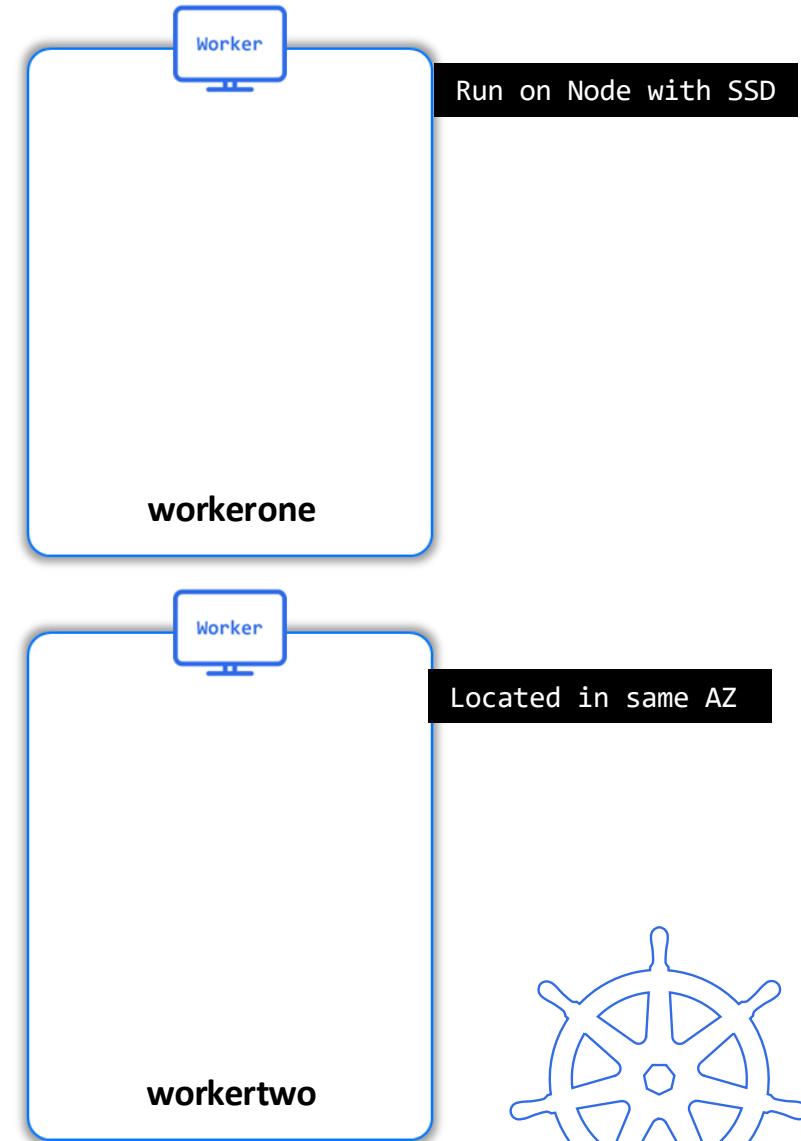


nodeName

nodeName



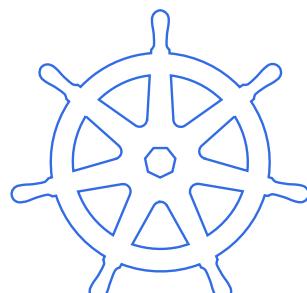
nodeName: workerone



nodeName

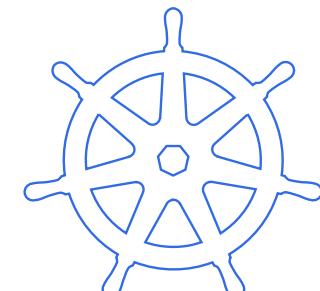
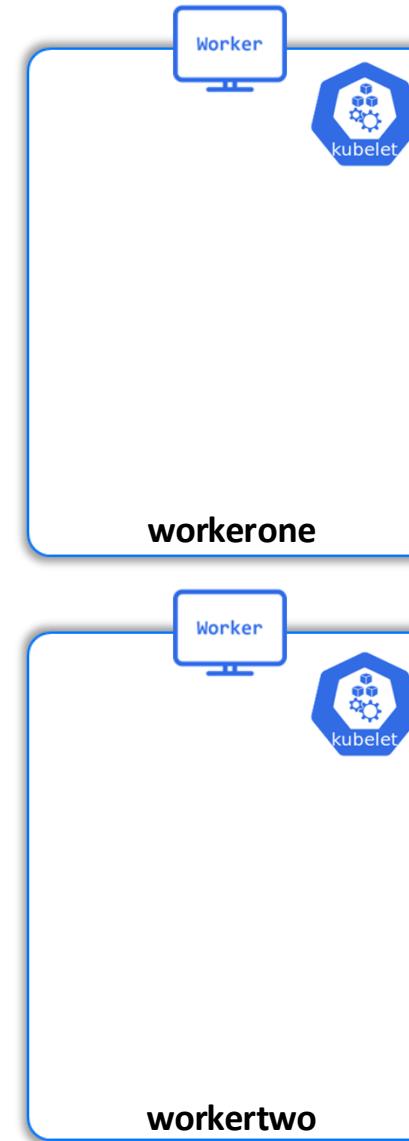
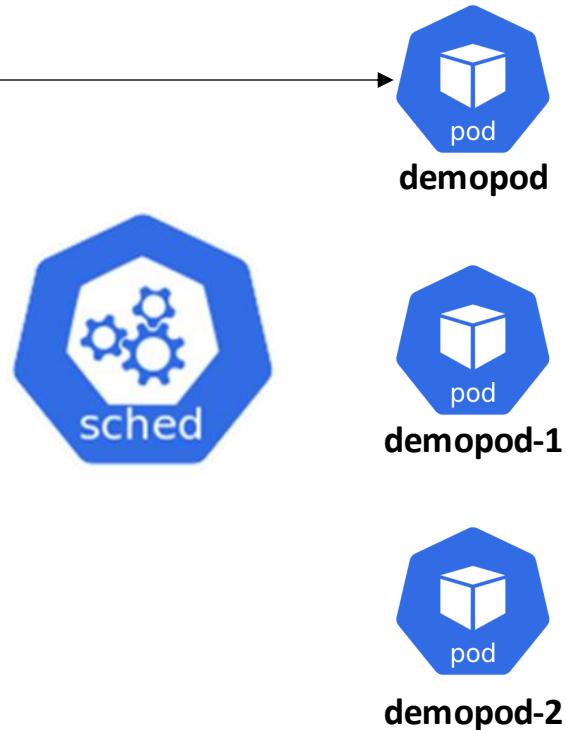
workerone

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



nodeName

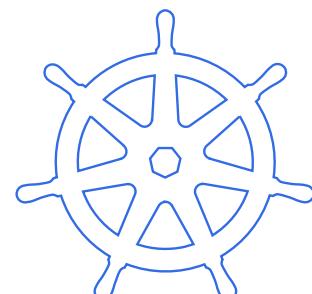
```
apiVersion: v1
kind: Pod
metadata:
  name: demopod
spec:
  containers:
    - name: demopodcon
      image: nginx:latest
  nodeName: workerone
```



nodeName

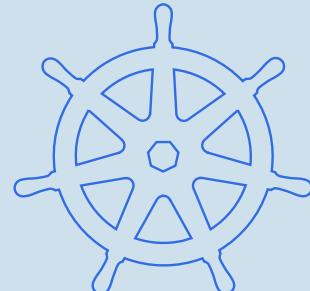
Limitations to consider

- 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

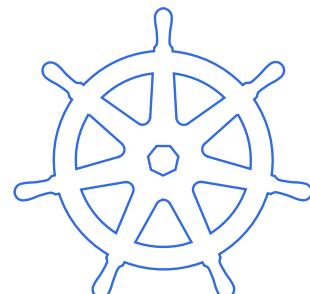


nodeSelector

nodeSelector

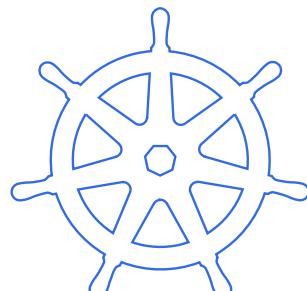
nodeSelector:

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?



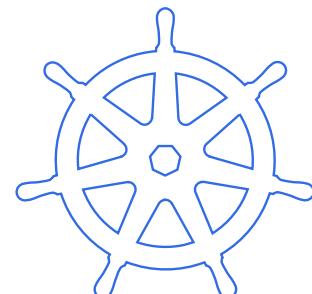
Resource
Management



Environment
Segregation



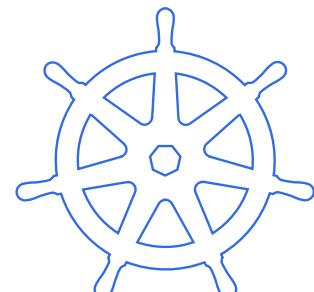
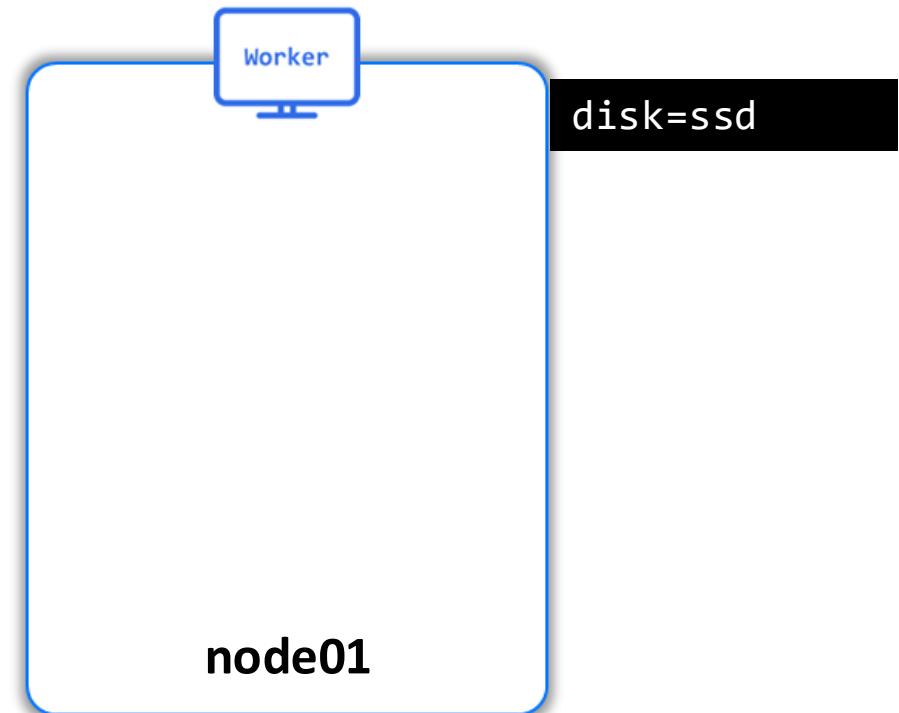
Workload
Isolation



How to Label Nodes?

kubectl

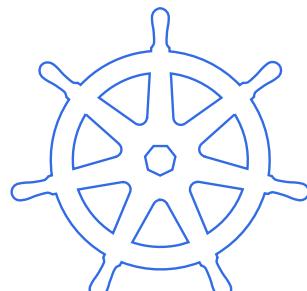
```
kubectl label nodes node01 disk=ssd
```



How to Label Nodes?

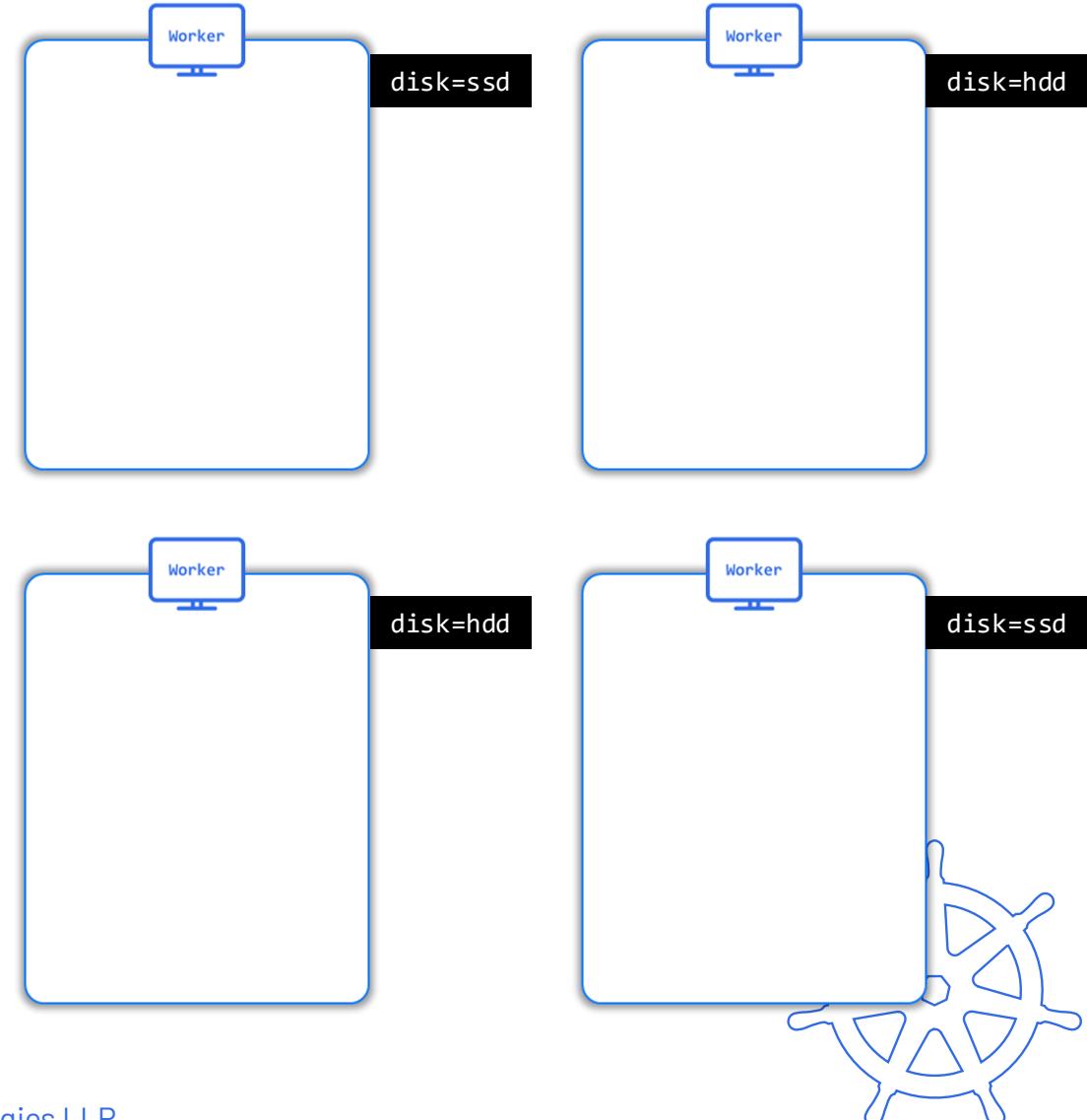
scheduler will look for nodes
labeled with **disk=ssd**

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



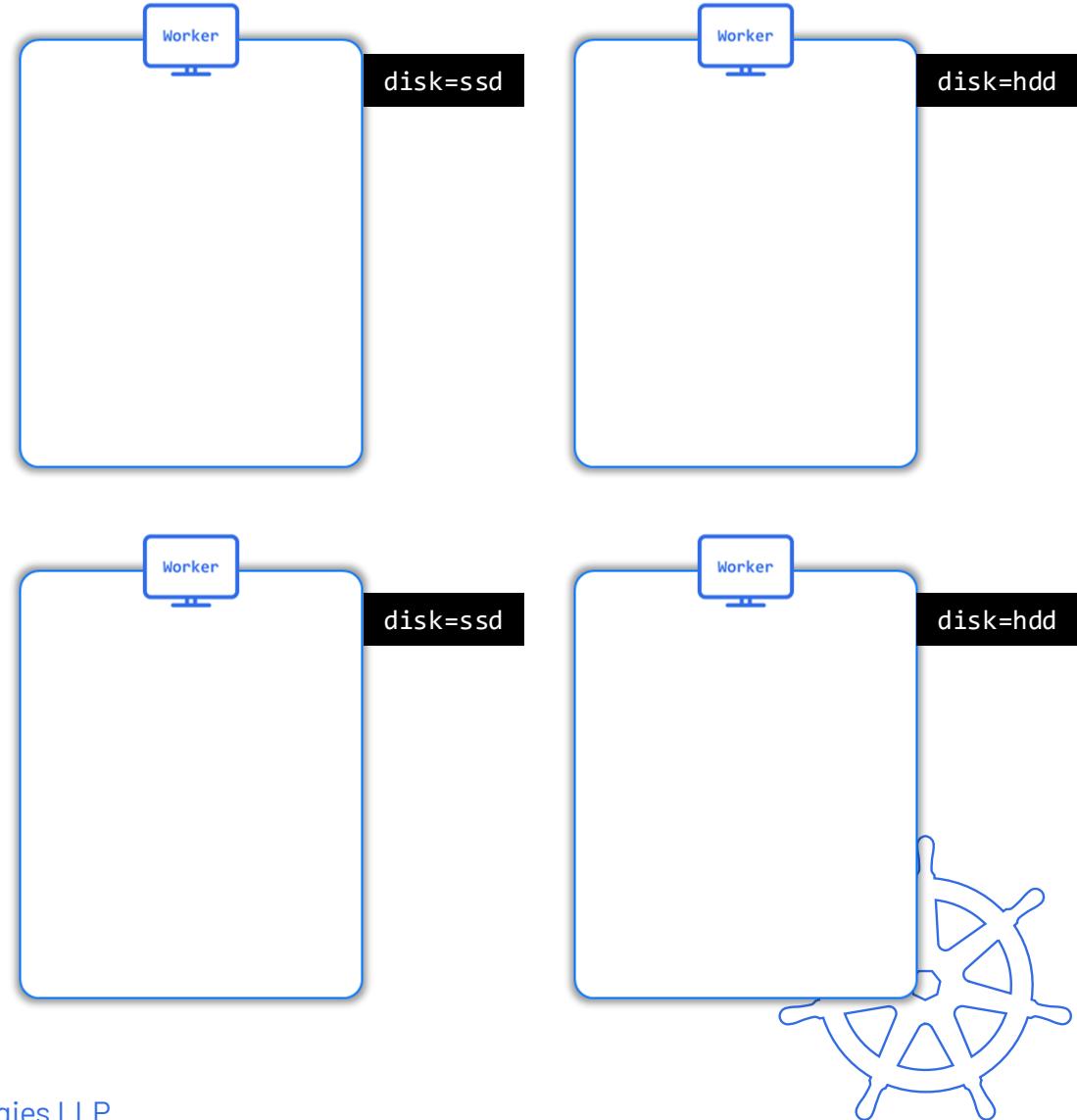
How nodeSelector Works?

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



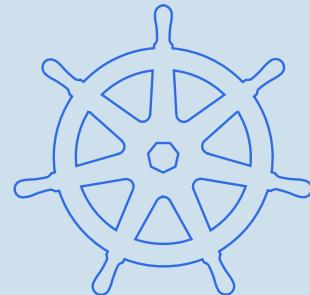
How nodeSelector Works?

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





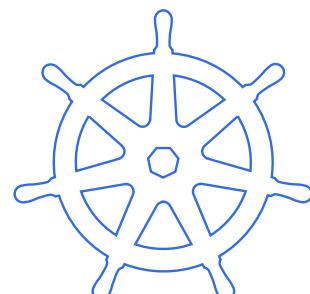
Demo | nodeSelector



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



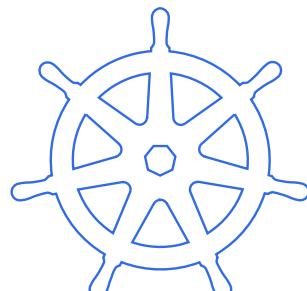
Taints & Tolerations

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

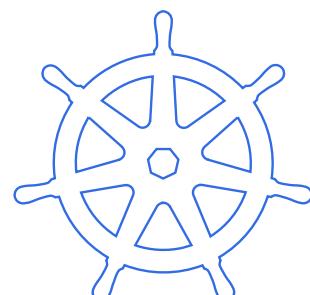
Adding a Taint

effect:

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

Removing a Taint

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



Applying Taints & Tolerations

Adding a Taint

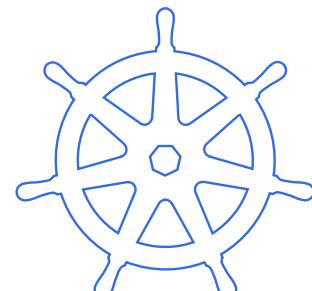
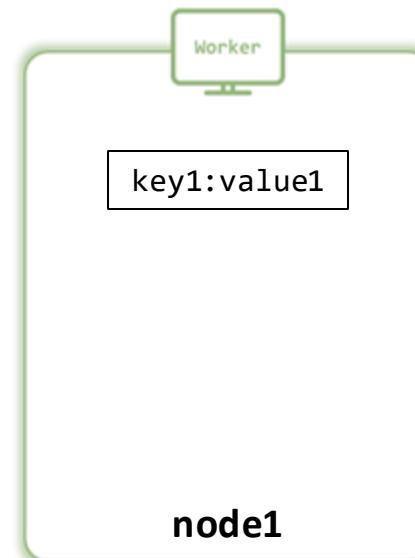
```
kubectl taint nodes node1 key1=value1:NoSchedule
```

Adding Tolerations to Pods

PodSpec:

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

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



Taint Effects

NoSchedule

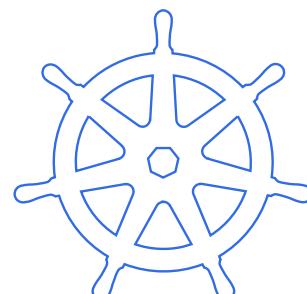
Pods without the taint tolerance won't be scheduled on the node

PreferNoSchedule

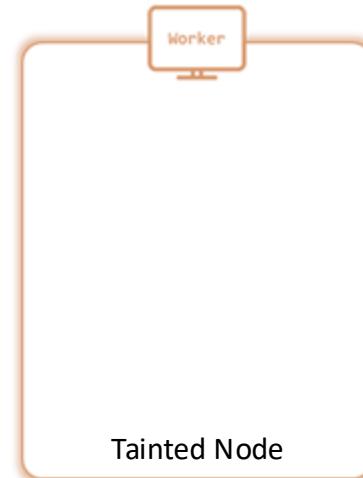
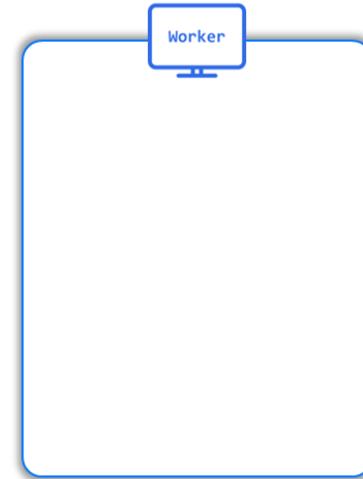
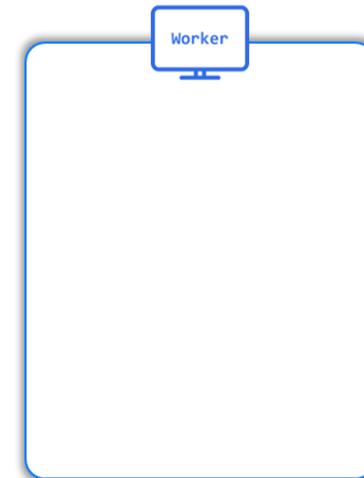
Scheduler attempts to avoid placing non-tolerant pods, but it's not guaranteed

NoExecute

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



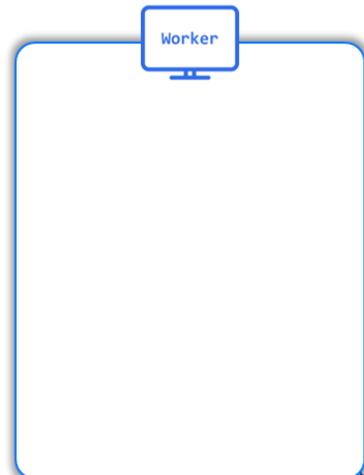
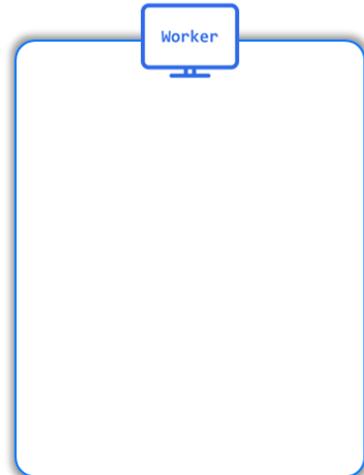
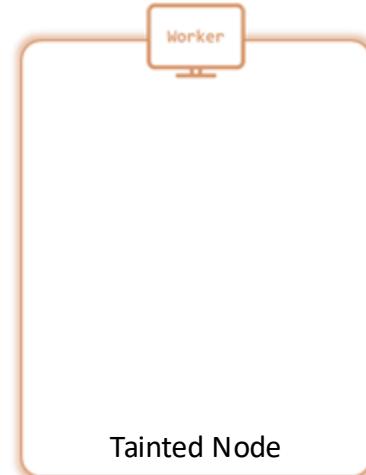
Interaction Between Taints & Toleration



Interaction Between Taints & Toleration

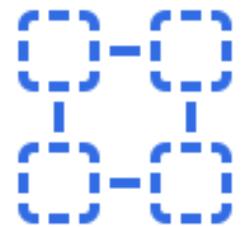


Remained
Unscheduled



Taints & Tolerations

Use-cases



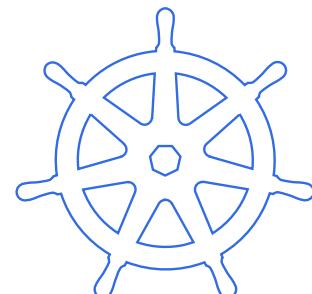
Dedicated
Nodes



Specialized
Hardware

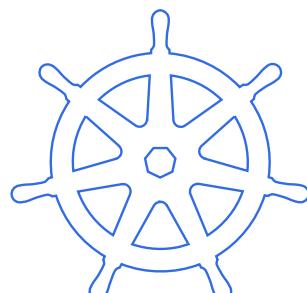


Node
Conditions



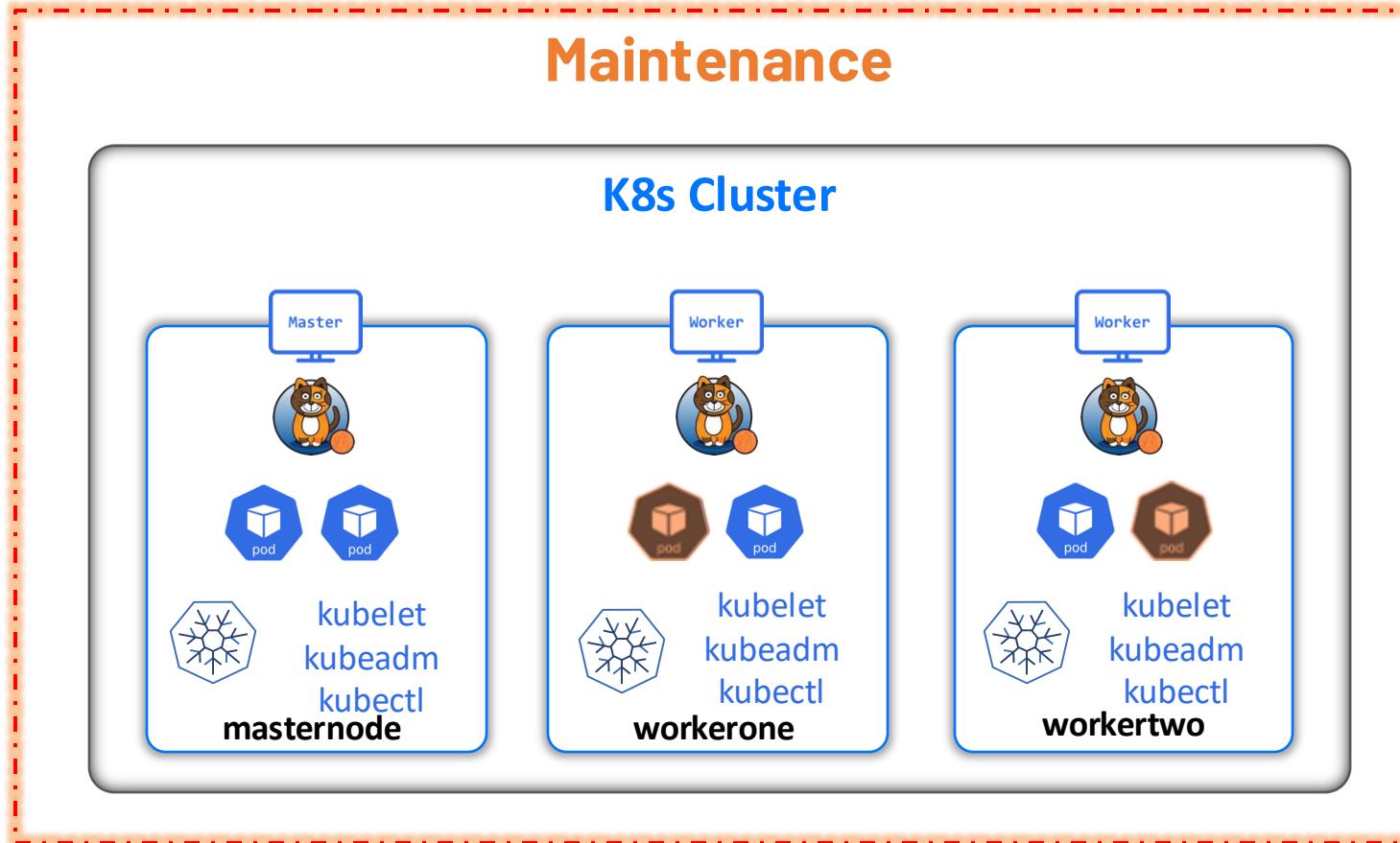
Taints & Tolerations

- 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



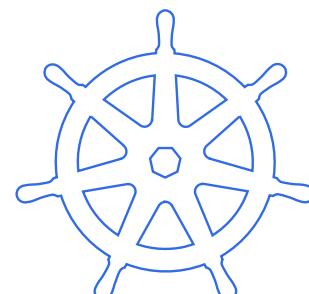
Pod Disruption Budget (PDB)

Pod Disruption Budget (PDB)



PDB

A Kubernetes resource that specifies the minimum number of pods that must remain available during a disruption

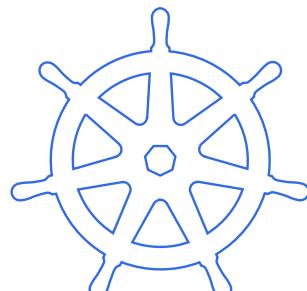


Pod Disruption Budget (PDB)

- Ensures high availability of applications running inside Kubernetes cluster during disruptions
- Defines the minimum number of pods an application requires to function smoothly during disruptions

Voluntary

Involuntary



Pod Disruption Budget (PDB)

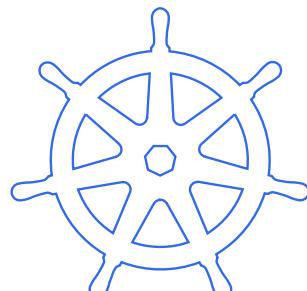
Voluntary

Actions initiated by the application owner and a Cluster Administrator, such as draining a node

Involuntary

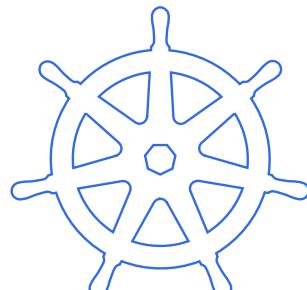
Unavoidable events such as Hardware Failure, VM deletion, a Kernel Panic etc.

Pod Disruption Budget



Pod Disruption Budget (PDB)

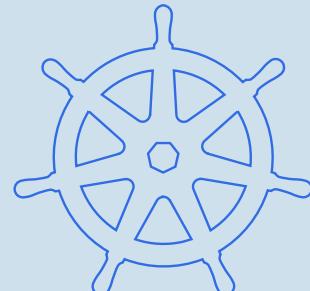
Not all voluntary disruptions are constrained by Pod Disruption Budgets
e.g., deleting deployments or pods bypasses Pod Disruption Budgets





Demo

Pod Disruption Budgets

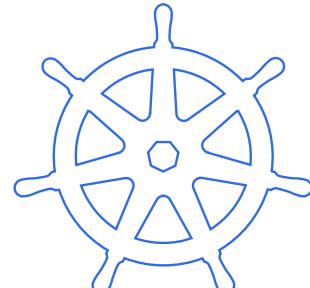


Section: 7

Section Overview

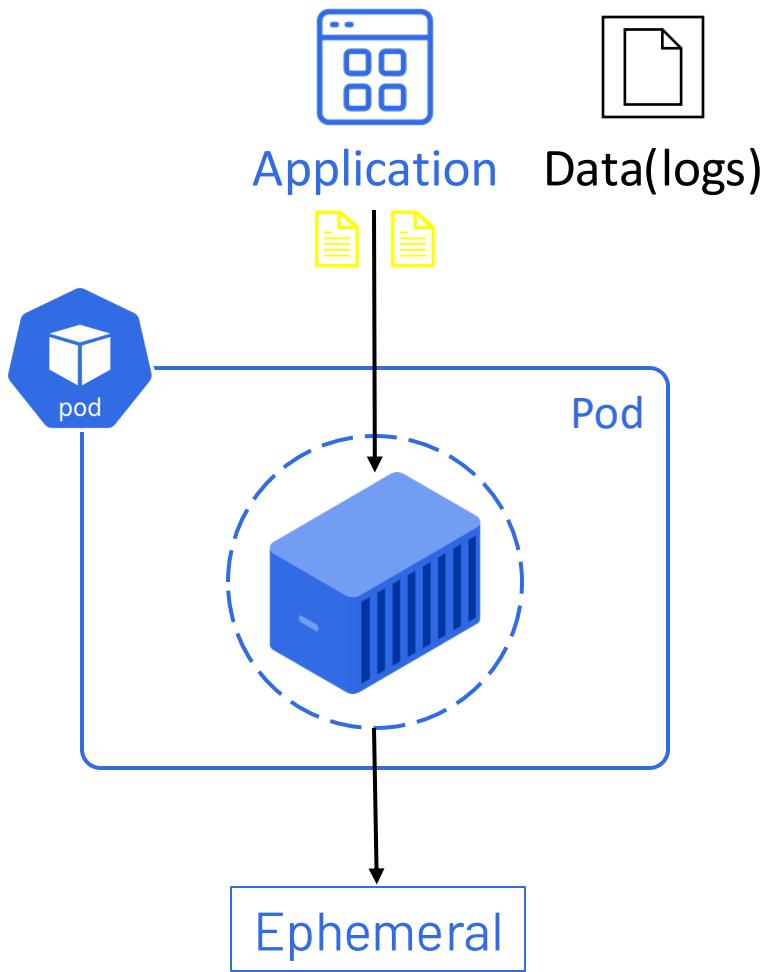
Storage

- └ **Introduction to Volumes**
- └ **Volume Types**
 - **EmptyDir**
 - **HostPath**

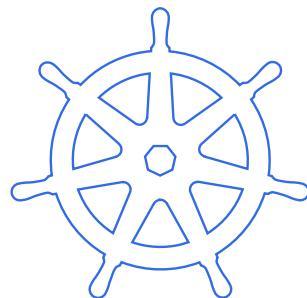


Volumes

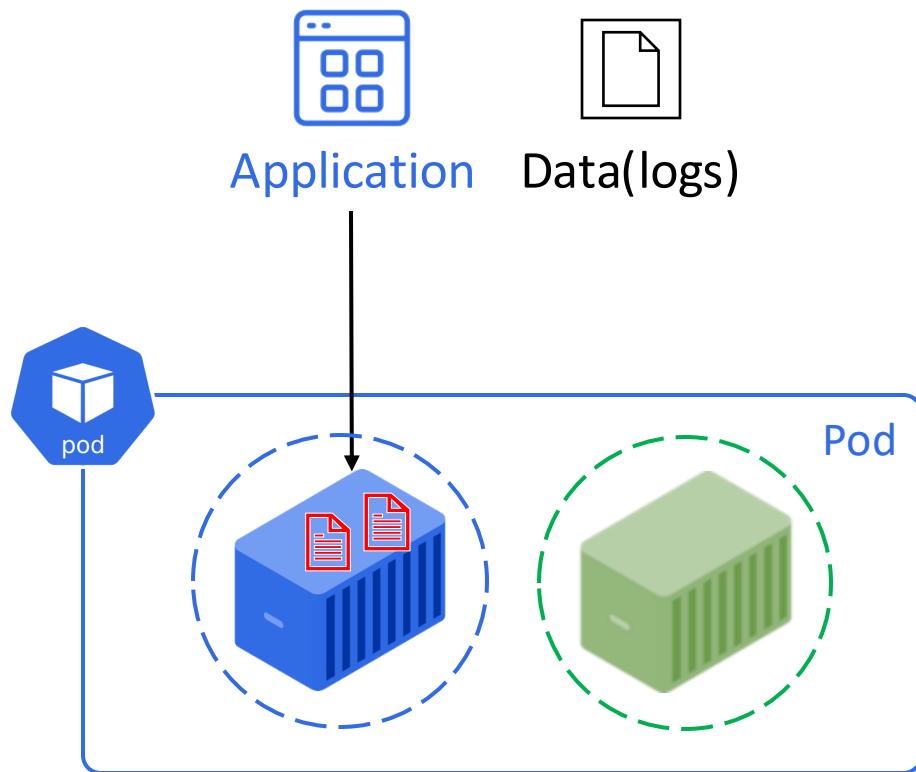
Volumes



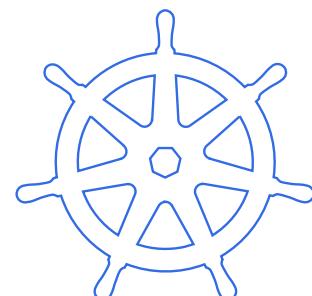
- **Container Crashes**
- **kubelet Restarted Container**



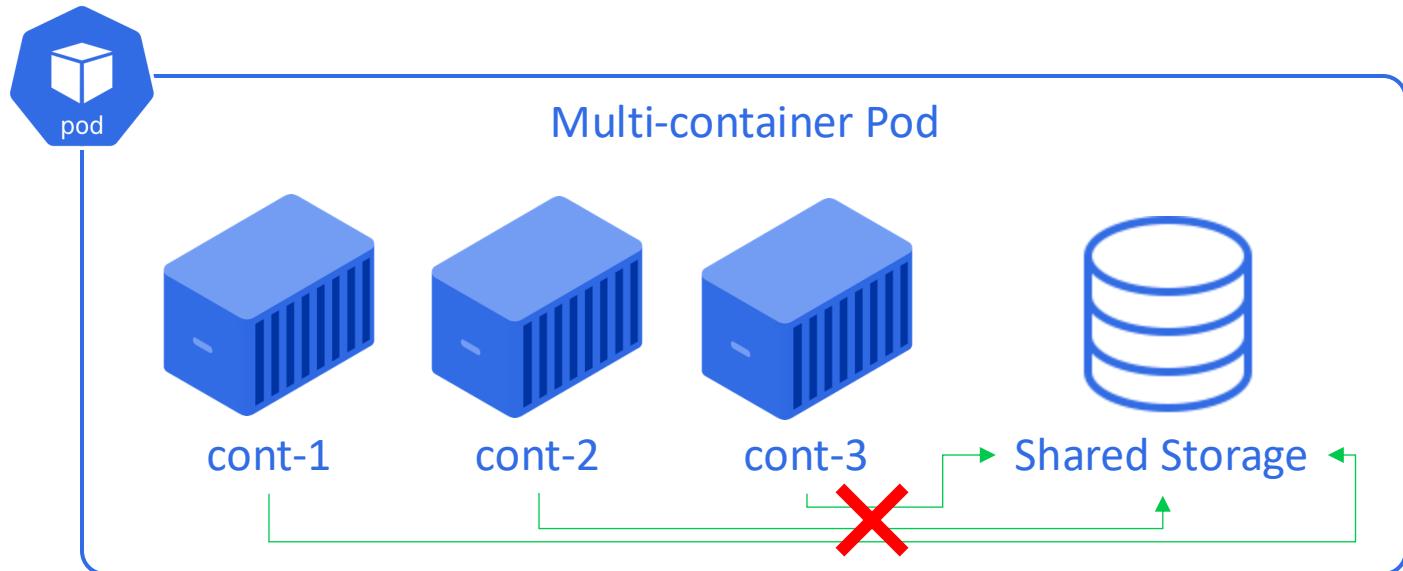
Volumes



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

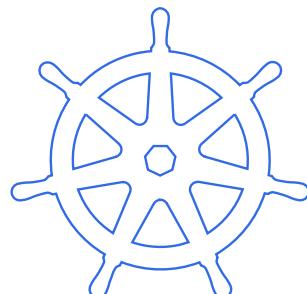


Volumes



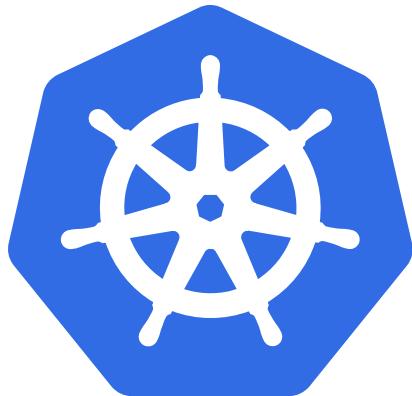
- **Container Crashes**
- **kubelet Restarted Container**
- **Shared Storage for Multi-container pod**

Kubernetes Volumes
Abstraction Layer



Volume Types

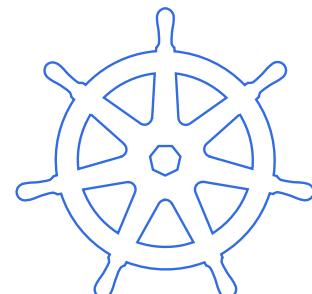
Volume Types



Ephemeral Volume Type

Projected Volume Type

Persistent Volume Type



Volume Types

Ephemeral Volume Type

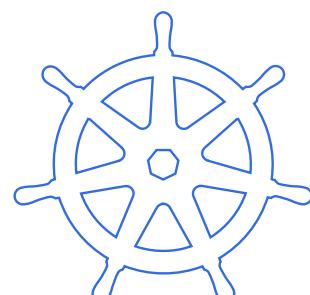
Persistent Volume Type

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

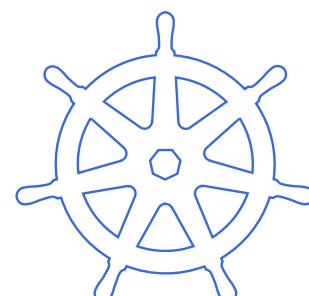
Persistent Volume

Persistent Volume Claim

Storage Classes

Types of Persistent Volumes

- hostpath
- nfs
- iscsi
- fc



Volume Types

Ephemeral Volume Type

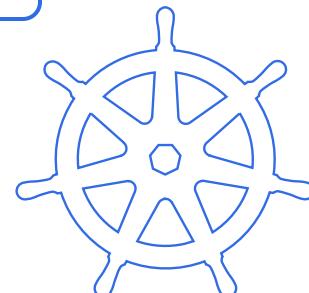
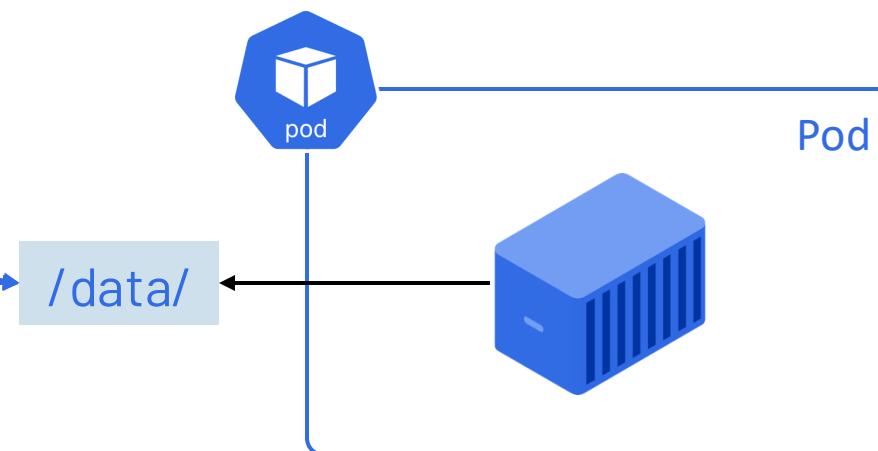
Persistent Volume Type

Projected Volume Type

- All-in-one Volumes

Types of Projected Volumes

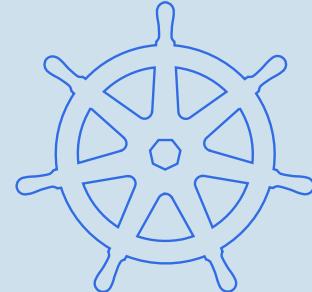
- secret
- ConfigMaps
- serviceAccountToken
- downwardAPI
- clusterTrustBundle





Demo

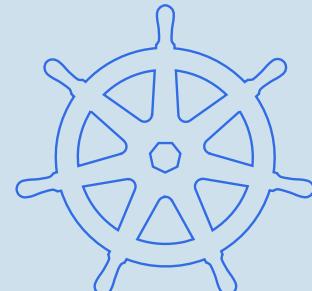
Empty dir Volume Type
(Ephemeral Volumes)





Demo

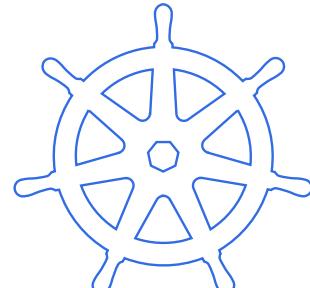
Hostpath Volume Type
(Persistent Volumes)



Section Overview

Storage

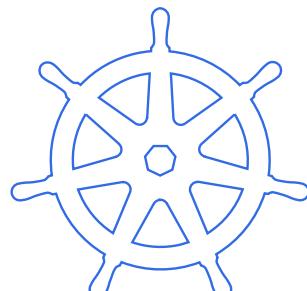
- └ **Persistent Volumes (PVs)**
- └ **Persistent Volume Claims (PVCs)**
- └ **Storage Classes**



Persistent Volume & Persistent Volume Claim

Persistent Volume & Persistent Volume Claim

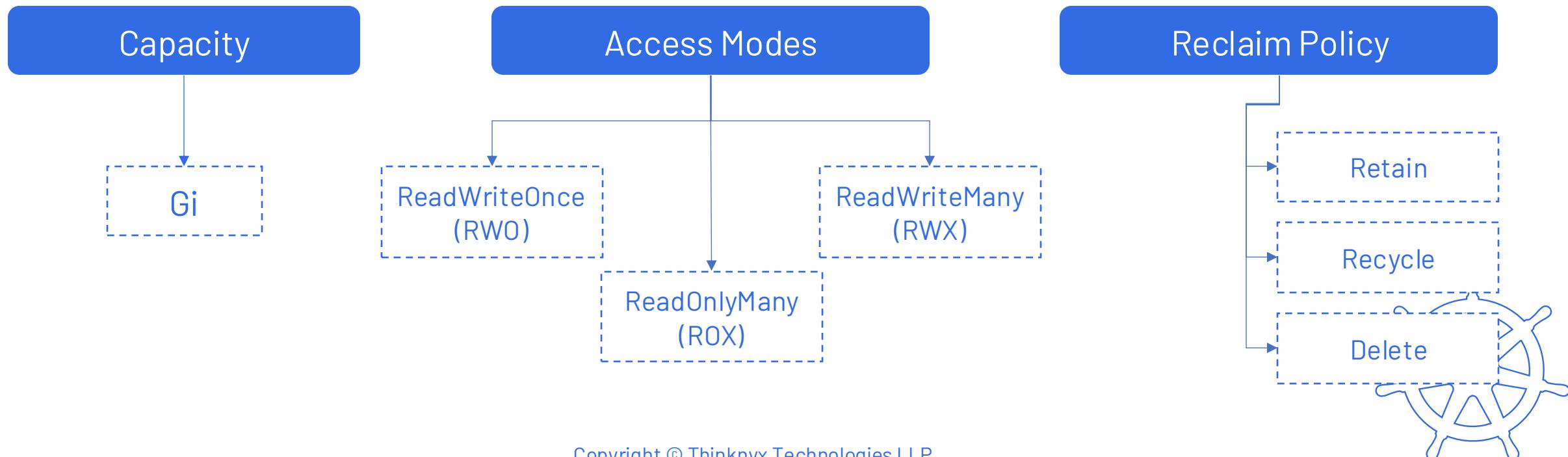
- Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) allow storage management independently of Pods
- They ensure data persists across Pod restarts, rescheduling, and node failures



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



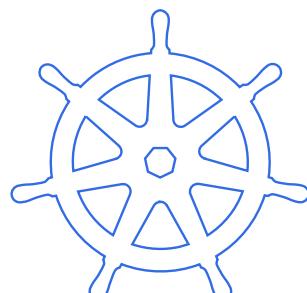
Persistent Volume (PV)

Statically Provisioned

PVs are manually created and defined by administrators in advance

Dynamically Provisioned

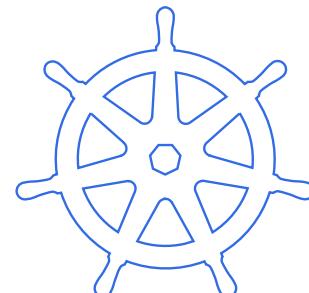
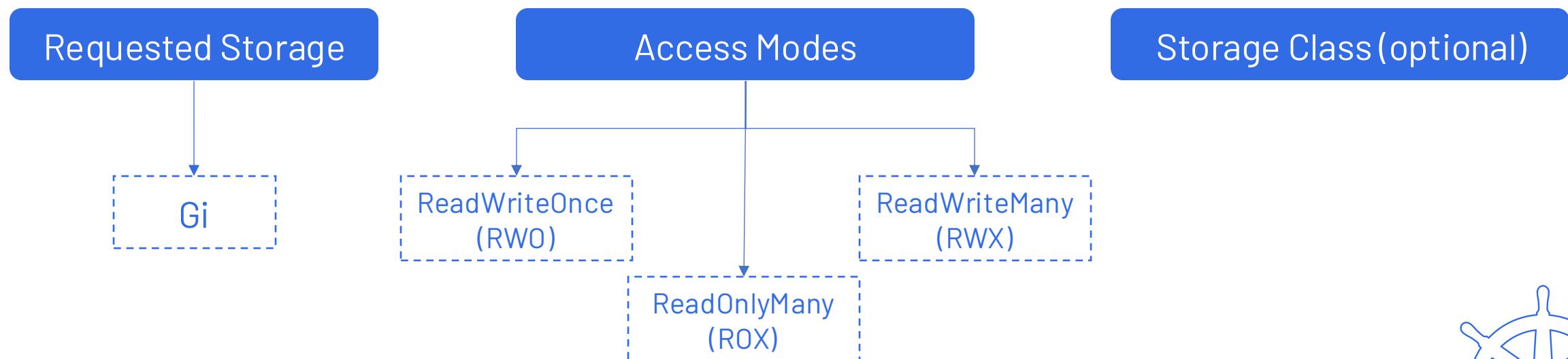
PVs are automatically created based on StorageClass and PVC requests



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



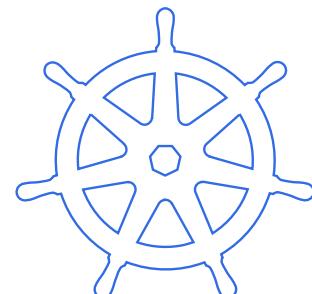
Creating a PVC



Binding



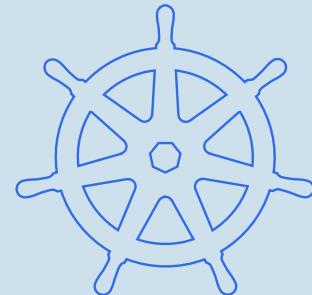
Using the Volume





Demo

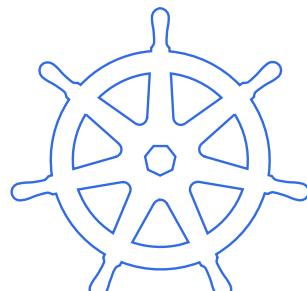
PV and PVC with NFS



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

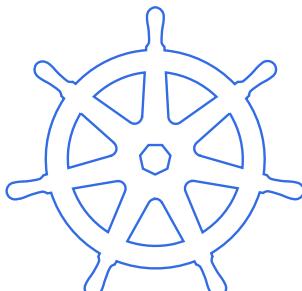
abstract the details of the
underlying storage
infrastructure

PVs

create PVs
dynamically

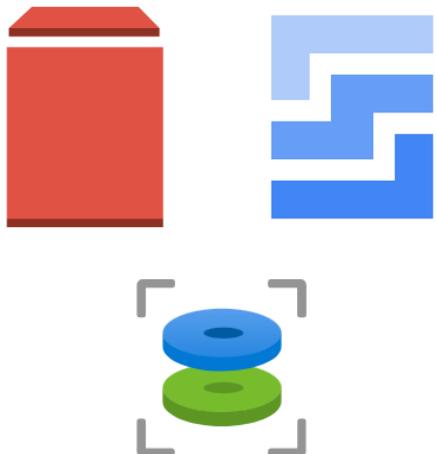
PVC

applications only need to
specify their storage
requirements through PVC



What is a Storage Class?

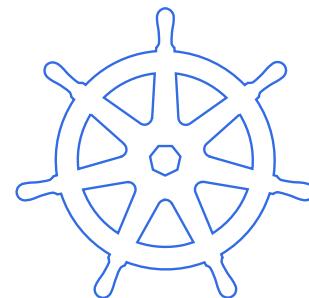
Provisioner



Parameters

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

Reclaim Policy



How Dynamic Provisioning Works

How Dynamic Provisioning Works

Creating a Storage Class



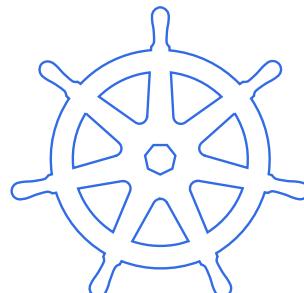
Requesting Storage through PVC



Automatic PV Creation & Binding



Using the Storage

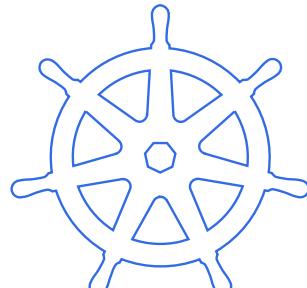


storageClassName

Section Overview

Storage

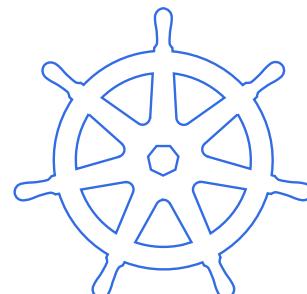
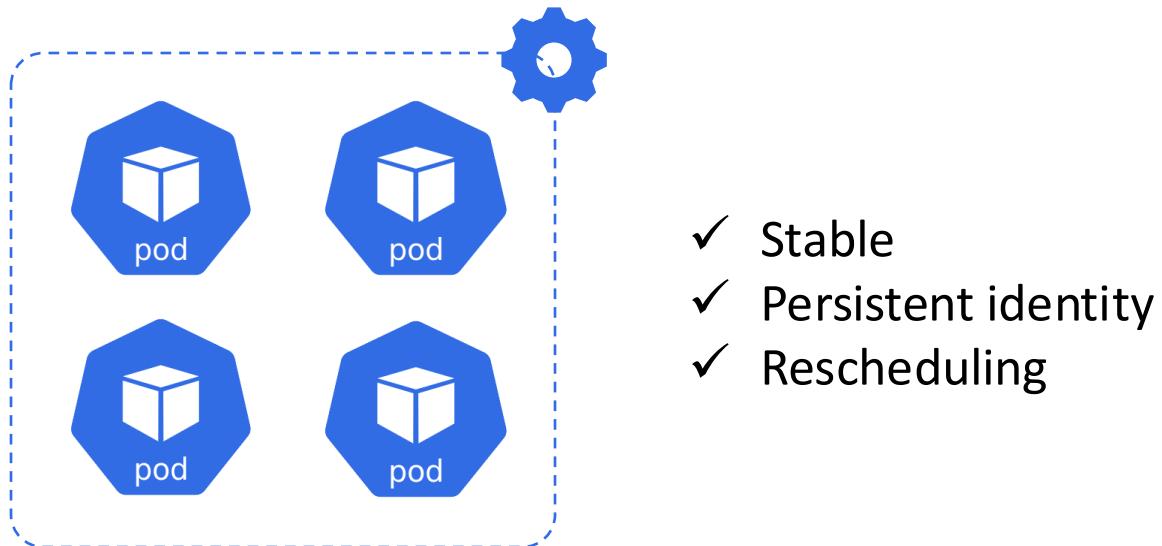
- ↳ **StatefulSets**
- ↳ **How StatefulSets help manage stateful applications**



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



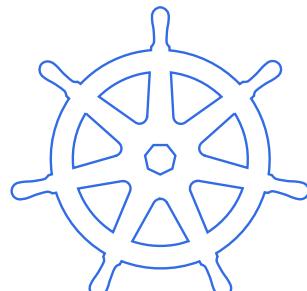
StatefulSets

- 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

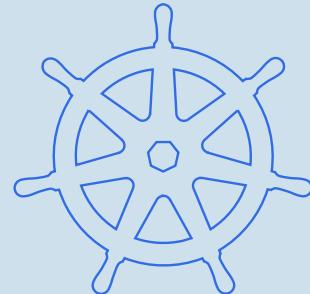
StatefulSets

PersistentVolumes (PVs)





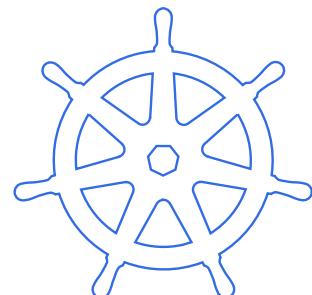
Demo | StatefulSets



StatefulSets vs Deployments

StatefulSets

Deployment



StatefulSets

- StatefulSets are designed for stateful applications, where each pod must retain its identity and state
- Stateful applications like databases need persistent storage and require each pod in the StatefulSet to have a unique, consistent identity



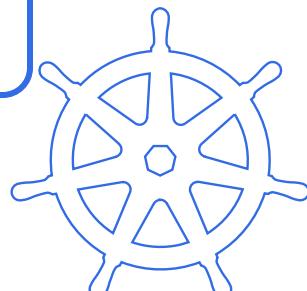
Identity & Ordering



Persistent Storage

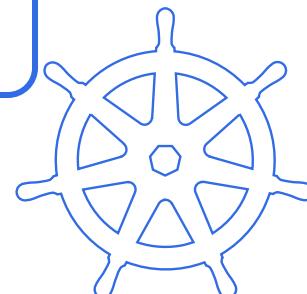
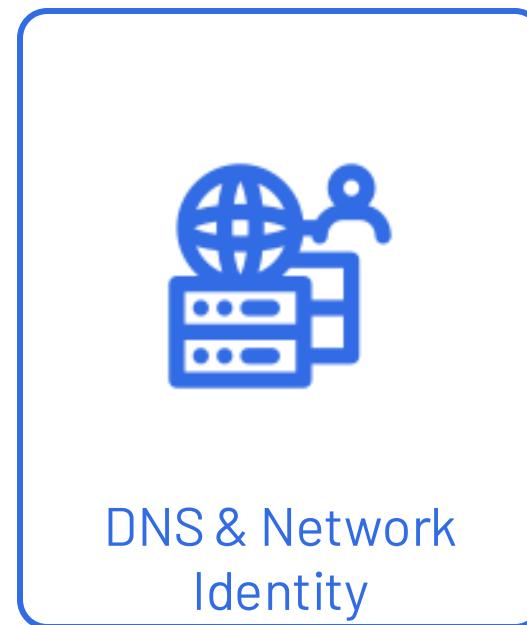
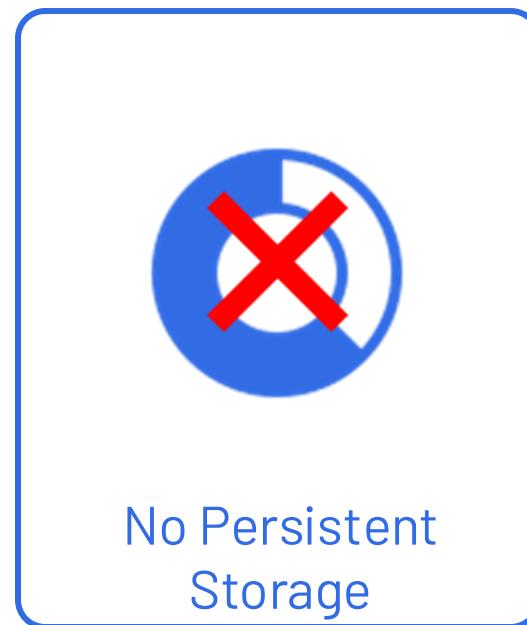
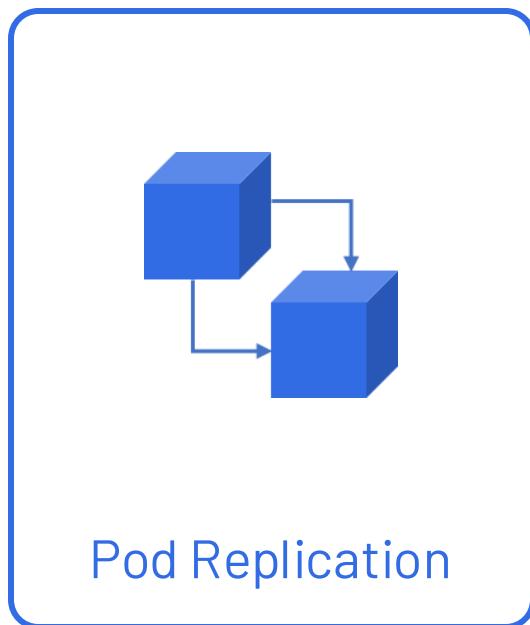


DNS & Network
Identity



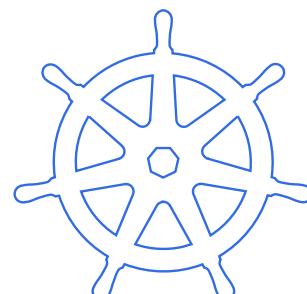
Deployments

- Deployments are suited for stateless applications
- These applications do not require the same identity and state management as stateful ones



StatefulSets vs Deployments

Network Policies	Network Policies	Network Policies
State Management	For apps that need to keep data and state	For stateless apps that don't need to keep state.
Pod Identity	Each pod has a unique, persistent name	Pods have random names, no identity
Storage	Each pod has its own storage that persists	No persistent storage needed
Scaling & Pod Creation Order	Pods are created/deleted in order	Pods can be created/deleted in any order



Section: 8

Kubernetes Security

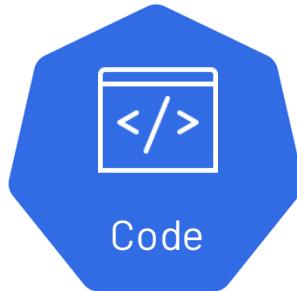
Kubernetes Security

CIS Kubernetes Benchmark

Focuses on enhancing cybersecurity readiness & response across public and private sectors

CIS Controls

CIS Benchmarks



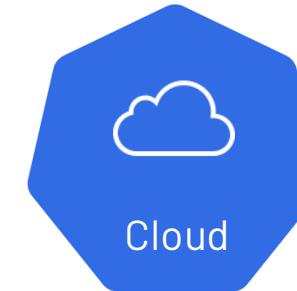
Code



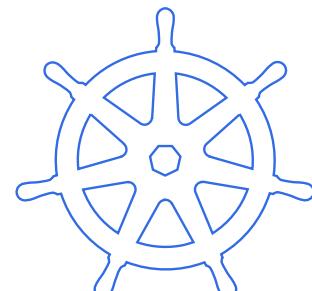
Container



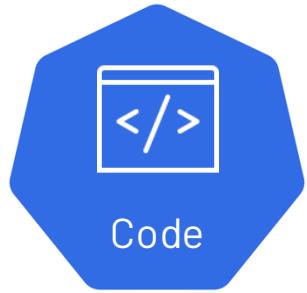
Cluster



Cloud



Kubernetes Security



Code



Container

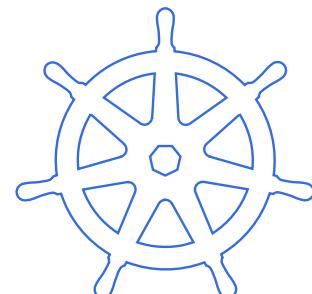


Cluster

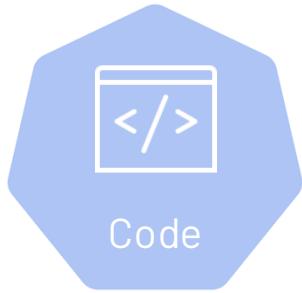


Cloud

Secure coding practices and regular testing help prevent vulnerabilities in applications



Kubernetes Security



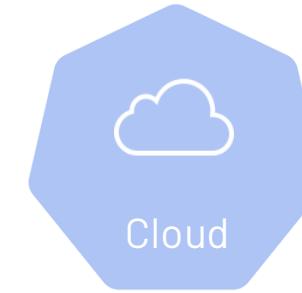
Code



Container

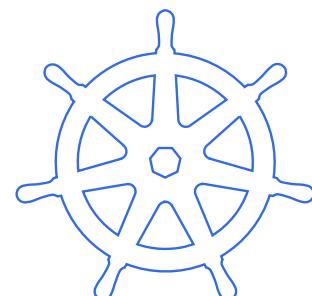


Cluster

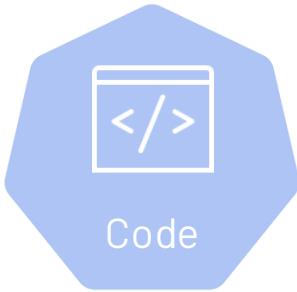


Cloud

Containers package apps with dependencies, but require scanning for vulnerabilities and secure base images



Kubernetes Security



Code



Container

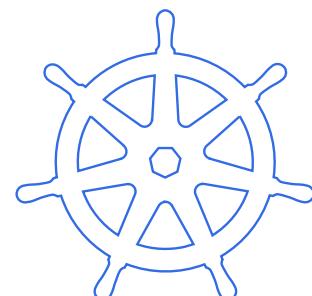


Cluster

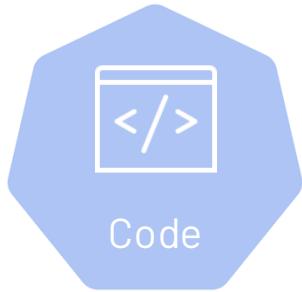


Cloud

Kubernetes clusters need proper RBAC, network policies, and audit logging to secure against unauthorized access



Kubernetes Security



Code



Container

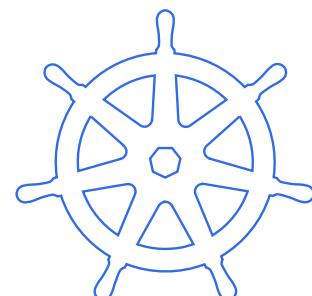


Cluster

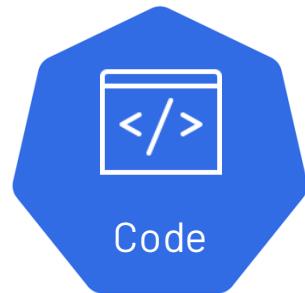


Cloud

Cloud security focuses on managing configurations, enforcing access controls, and monitoring for misconfigurations



Kubernetes Security



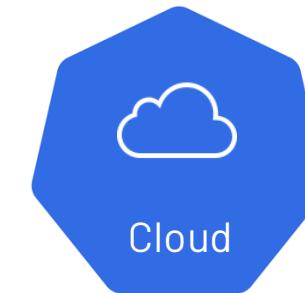
Code



Container

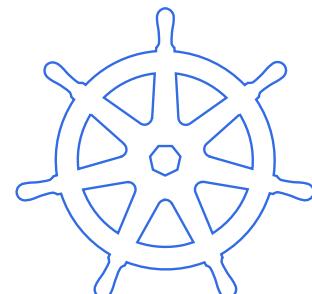


Cluster



Cloud

CIS Benchmarks



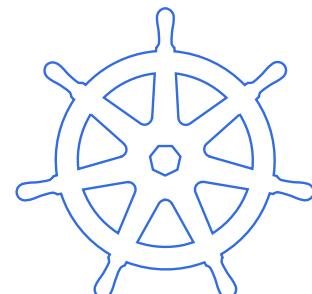
Securing Kubernetes Cluster

Securing Kubernetes Cluster

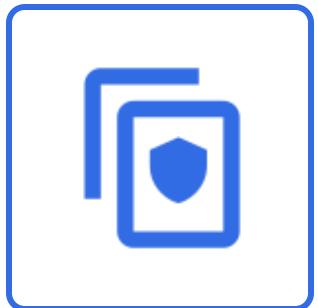


RBAC

- Apply the principle of least privilege by granting only necessary permissions
- Use roles and bindings to manage access to Kubernetes resources
- Regularly audit RBAC rules to ensure compliance

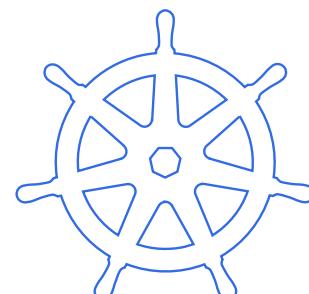


Securing Kubernetes Cluster

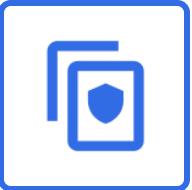


- Control traffic flow between pods and namespaces
- Limit communication to necessary services, like restricting database access to specific pods or namespaces

Network Policies



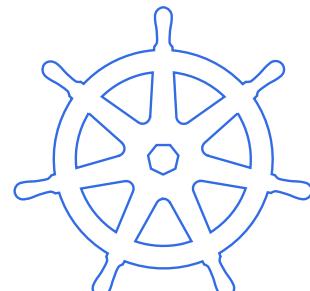
Securing Kubernetes Cluster



Security Contexts

- Set security contexts to enforce restrictions like running as non-root, preventing privilege escalation, and using read-only file systems where possible

```
securityContext:  
  runAsNonRoot: true  
  readOnlyRootFilesystem: true  
  allowPrivilegeEscalation: false
```

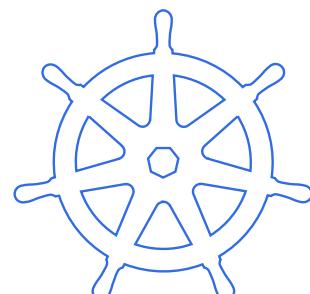


Securing Kubernetes Cluster



- Use admission controllers (like OPA or Kyverno) to enforce security policies, e.g., deny deployments that don't define network policies or attempt to run as root

Admission Controllers
& Policies

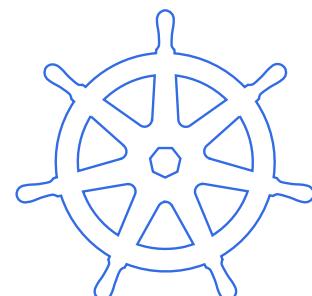


Securing Kubernetes Cluster

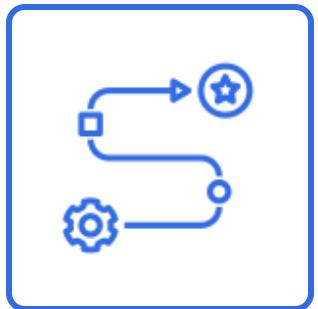
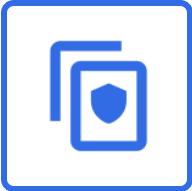


Audit Logging

- Enable Kubernetes audit logs to track activities, detect anomalies, and monitor unauthorized access for faster incident response

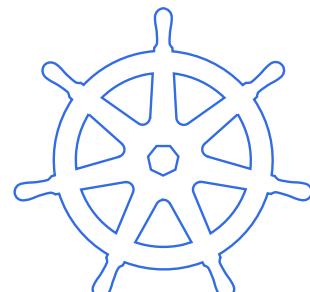


Securing Kubernetes Cluster

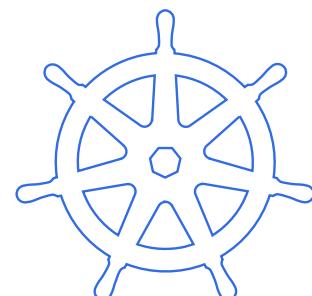
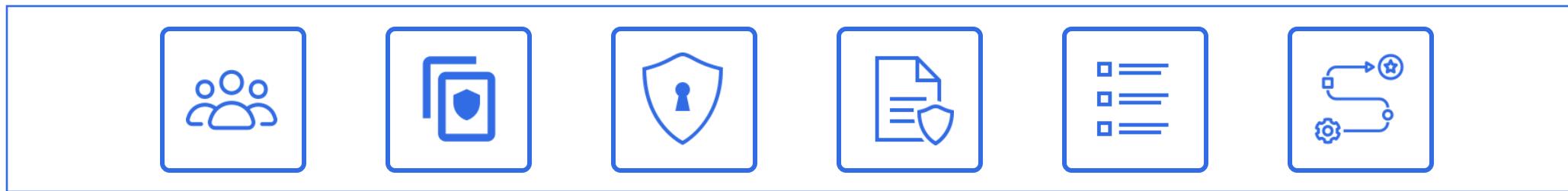


CIS Kubernetes
Benchmark

- Follow the CIS Kubernetes Benchmark for cluster hardening. Automate validation with tools like kube-bench to ensure compliance with best practices



Securing Kubernetes Cluster



Securing Containers

Securing Containers

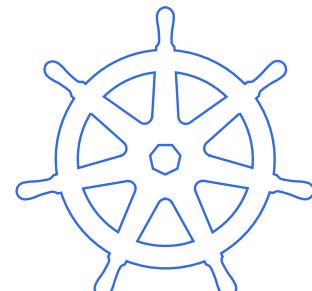
Image
Scanning

Runtime
Scanning

Immutable
Infrastructure



 clair
anchore



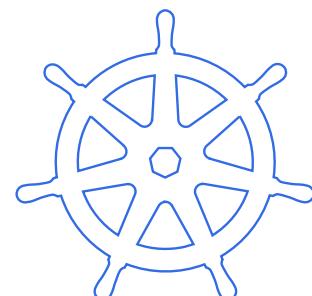
Securing Containers



aqua
trivy



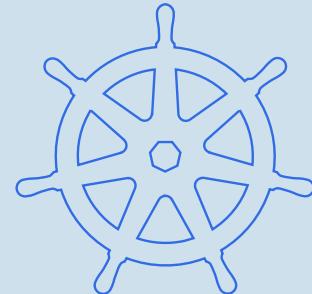
aqua
kube-bench





Demo

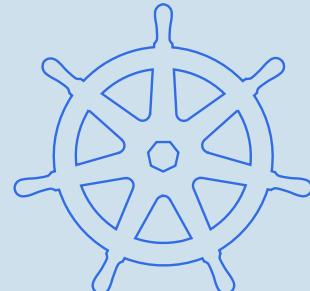
Image and
Manifests Scans





Demo

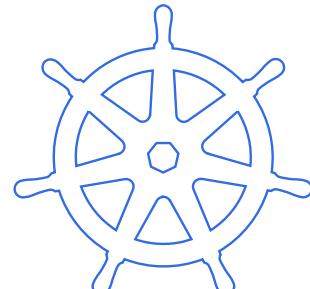
CIS benchmarking
with Kube-bench



Network Policies

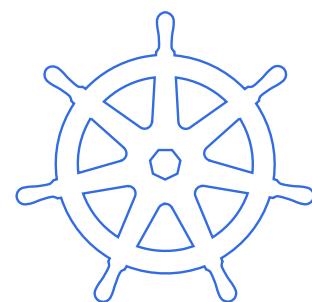
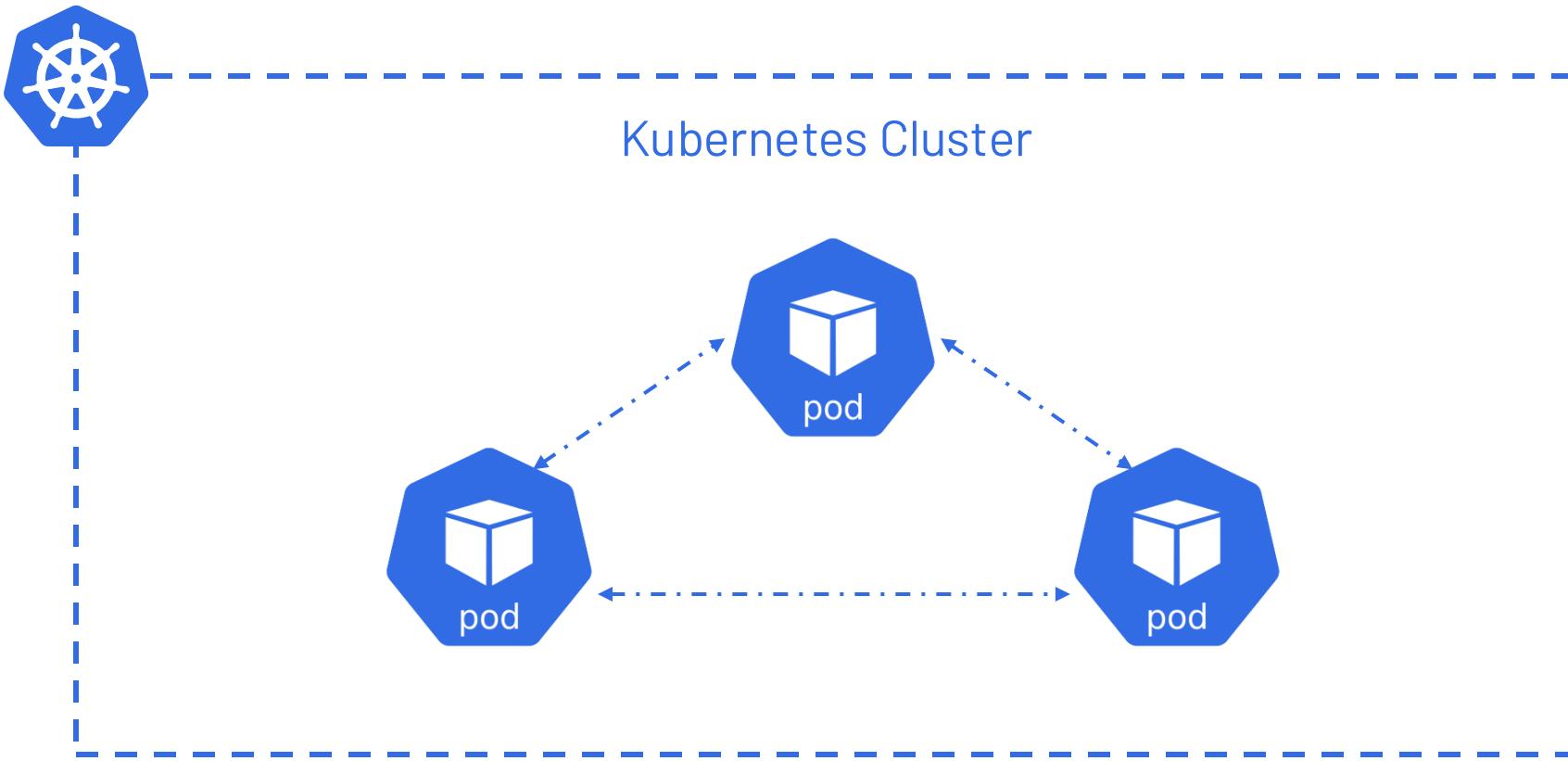
Section Overview

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



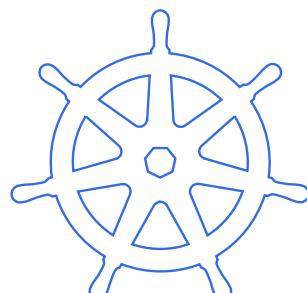
Network Policies

- 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

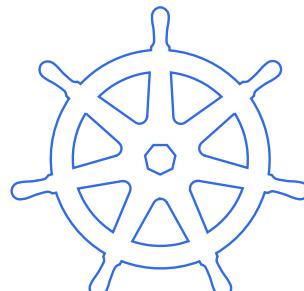
```
namespaceSelector:  
  matchLabels:  
    environment: production
```

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

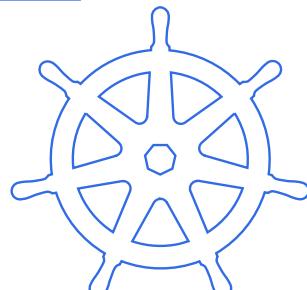
- Network Policy defines rules that control traffic by specifying allowed sources for incoming or outgoing connections based on various selectors and criteria

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:
```

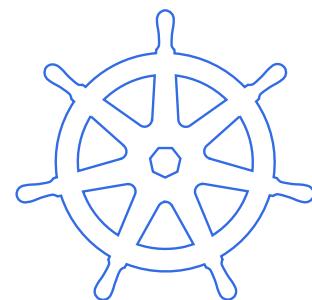
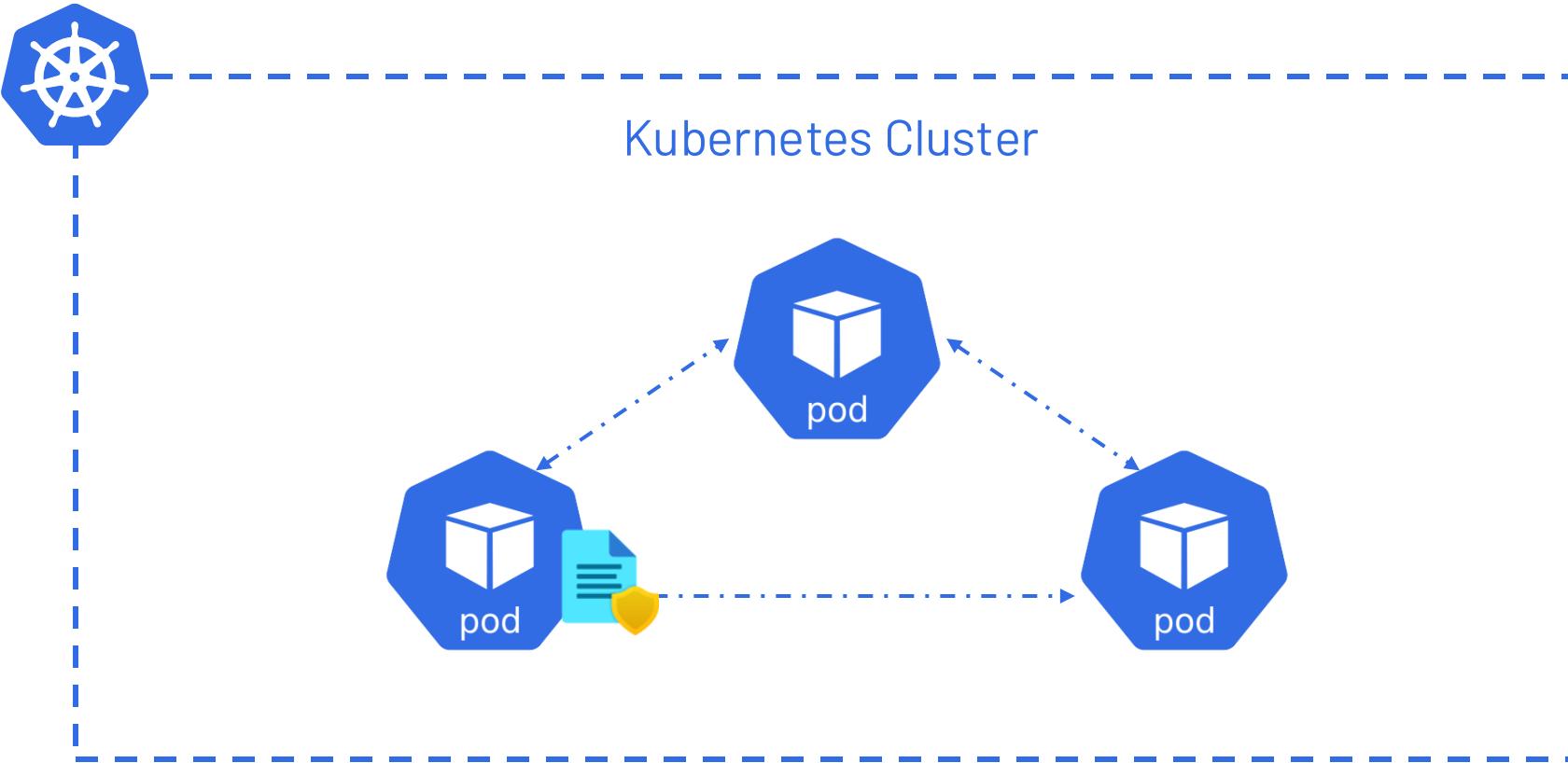
```
        matchLabels:
```

```
          app: thinknyxns
```



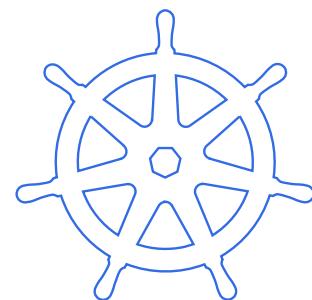
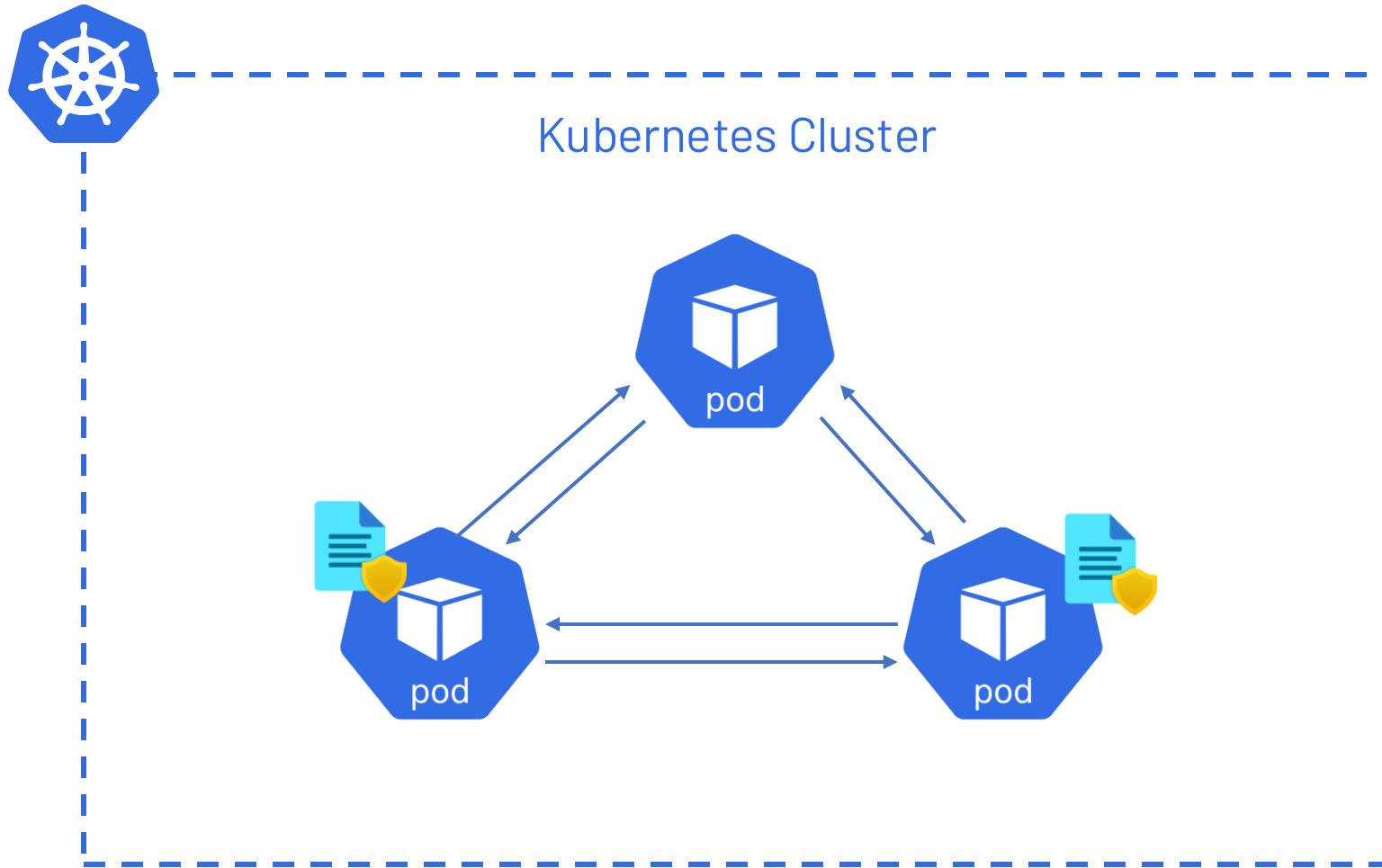
Network Policies

Default behavior



Network Policies

Default behavior



Network Policies

Limitations



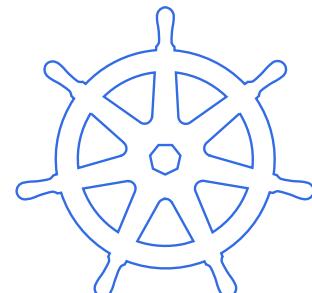
Scope



Kubernetes
Network Plugins



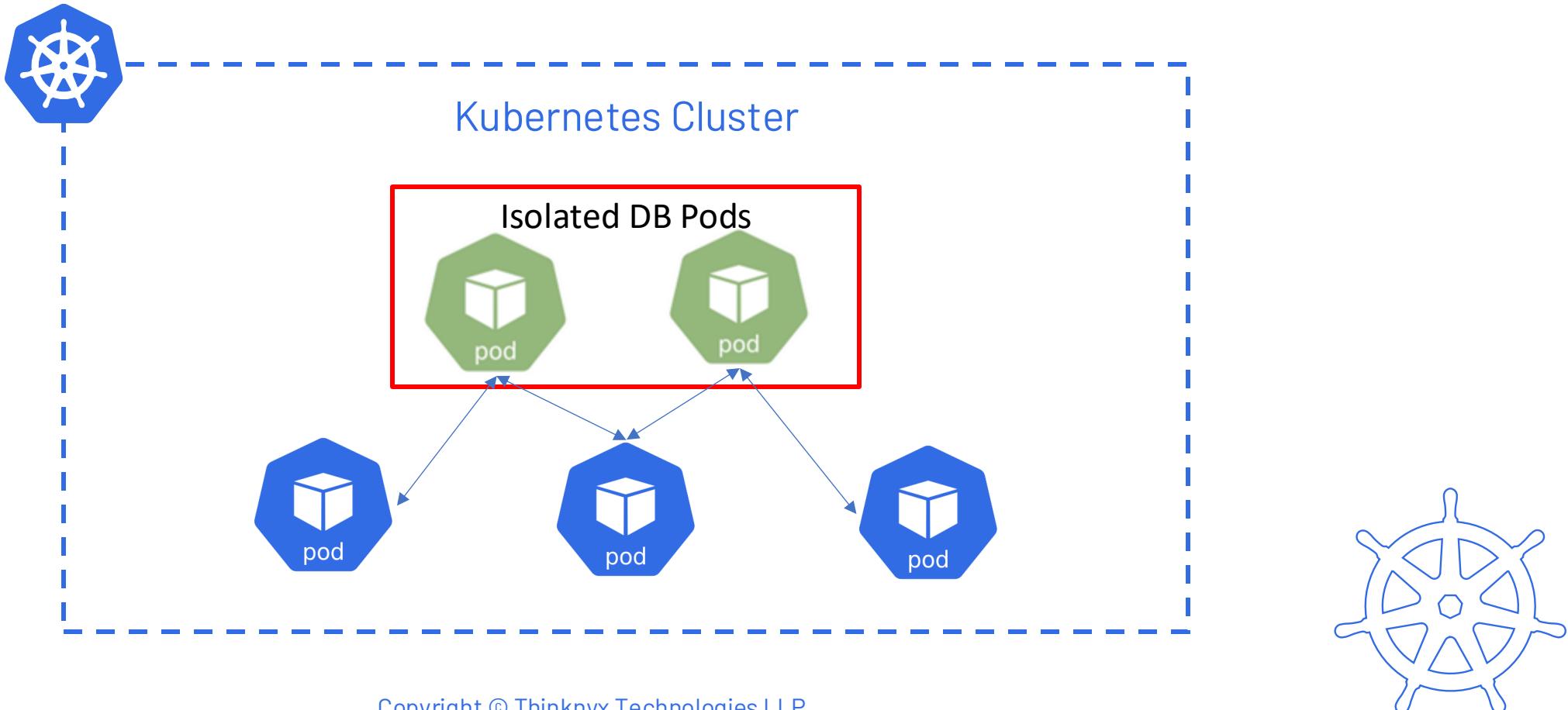
Cluster External
Traffic



Network Policies

Practical Uses

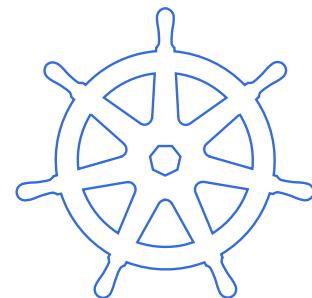
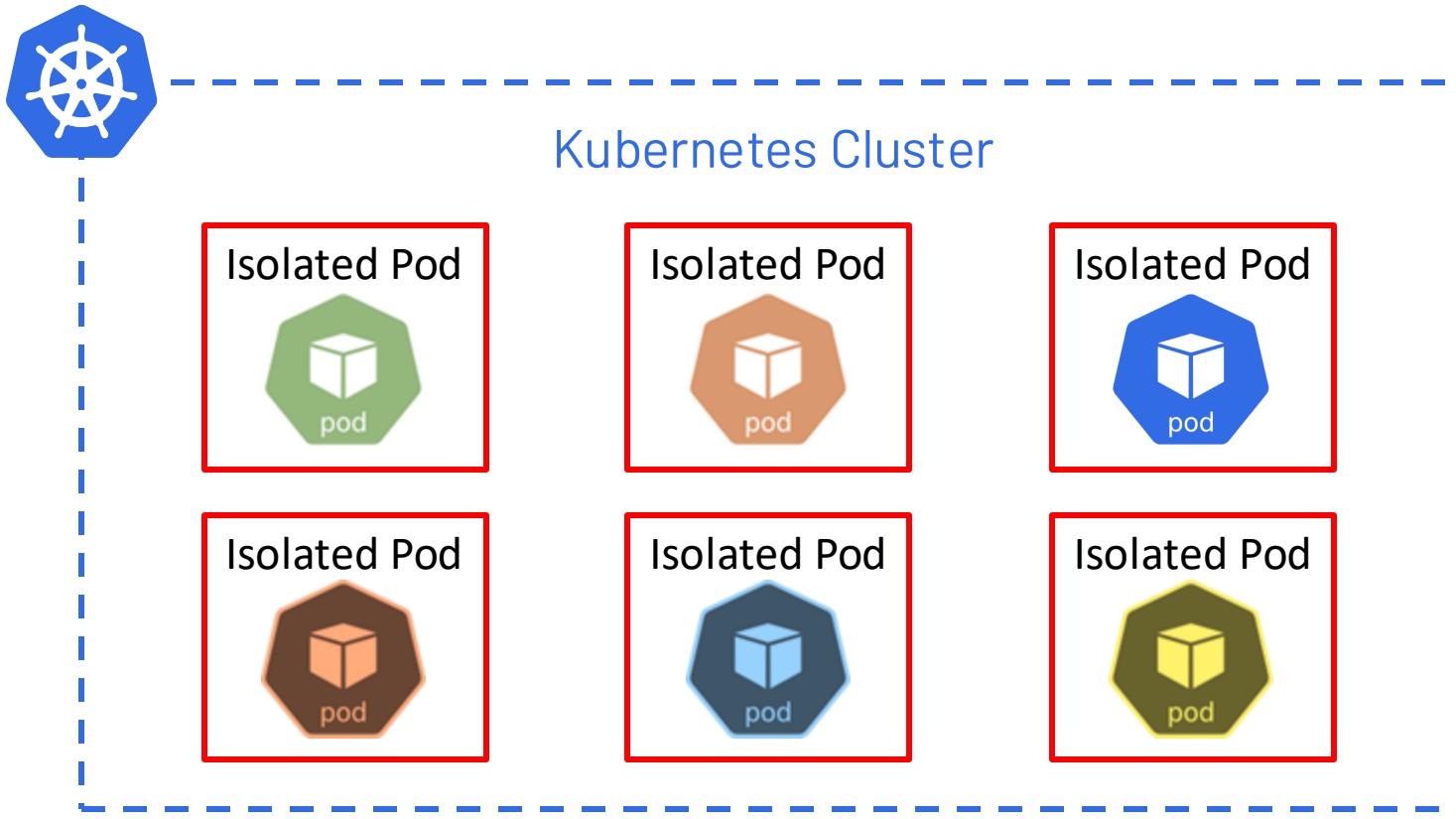
- Isolating Sensitive Pods



Network Policies

Practical Uses

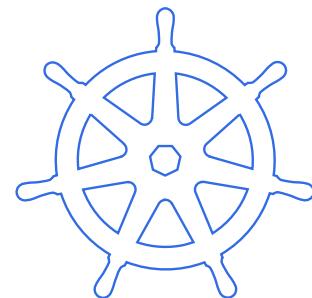
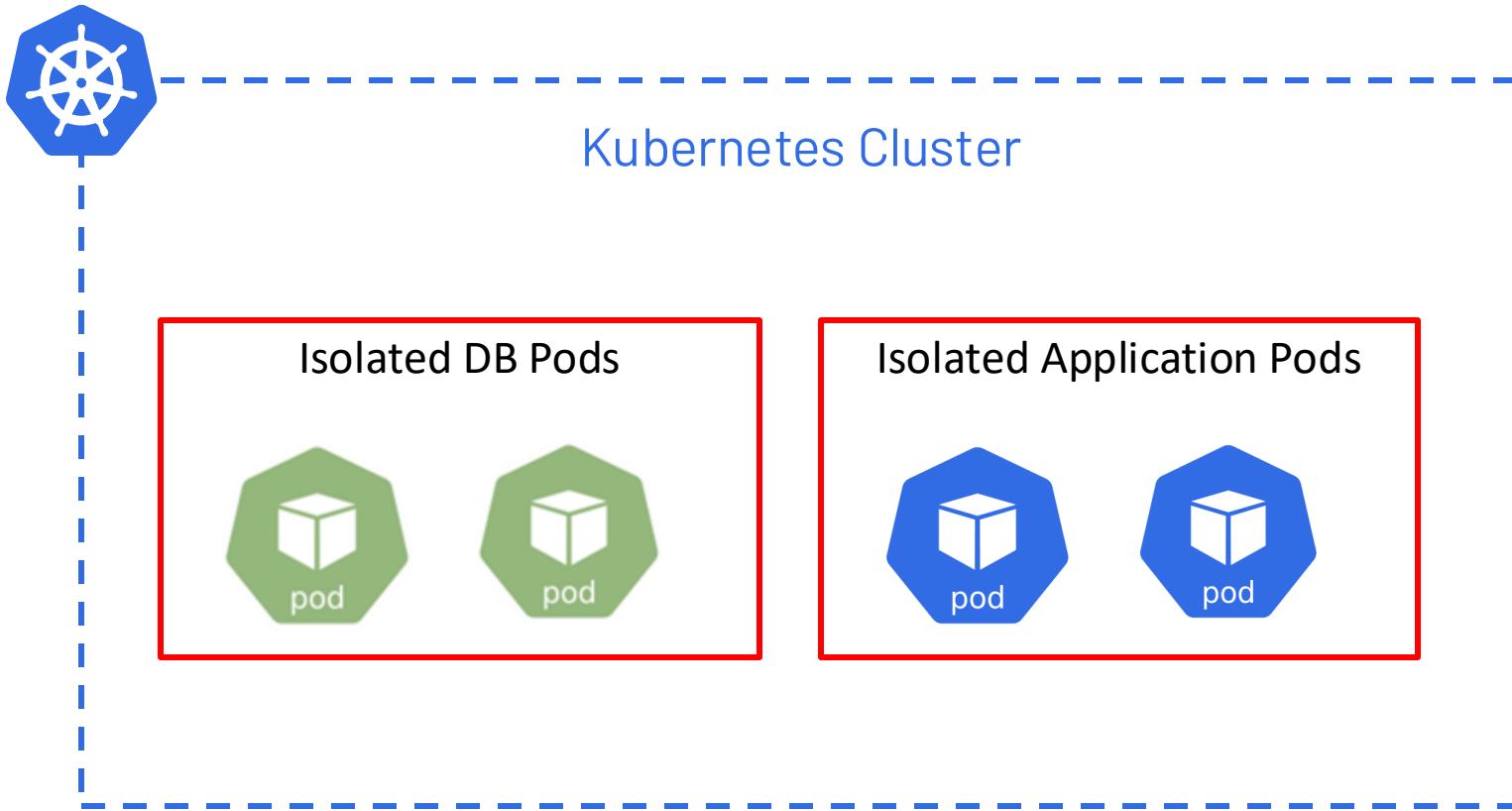
- Securing Microservices



Network Policies

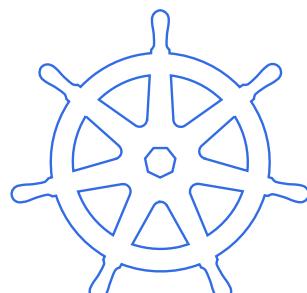
Practical Uses

- Regulatory Compliance



Network Policies

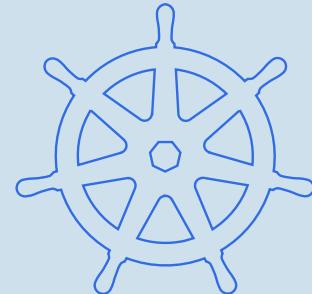
- Network Policies control how pods communicate inside Kubernetes clusters
- Enhance cluster security by enforcing rules that restrict traffic based on defined criteria
- Administrators can define incoming and outgoing traffic to prevent unauthorized access





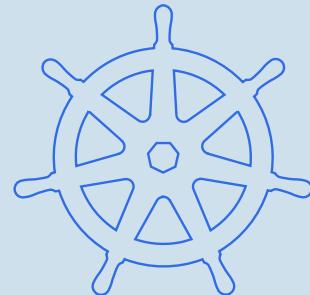
Demo

Default All Allow
Policy





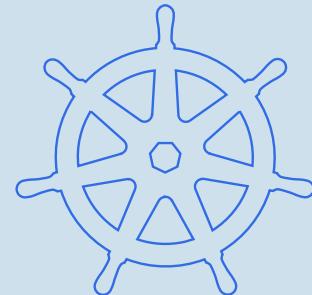
Demo | All Deny Policy





Demo

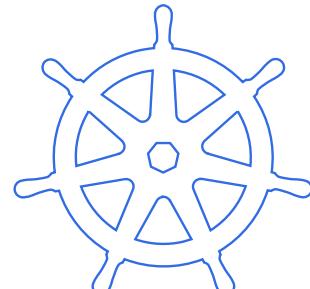
Namespace Scoped Policy



RBAC in Kubernetes

Section Overview

- ↳ RBAC (Role-Based Access Control)



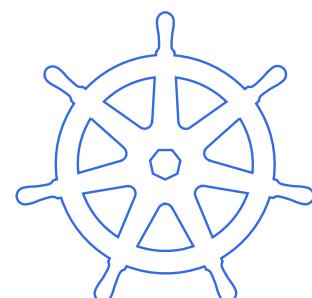
Role-Based Access Control

Roles

Role Bindings

Cluster Roles

Cluster Role Bindings



Role-Based Access Control

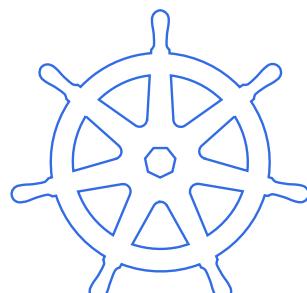
Roles

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

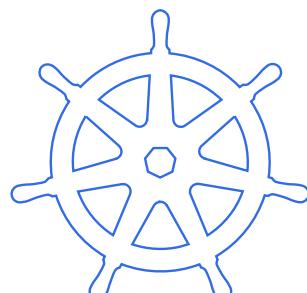
Roles

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

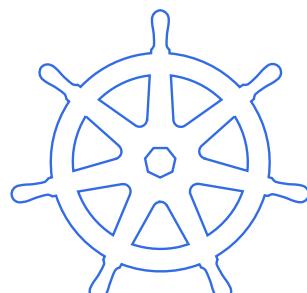
Roles

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

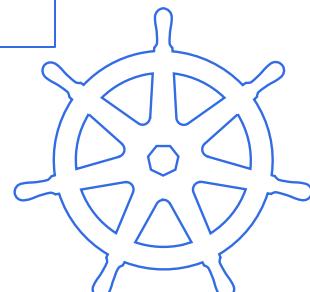
Roles

Role Bindings

Cluster Roles

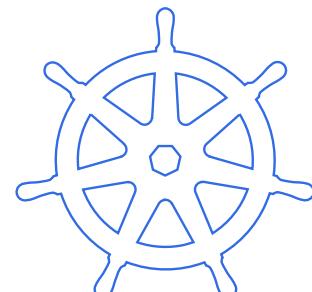
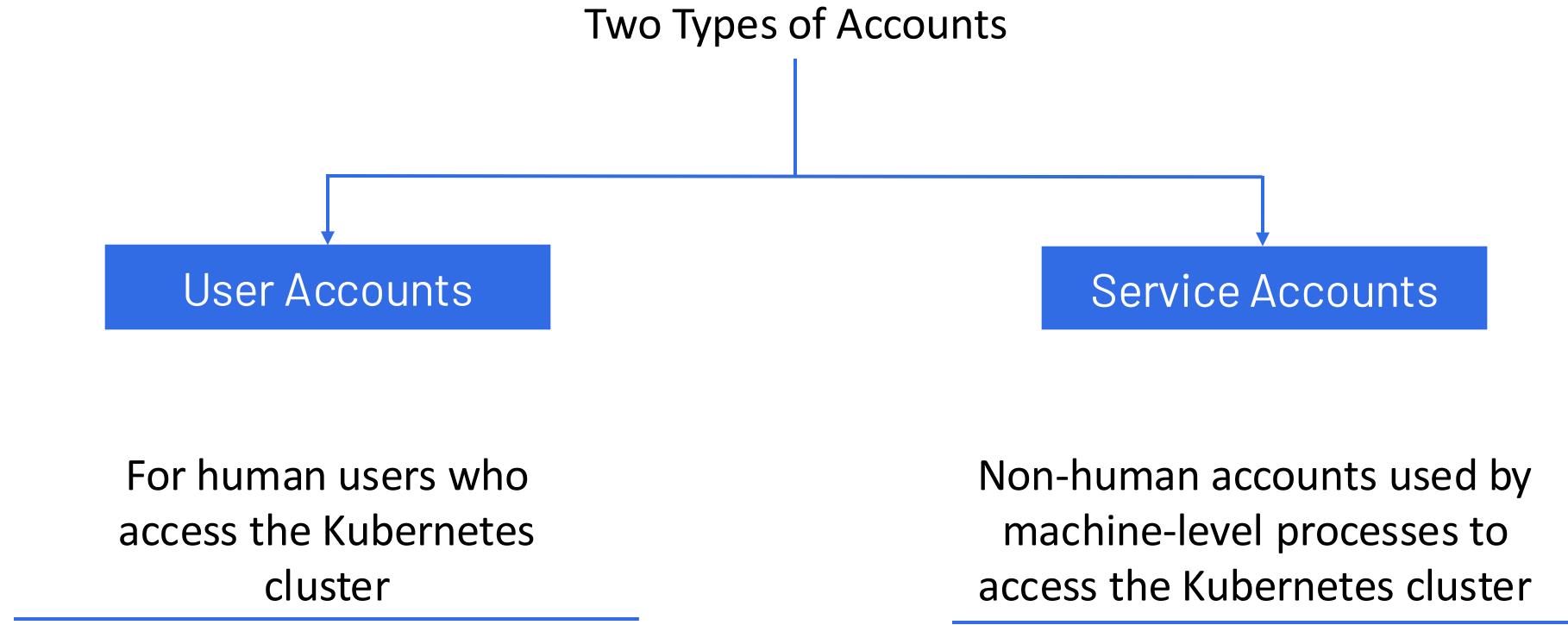
Cluster Role Bindings

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



Service Accounts

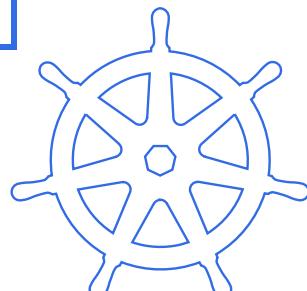
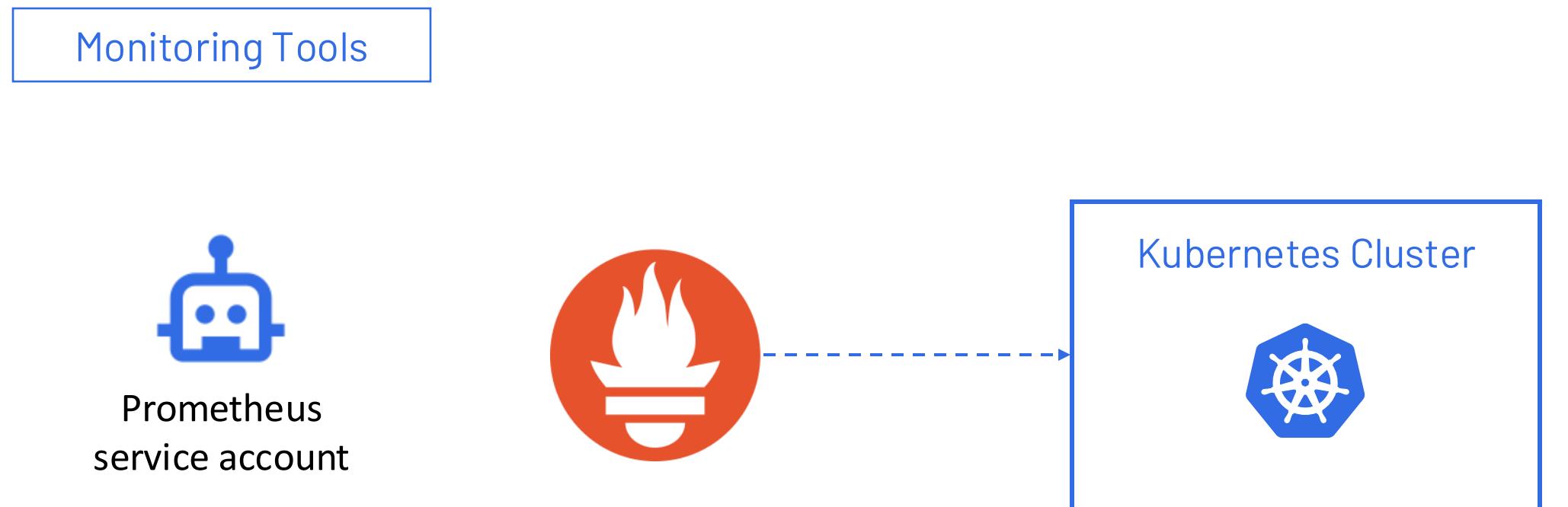
Service Accounts



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

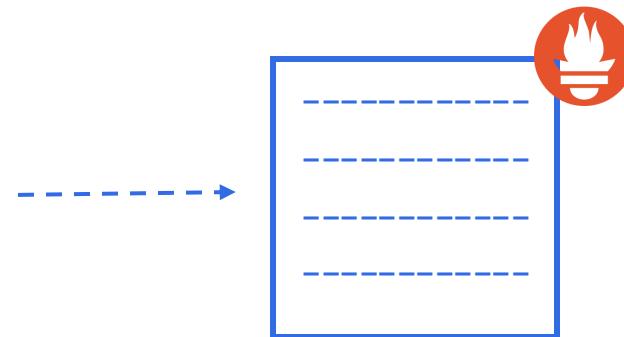


Prometheus
service account

authenticate

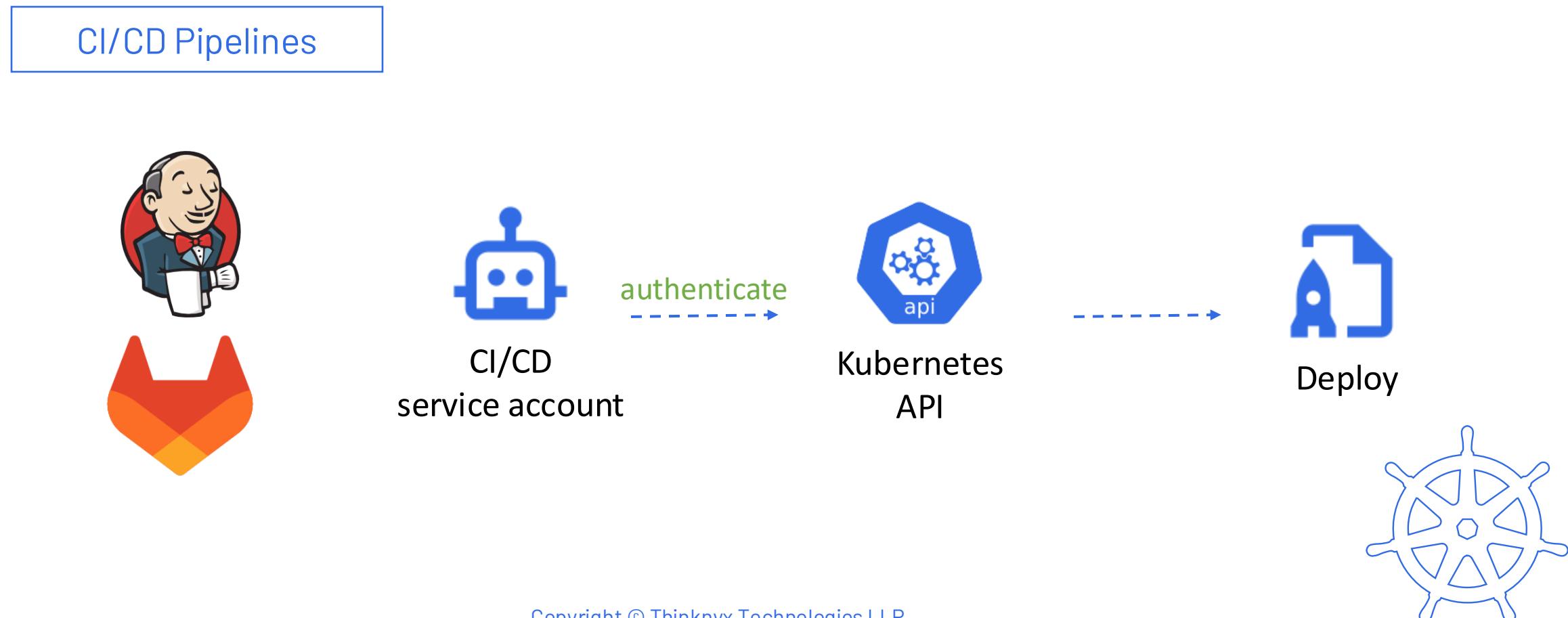


Kubernetes
API



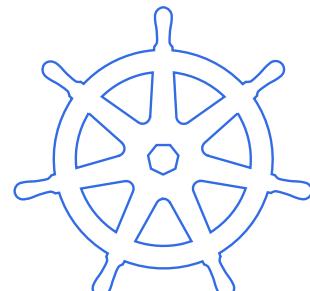
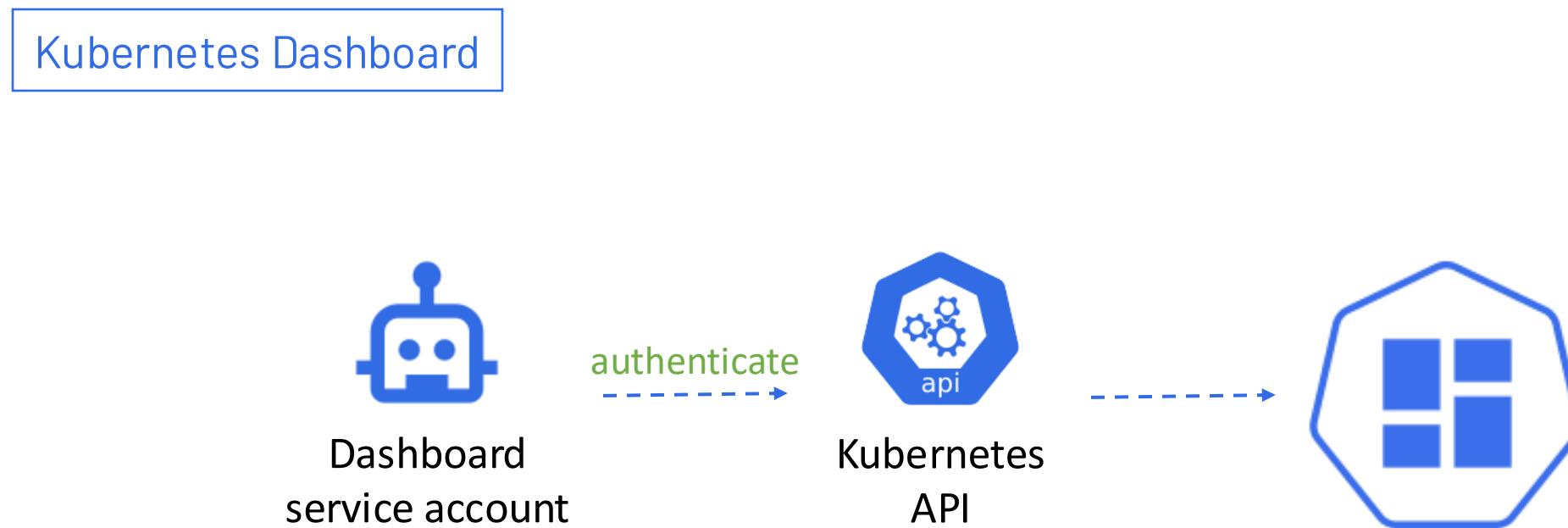
Why Service Accounts?

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



Why Service Accounts?

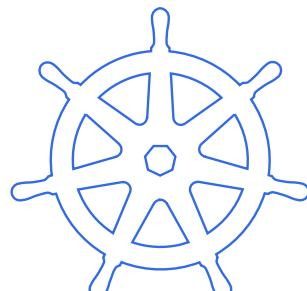
If two applications in the cluster need to communicate, they use service accounts instead of user accounts



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

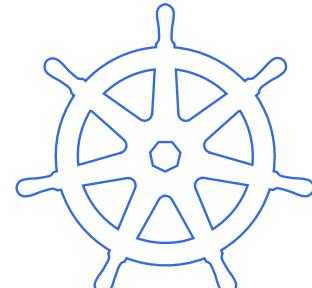


Section: 9

Service Meshes

Section Overview

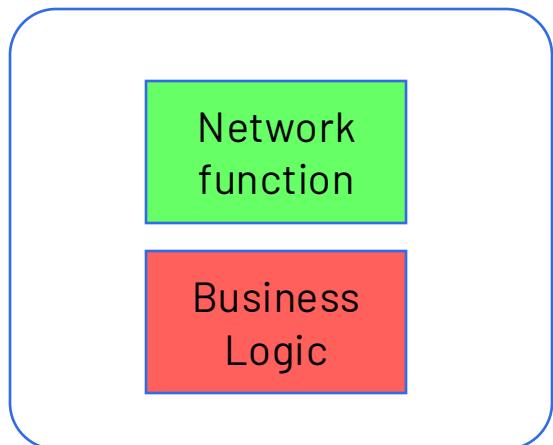
- **What are service meshes**
- **Purpose of service meshes**
- **Explore popular service mesh tools**



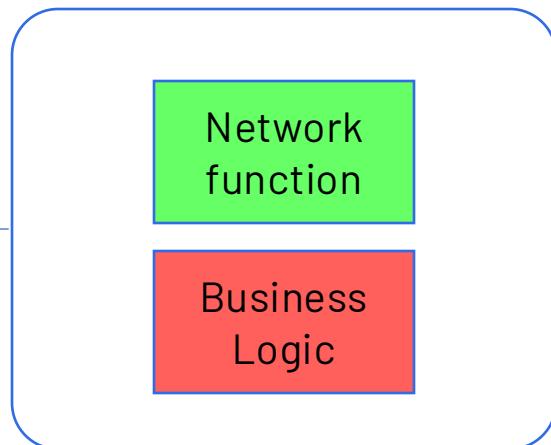
Service Meshes

Service Meshes

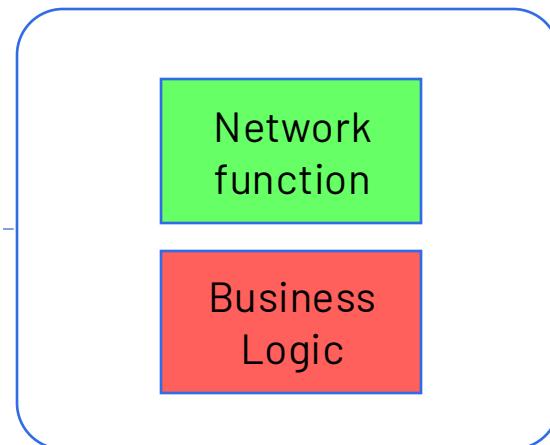
Microservice - A



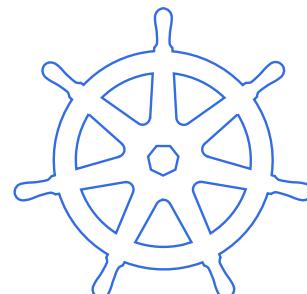
Microservice - B



Microservice - C



Service-to-service communication



Service Meshes

Service Mesh

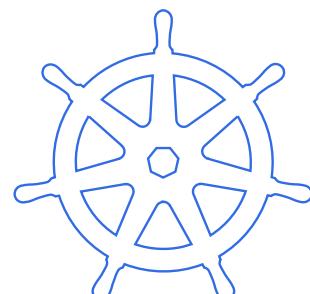
Mechanism for managing inter-service communication
that is routed through proxies

Encryption

Mutual TLS

Load Balancing

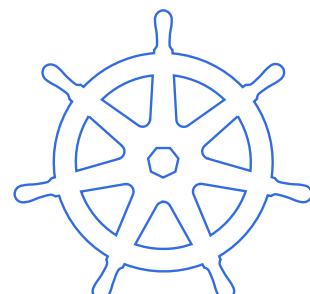
Decouples the network logic from the application or business logic of each microservice



Service Meshes

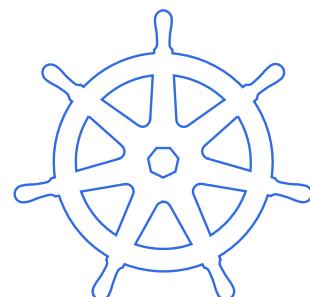
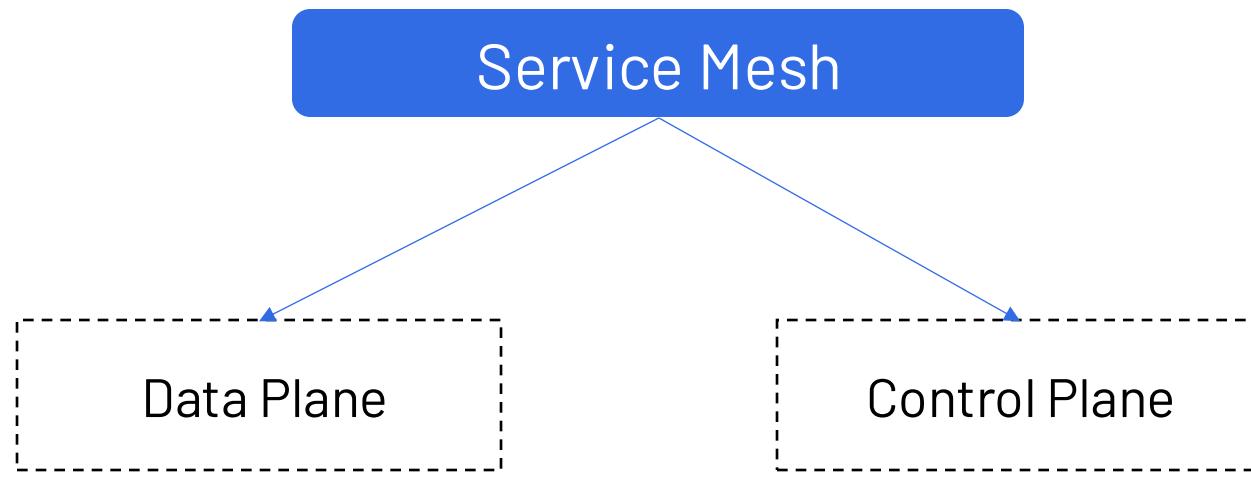
Service Mesh

Acts like traffic controller for all communication between services

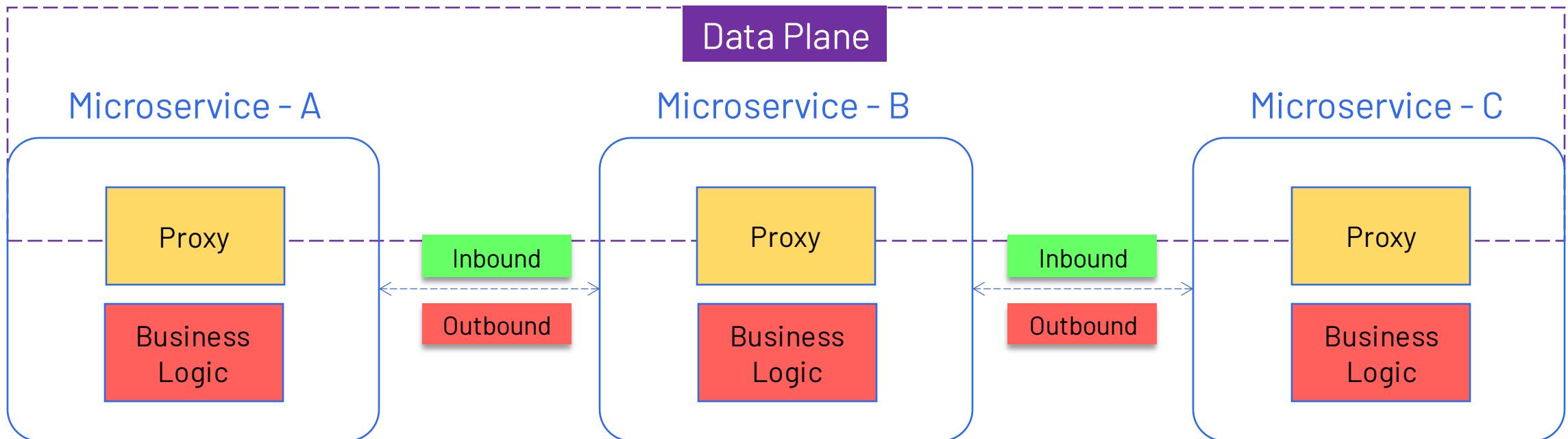


Service Mesh Components: Data Plane & Control Plane

Service Mesh Components



Data Plane



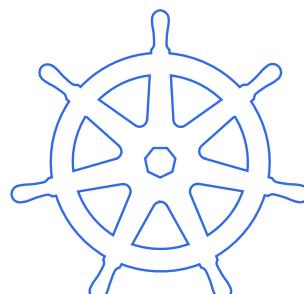
Handle functions

Service discovery

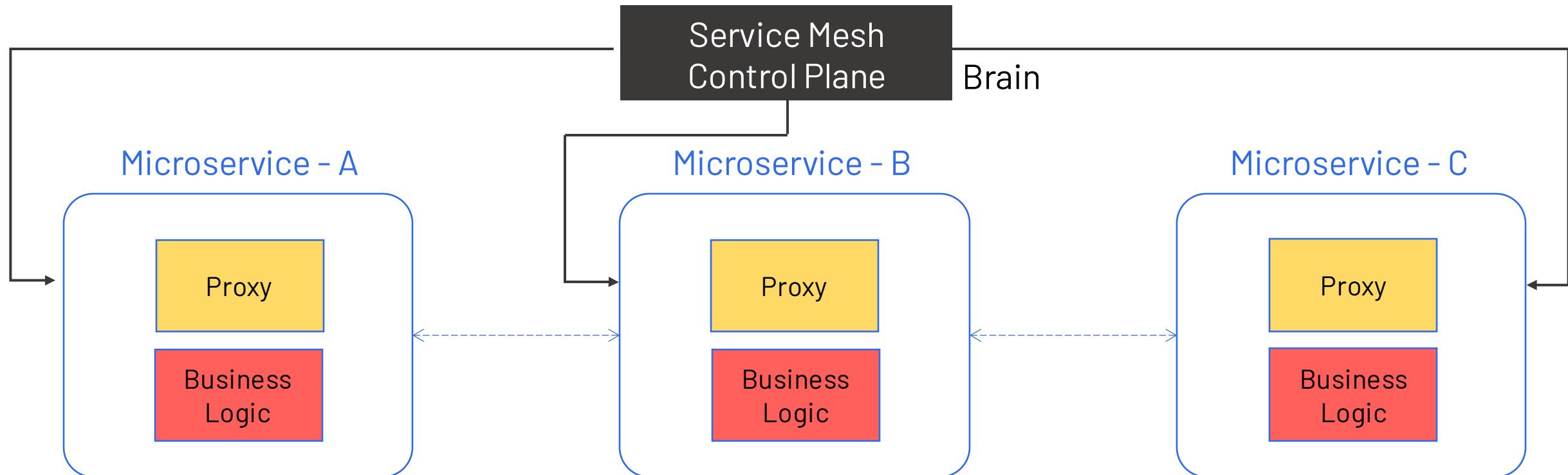
Load balancing

Security

Reliability



Control Plane

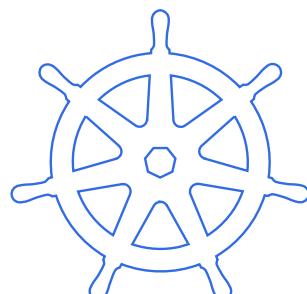


Configuration and management take place

Traffic
routing

Security
policies

Observability



Common Service Mesh Offerings

Common Service

Istio



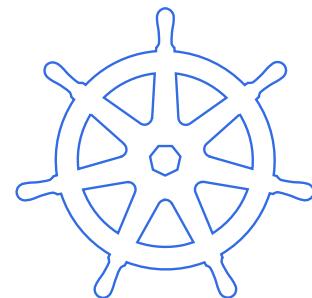
Linkerd



Consul

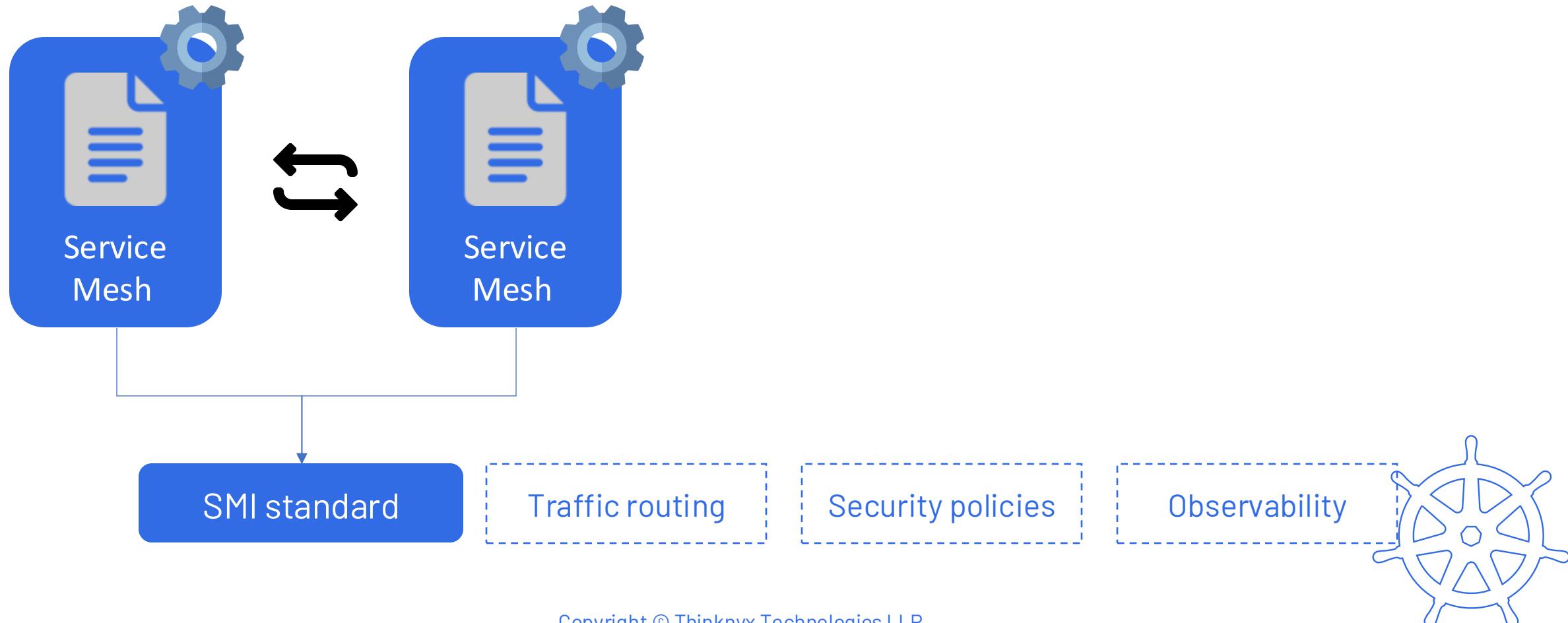


Traefik



SMI

- Service Mesh Interface
- Set of standards aimed at making service meshes more interoperable
- Provide a common set of APIs that different service meshes can support



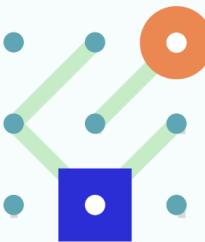


<https://smi-spec.io>

Service Mesh Interface

[Specification](#) [Ecosystem](#) [Blog](#) [Slack](#)

This project is [archived](#). [Learn more](#)

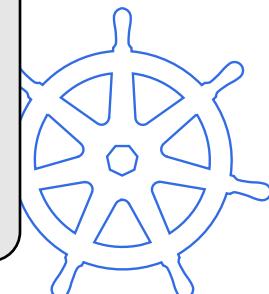


A standard interface for service meshes on Kubernetes.

Service Mesh Interface provides:

- A standard interface for service meshes on Kubernetes
- A basic feature set for the most common
- Flexibility to support new service mesh capabilities over time
- Space for the ecosystem to innovate with

[View the Spec](#)

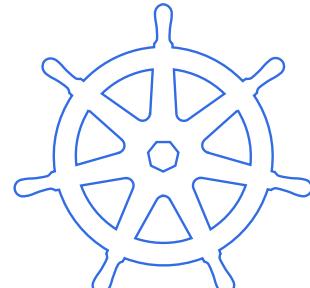


Section: 10

Section Overview

Helm Charts

- ↳ **Helm**
- ↳ **Helm Charts**
- ↳ **Developing Helm Charts**
- ↳ **Artifact Hub**

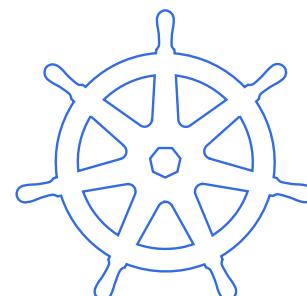


What is Helm?

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



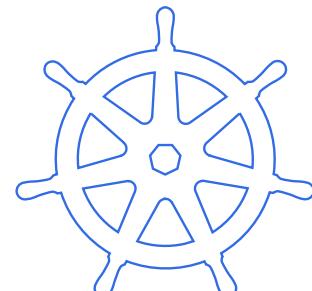
Reusability &
Consistency



Community
Collaboration



Customization &
Flexibility



Key Terminologies

Helm Charts

Helm Repositories

values.yaml

Hooks

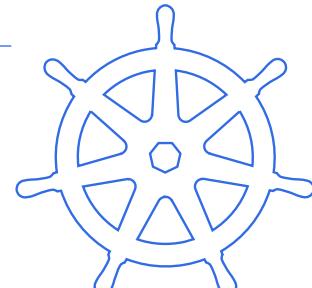


Releases

Template Rendering

Chart.yaml

Helm Plugins



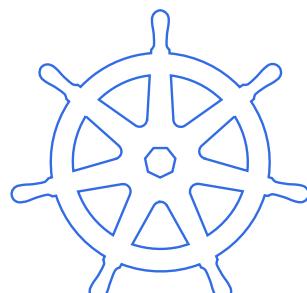
Installing Helm

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

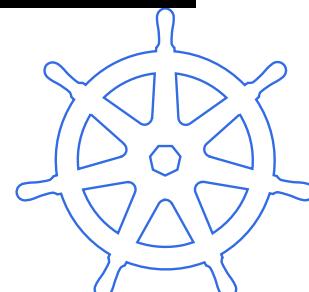
```
$ helm package my-chart
```

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

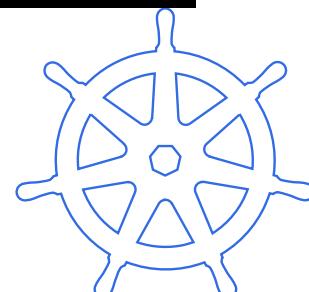
```
$ helm upgrade my-release my-chart
```

helm uninstall

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

helm repo

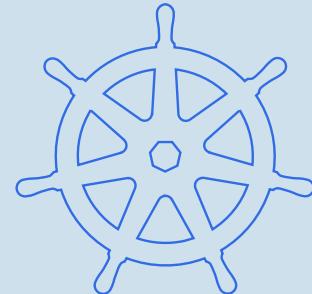
```
$ helm repo add stable https://charts.helm.sh/stable
```





Demo

Helm Charts from
Artifact Hub

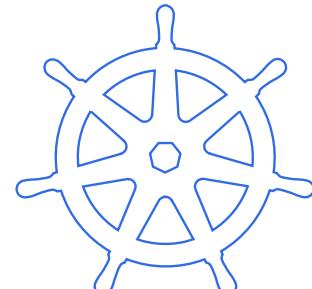


Section: 11

Cloud Native Architecture

Section Overview

- ↳ Cloud Native Architecture & it's components



Cloud Native

Cloud Native

Modern approach to building and running software applications that are designed to fully leverage the capabilities and benefits of cloud computing

Dynamic Scalability

Distributed Infrastructure



Speed



Agility



Scalability



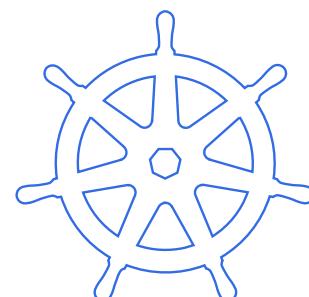
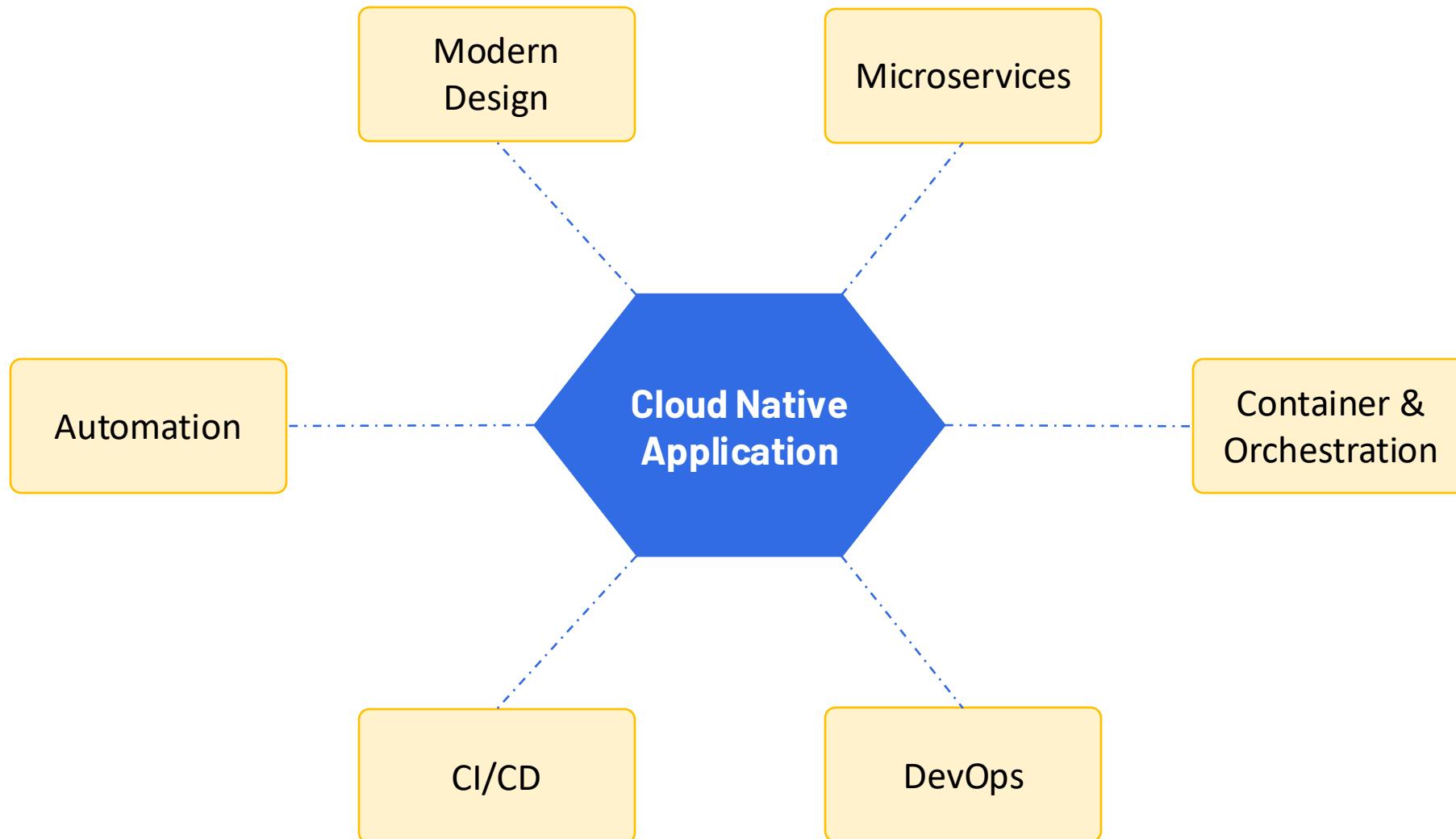
Reliability



Cost-effectiveness



Cloud Native Principles



Modern Design

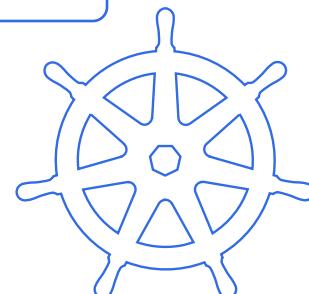
Modern Design

Twelve Factor Application

- Codebase
- Config
- Build, Release, Run
- Port Binding
- Disposability
- Logs



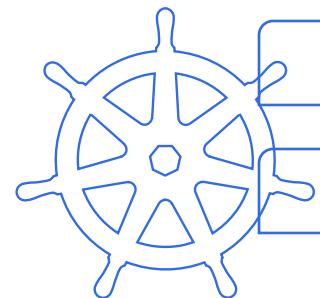
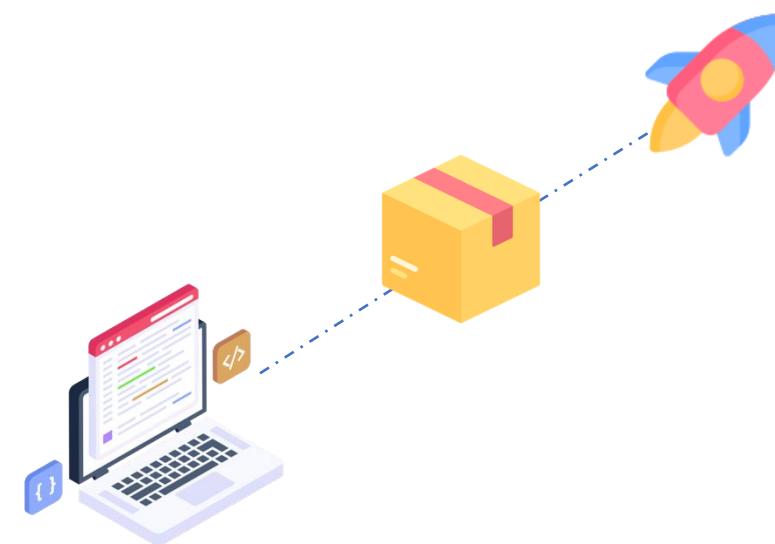
- Dependencies
- Backing Service
- Processes
- Concurrency
- Dev/Prod Parity
- Admin Processes



Modern Design

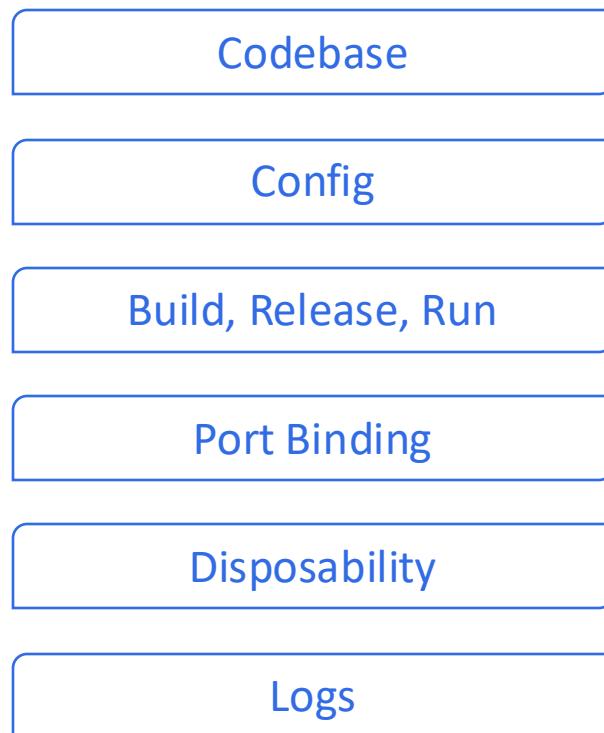
Twelve Factor Application

Build, Release, Run

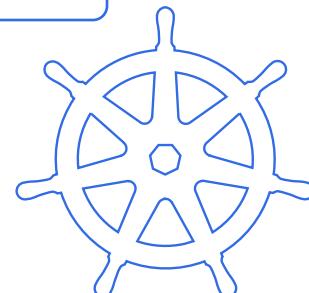
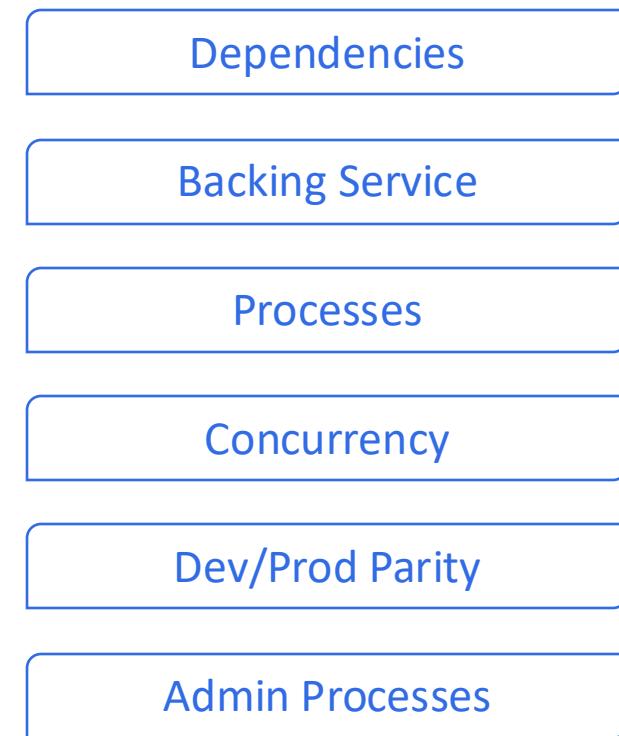


Modern Design

Twelve Factor Application



Q https://12factor.net

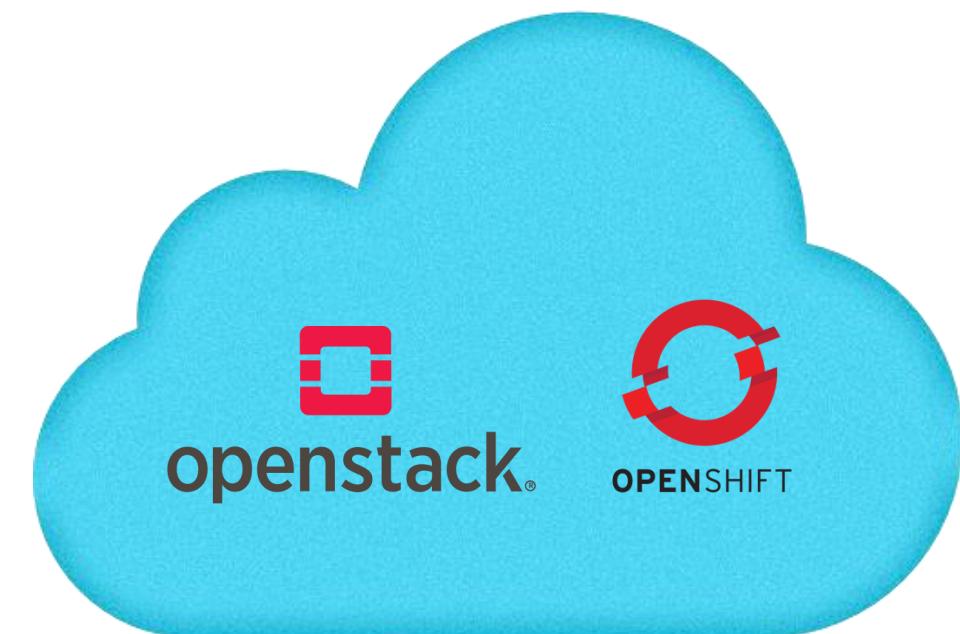


Modern Design

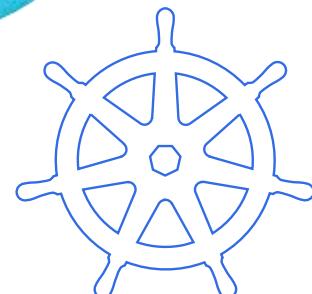
Cloud providers offer well-architected frameworks that supply industry-standard best practices



Public Cloud



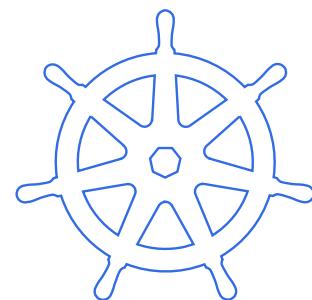
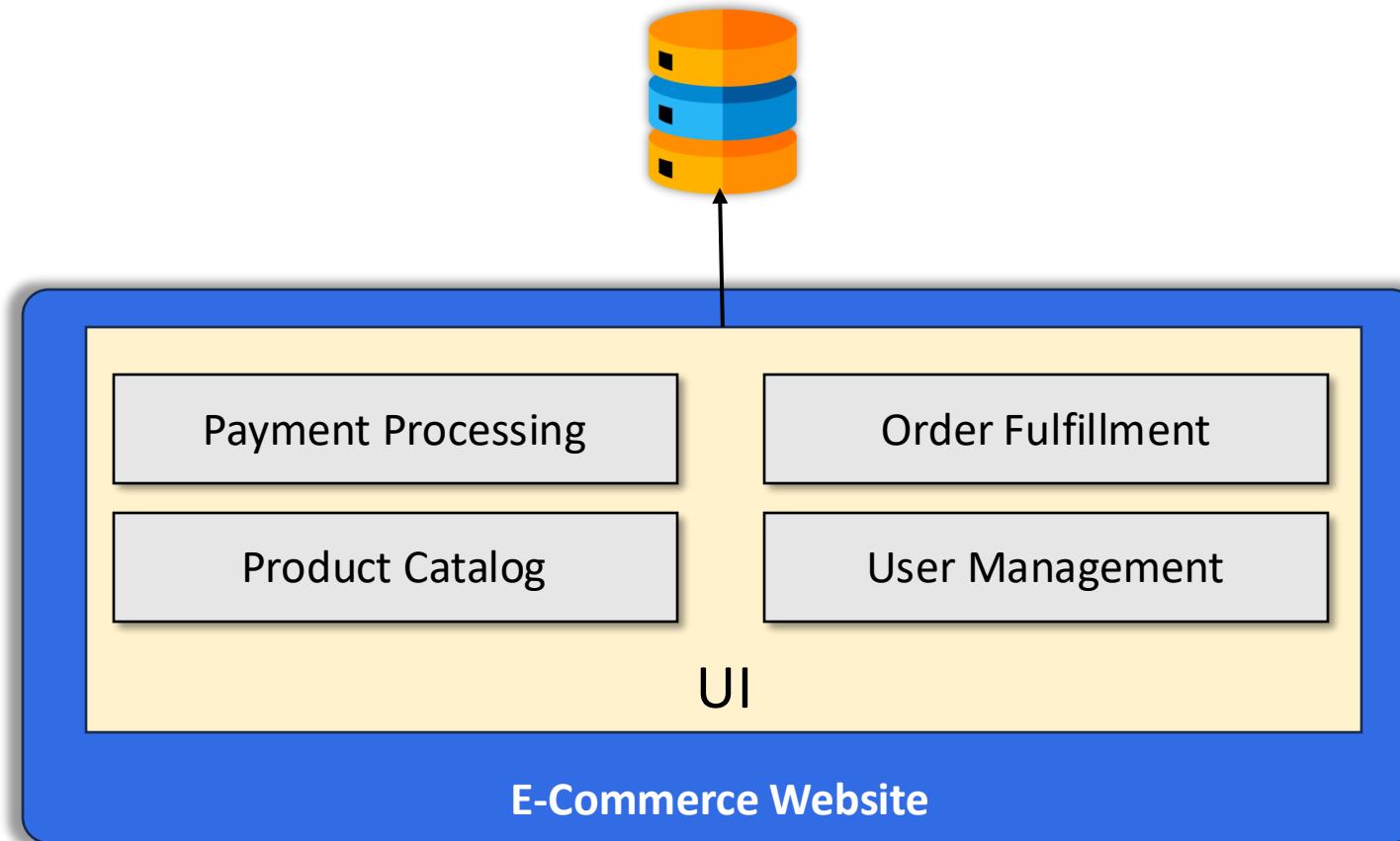
Private Cloud



Monolithic & Microservices

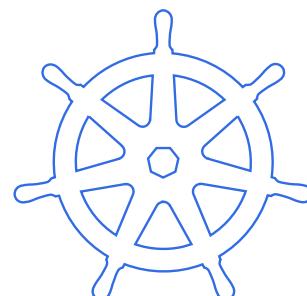
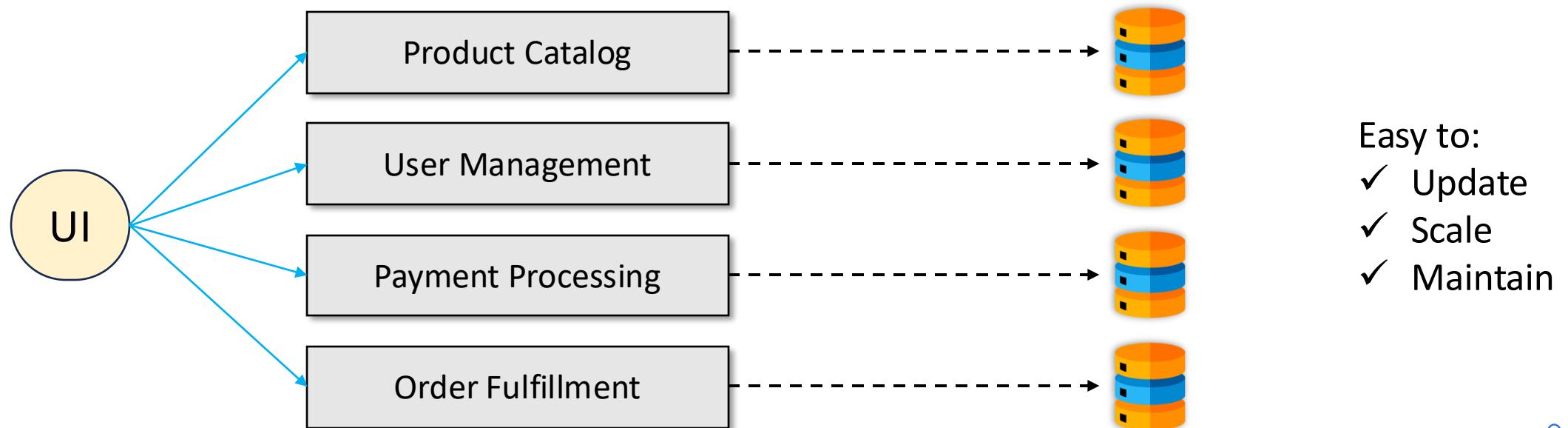
Monolithic Architecture

Single, large codebase that contains all components and functions of the application



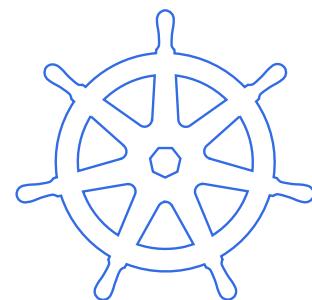
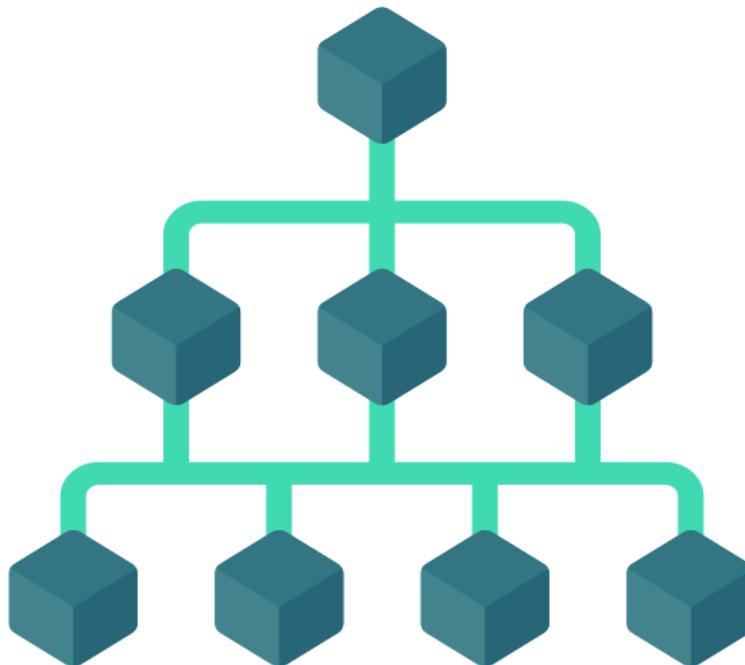
Microservice Architecture

Breaks the applications into smaller, independent services, each with it's own codebase and responsibilities including managing it's own database



Microservices

Minimum Valuable Product



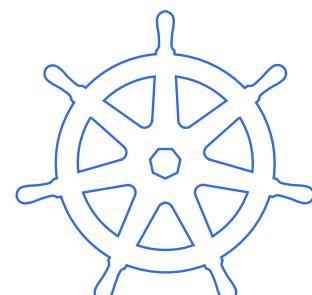
Containers & Orchestration

Containers & Orchestration



Containerization

- Plays a pivotal role in the cloud-native ecosystem by simplifying deployment and management, making operations more efficient
- Lightweight, self-contained units that encapsulate an application, including its source code and dependencies
- Embody the "**build anywhere, run anywhere**" philosophy

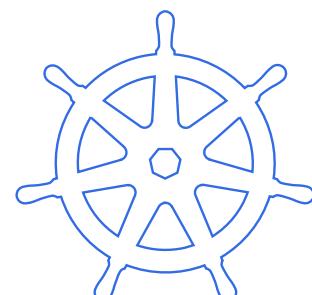


Containers & Orchestration



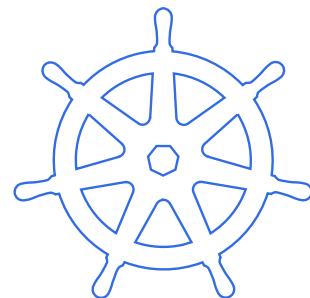
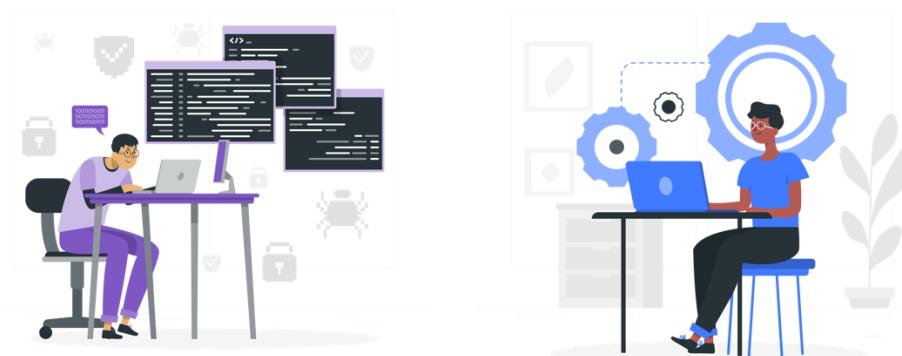
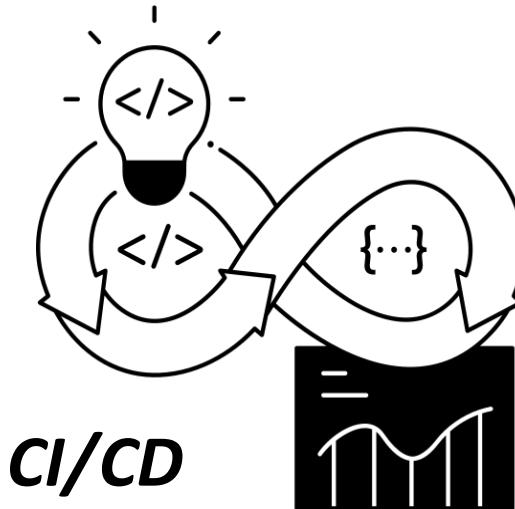
Container Orchestration

- Kubernetes act as orchestrators, ensuring smooth container operations
- Offer centralized control, automated recovery & load balancing
- Kubernetes facilitates **horizontal & vertical scaling**, dynamically adapting to workloads

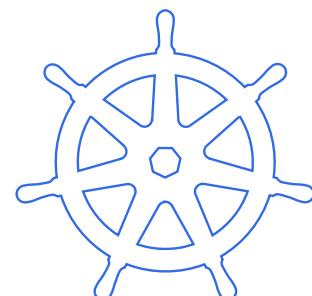
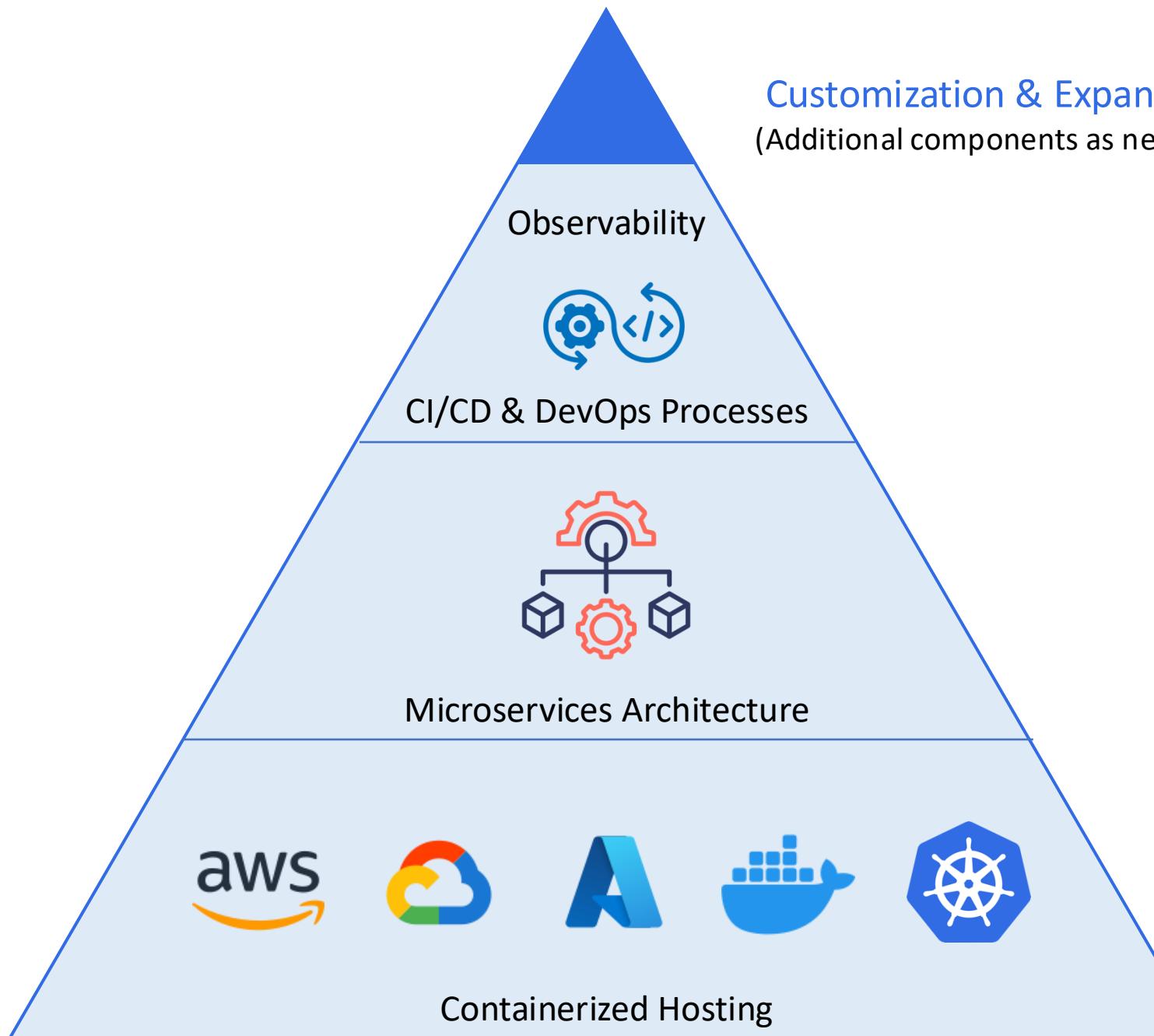


DevOps

DevOps



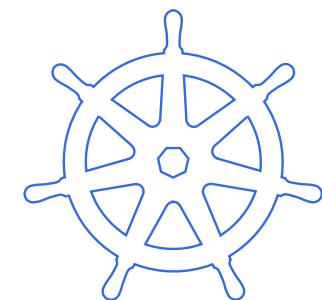
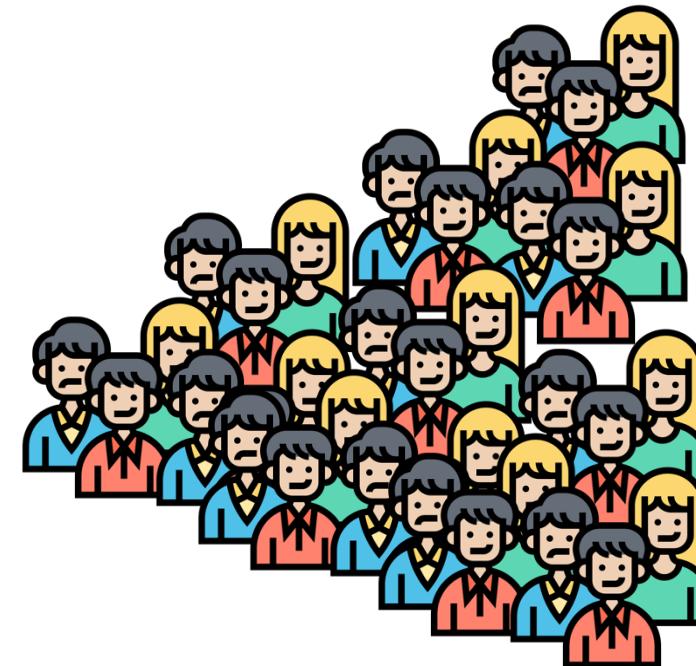
Cloud Native Technologies Pyramid



Autoscaling

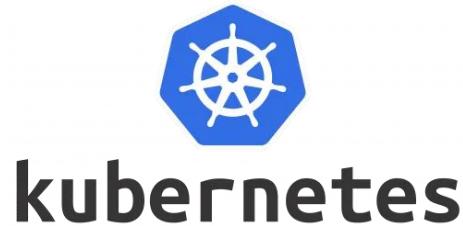


Autoscaling

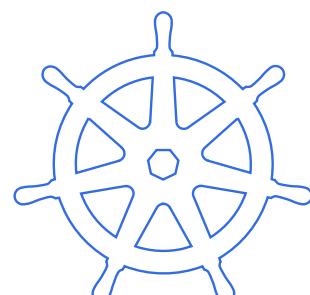


Autoscaling

System adjusts its power usage
depending on how busy it is



Adjusting the number of resources
pods or nodes available to handle
varying workloads



Autoscaling

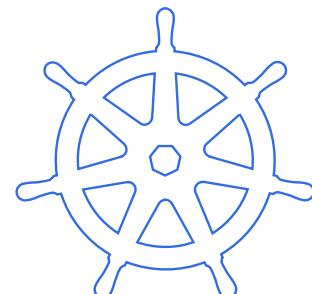


kubernetes

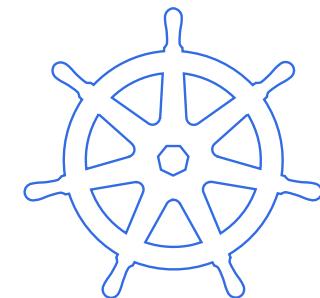
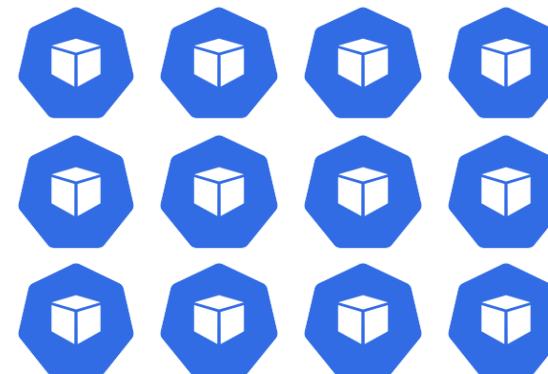
Horizontal Pod Autoscaler (HPA)

Vertical Pod Autoscaler (VPA)

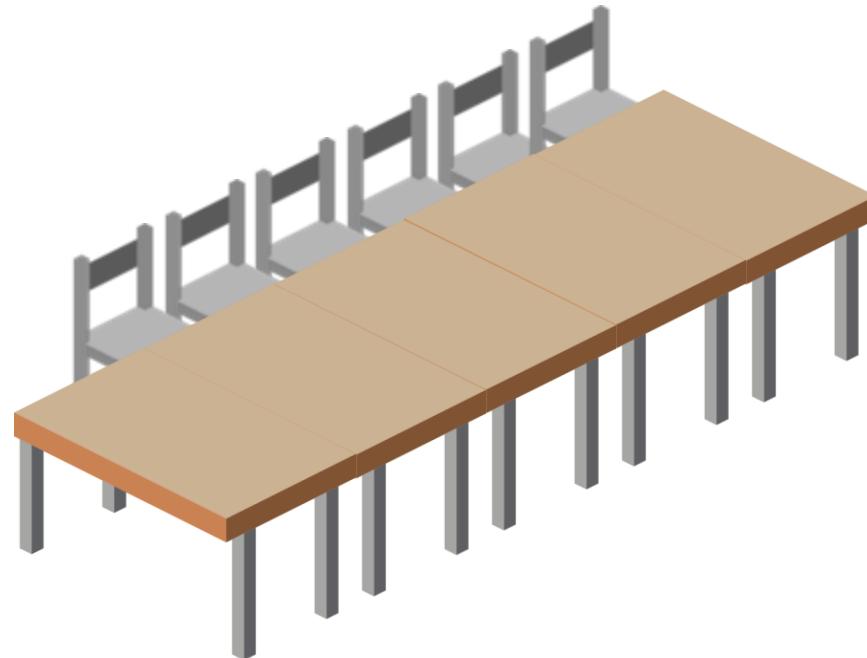
Cluster Autoscaler



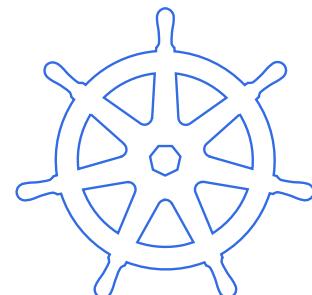
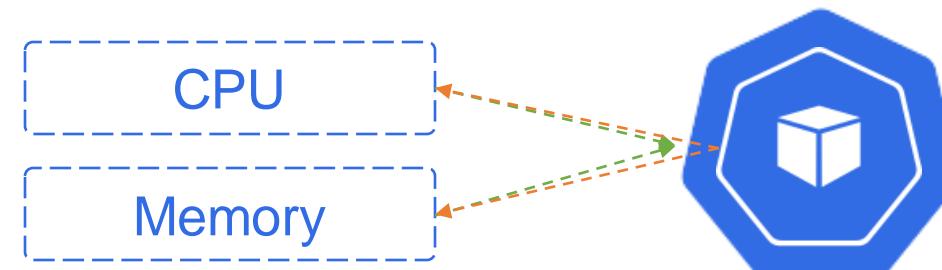
Horizontal Pod Autoscaler (HPA)



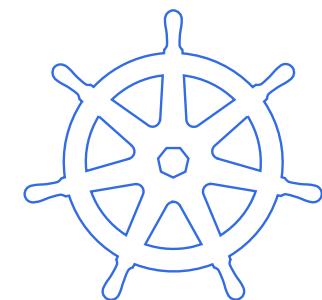
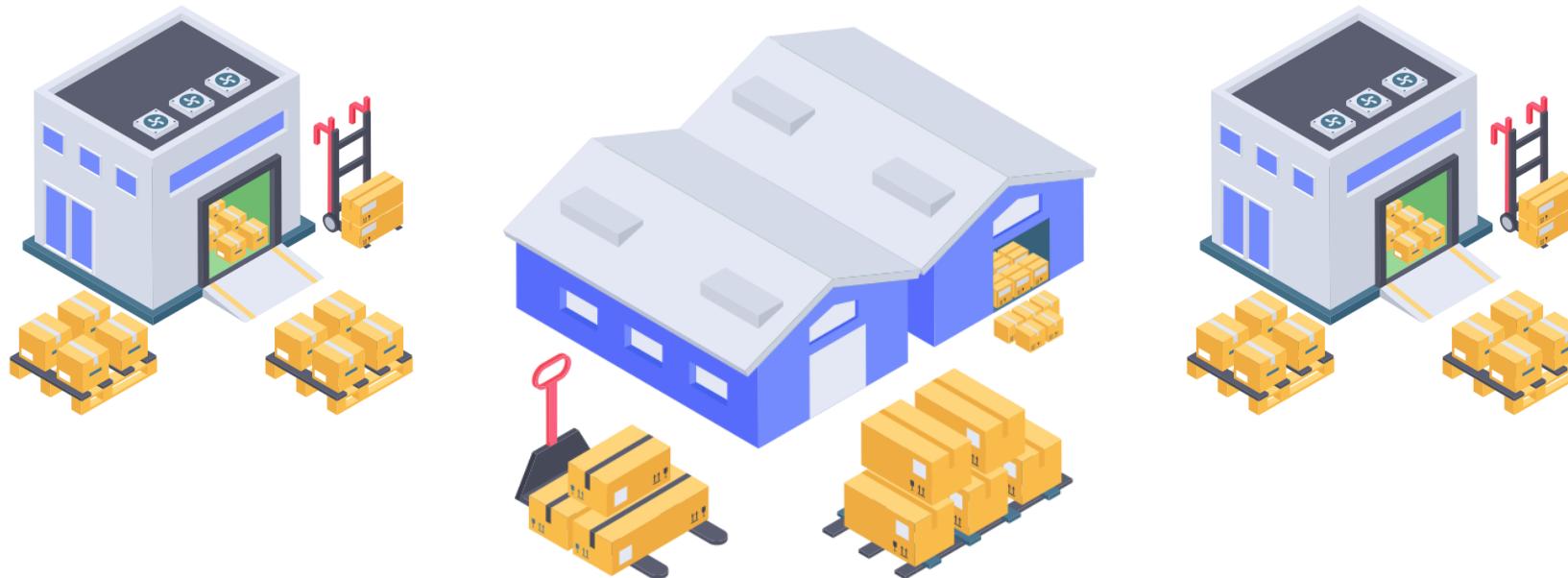
Vertical Pod Autoscaler (VPA)



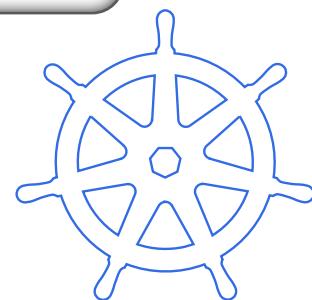
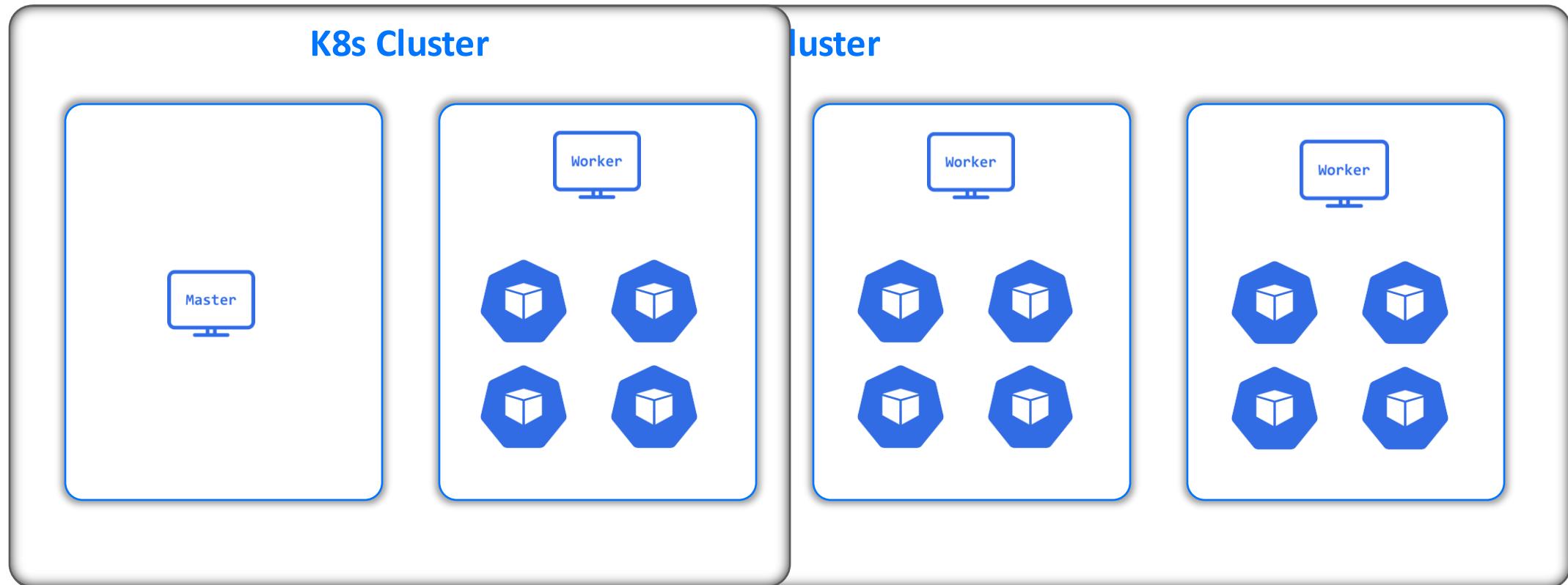
VPA



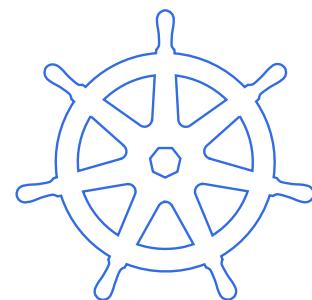
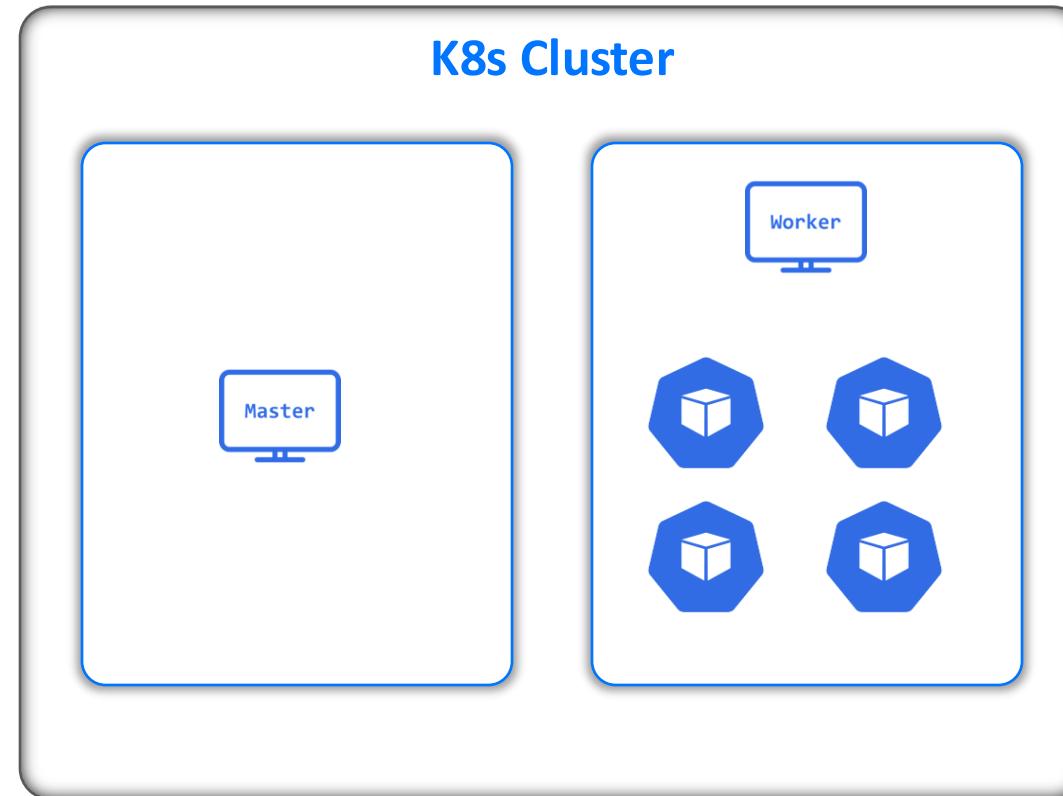
Cluster Autoscaler



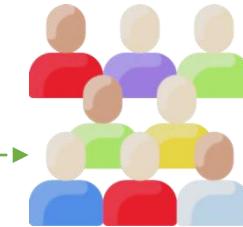
Cluster Autoscaler



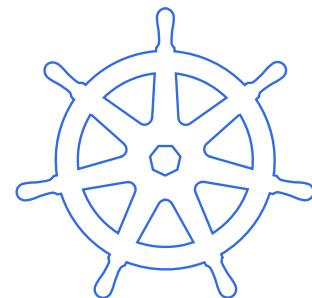
Cluster Autoscaler



Serverless

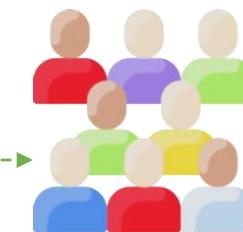


100,000 Users

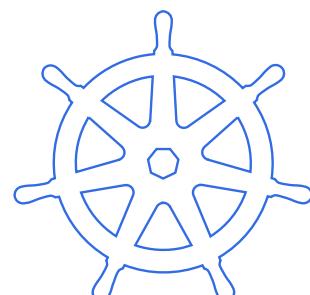


Serverless Computing

Function-as-a-Service (FaaS)



100,000 Users



Serverless Computing

Function-as-a-Service (FaaS)



Generating
PDFs



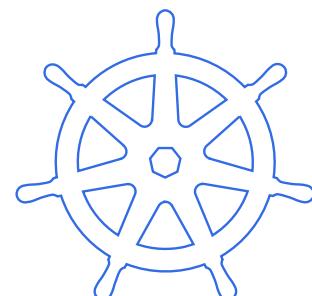
Sending
Messages



Uploading
Images

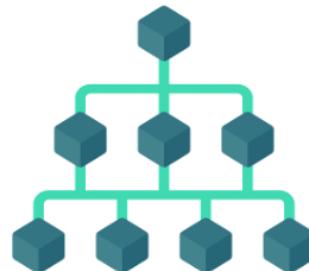


Other
Operations



Serverless Computing

Function-as-a-Service (FaaS)



Microservice
Architecture



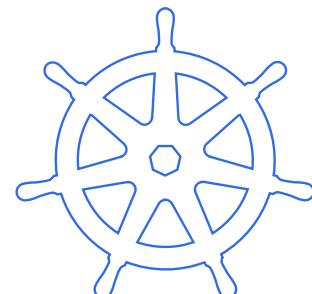
Resource
Utilization



Automatic
Scaling



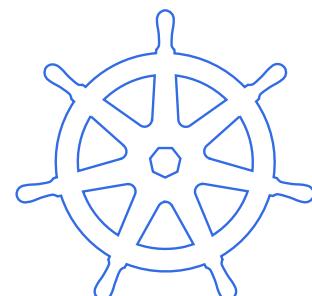
Cost
Effective



Serverless Computing

Function-as-a-Service (FaaS)

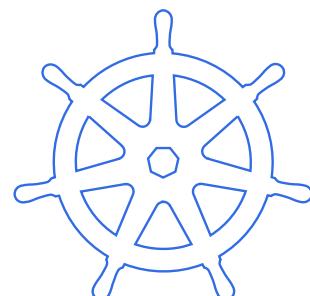
Allows you to run code in response to events without managing servers, offering scalability & efficiency for non-continuous tasks



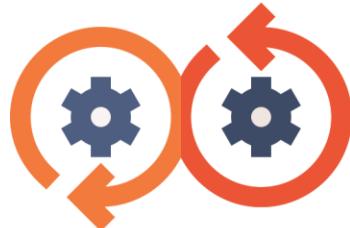
Roles and Personas

Roles and Personas

Defines the responsibilities and skillsets of professionals
working in Cloud Native environments



Roles and Personas



DevOps

Collaboration between development and operations teams to streamline the application delivery pipeline



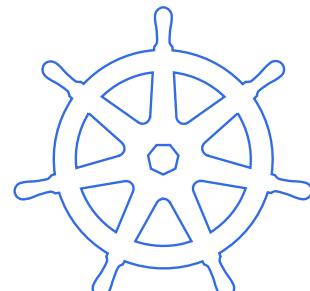
Communication



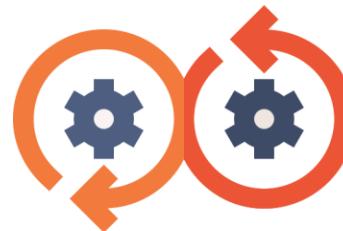
Integration



Automation



Roles and Personas



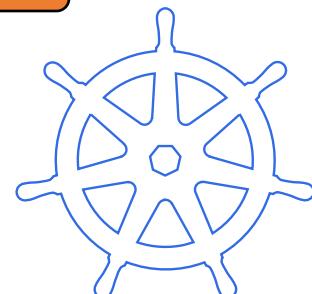
DevOps

Continuous
Integration

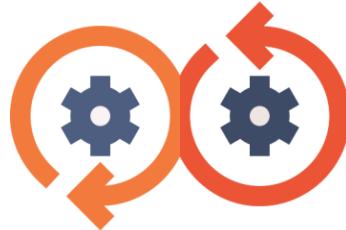
Continuous
Delivery

Continuous
Deployment

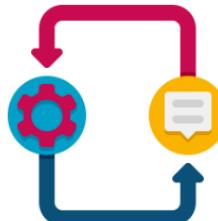
Automated
Testing



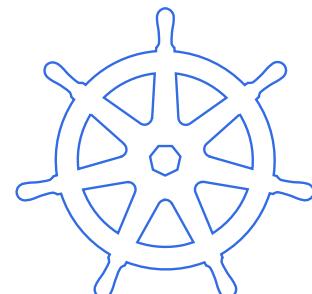
Roles and Personas



DevOps



Quick Feedback
Loops



Roles and Personas



Platform
Engineering

Focused on designing, building, and maintaining the platforms and infrastructure that support software development and deployment at scale



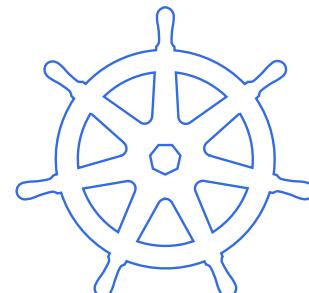
Reliable



Scalable



Efficient



Roles and Personas



CloudOps

Management, monitoring, and optimization of cloud infrastructure and services to ensure the smooth operation of applications hosted in the cloud



Cloud Computing



IT Operations



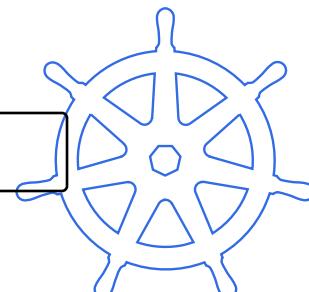
DevOps

Reliable

Scalable

Secure

Performant

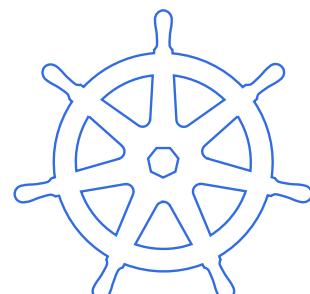


Roles and Personas



CloudOps

Focuses on making cloud environments affordable while helping organizations use cloud technologies securely and efficiently



Roles and Personas

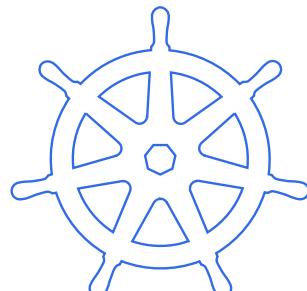


SRE

Focused on ensuring resilient, reliable applications
that are continuously available to users

Minimize
Downtime

Smooth Application
Performance



Roles and Personas



SRE

Monitoring & Observability



Prometheus

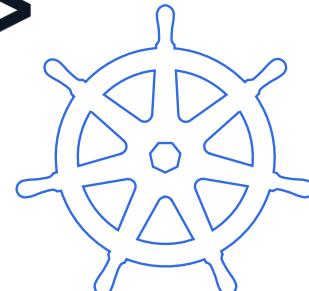


Grafana

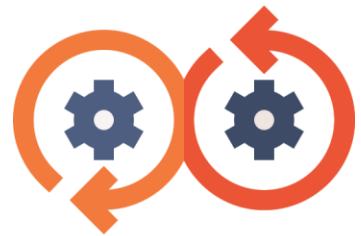


ELK Stack

splunk®>



Roles and Personas



DevOps



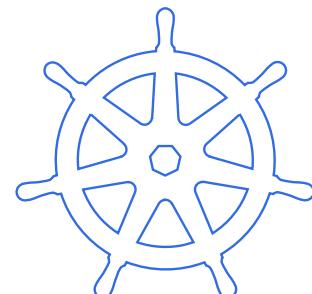
Platform Engineering



CloudOps



SRE



Open Standards

Open Standards



Cloud Native
Application

Open Standards

Shared guidelines or rules that help different software systems to work together smoothly, even if they're from different companies or vendors



Kubernetes



Container



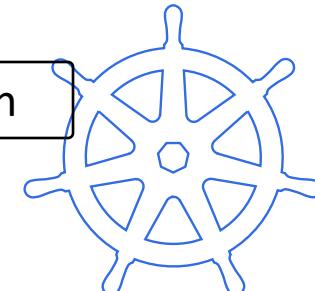
Storage solution



Networking



Service Mesh



Open Standards



Universal Plug

Allows you to use any electrical device, no matter the brand, as long as the plug matches

Open Standards

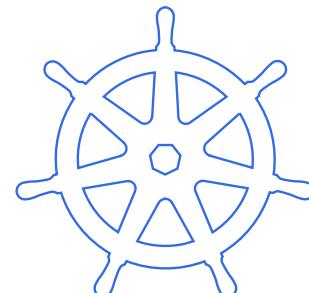
Various tools

Technologies

Systems



Move their apps between different environments and providers



OCI (Open Container Initiative)

OCI Standards

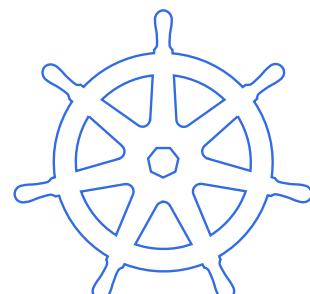
OCI (Open Container Initiative)

Creating

Building

Running containers

Making container technology more interoperable, so that containers can work across different platforms, cloud providers, or tools without problems



OCI Standards

Container Runtime
Specification

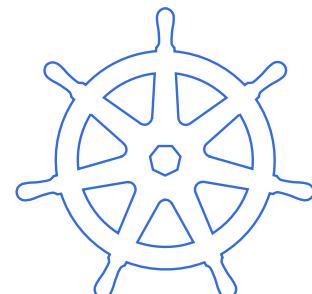
runtime-spec

Image Specification

image-spec

Image Distribution
Specification

distribution spec



OCI Standards

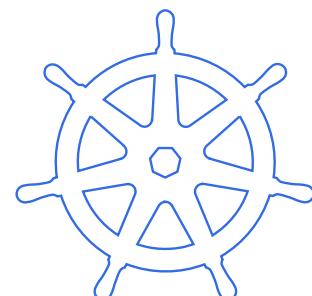
Important standards in Kubernetes

CRI(Container
Runtime Interface)

CNI(Container
Network Interface)

CSI(Container
Storage Interface)

SMI(Service Mesh
Interface)



OCI Standards

CRI (Container Runtime Interface)

runc

- Interface between Kubernetes and the container runtime
- Allows Kubernetes to manage containers in a standardized way, without depending on a specific container runtime

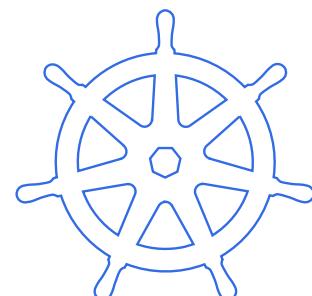
OCI Standards

CRI(Container
Runtime Interface)

CNI(Container
Network Interface)

CSI(Container
Storage Interface)

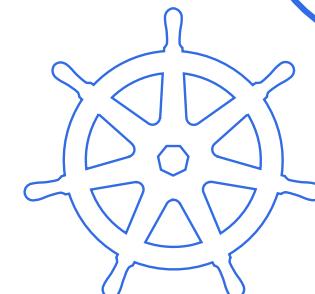
SMI(Service Mesh
Interface)



OCI Standards

- Standard that defines how containers connect to networks
- Enables Kubernetes to handle container networking in a standardized way

CNI (Container Network Interface)



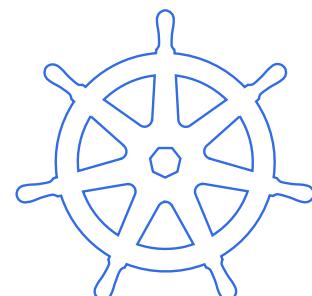
OCI Standards

CRI(Container
Runtime Interface)

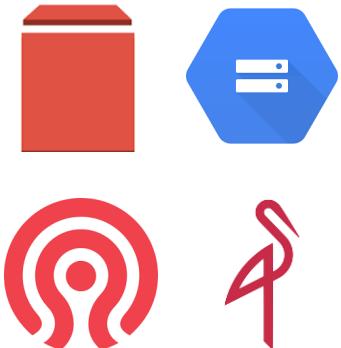
CNI(Container
Network Interface)

CSI(Container
Storage Interface)

SMI(Service Mesh
Interface)



OCI Standards



- Standard for how Kubernetes manages storage
- Switch storage providers without massive rewrites of your app or cloud infrastructure

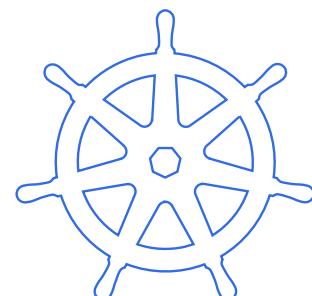
OCI Standards

CRI(Container
Runtime Interface)

CNI(Container
Network Interface)

CSI(Container
Storage Interface)

SMI(Service Mesh
Interface)



OCI Standards

- Standard way to manage microservices communications in Kubernetes
- Service meshes can work with different applications and Kubernetes setups

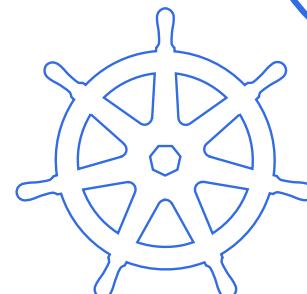
SMI (Service Mesh Interface)



Traffic management

Observability

Security



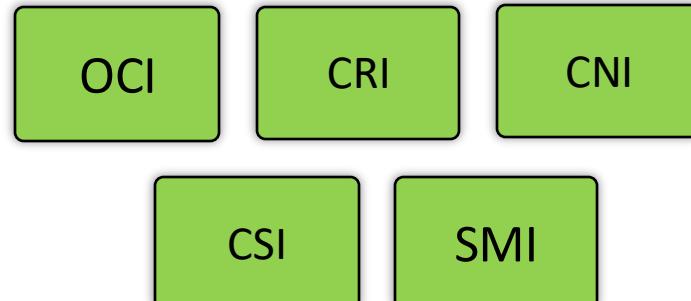
Why These Standards Matter

Easier to migrate from one tool or software to another

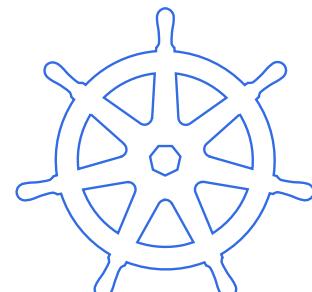


Very difficult to move to another one

Complexity and lock-in

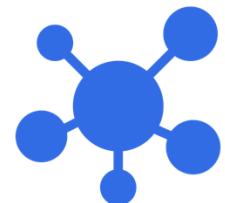


Much easier to scale

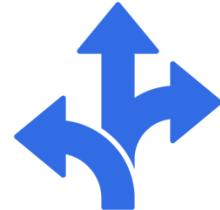


Why These Standards Matter

Standards



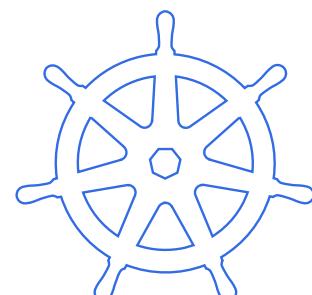
Interoperable



Flexible



Future-proof

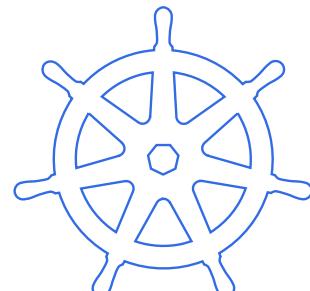


Section: 12

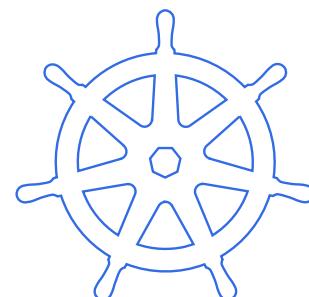
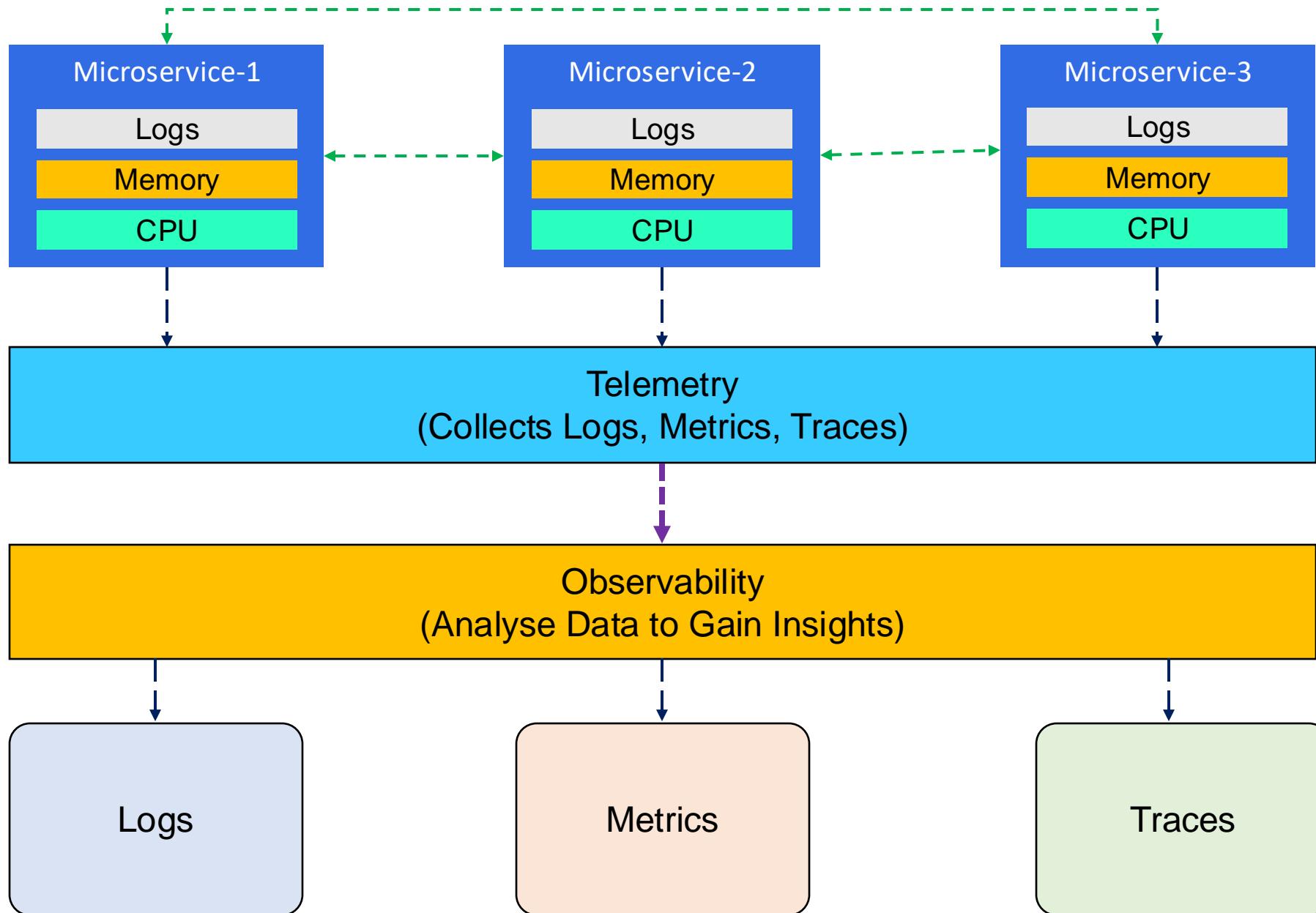
Cloud Native Observability

Section Overview

- ↳ **Telemetry**
- ↳ **Observability**
- ↳ **SLI, SLO and SLA**



Telemetry & Observability



Telemetry

Observability

Monitor

Troubleshoot

Optimize

Monitoring & Dashboarding



Prometheus



Grafana

Tracing



JAEGER

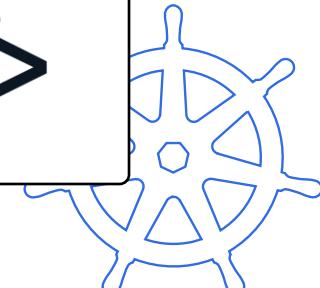


ZIPKIN

Logging

ELK

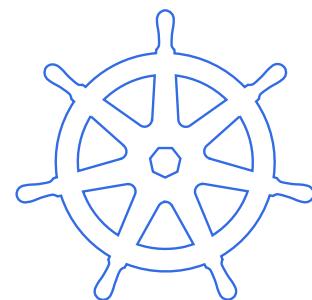
splunk[®]



Observability



DATADOG



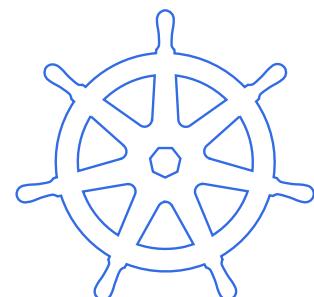
SLI, SLO and SLA

Site Reliability Engineering

SLI

SLO

SLA



SLI

Service Level Indicator

Measurement of how well a service is performing

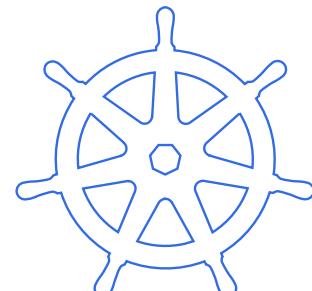
Response Time

Availability

Error Rate



%age of requests
served within 1 second

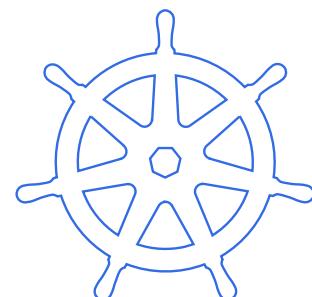


Site Reliability Engineering

SLI

SLO

SLA



SLO

Service Level Objective

Target or goal for a specific SLI

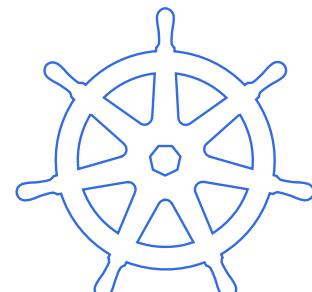
Level of service you aim to provide to users over a certain period

SLI

Response time

SLO

99% of requests
should be served
within 500ms

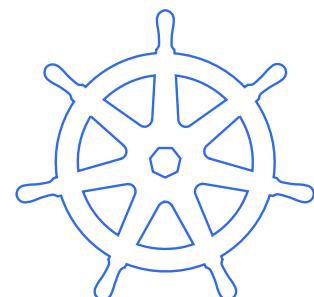


Site Reliability Engineering

SLI

SLO

SLA



SLA

Service Level Agreement

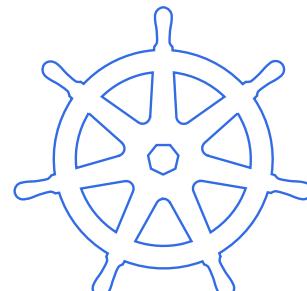
Formal contract between a service provider and a customer

If service fails to meet those SLOs, there could penalties or credits

SLA

Website's uptime

99% uptime per month



Site Reliability Engineering

SLI

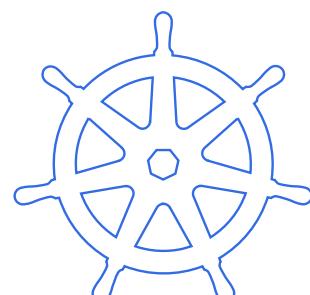
Actual measurement
of performance

SLO

Target you're
aiming for

SLA

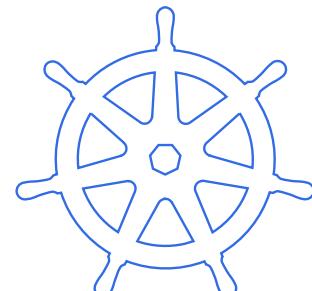
Formal promise to customers,
with consequences if the target is
missed



Monitoring with Prometheus

Lecture Overview

- † **Introduction to Prometheus**
- † **Prometheus Architecture**
- † **Prometheus Terminologies**



Prometheus Introduction

Prometheus Introduction

System monitoring and alerting toolkit



Open-Source

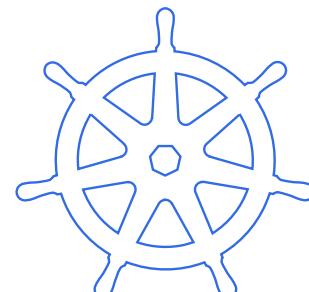


2016

2018

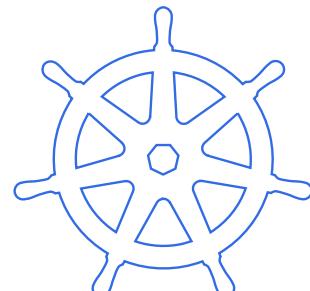
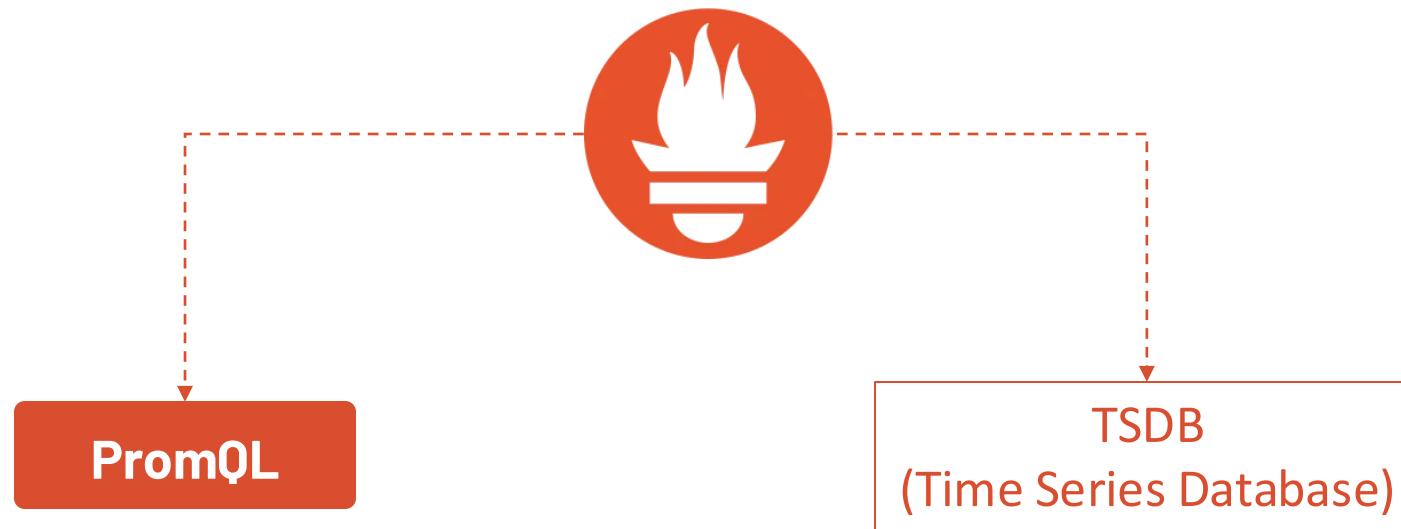


Graduated
CNCF project



Prometheus Introduction

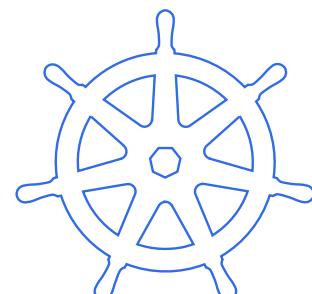
- Platform-agnostic and can be installed on various operating systems and environments
- Capability to monitor wide range of technologies, including databases, hardware, messaging systems, & Kubernetes



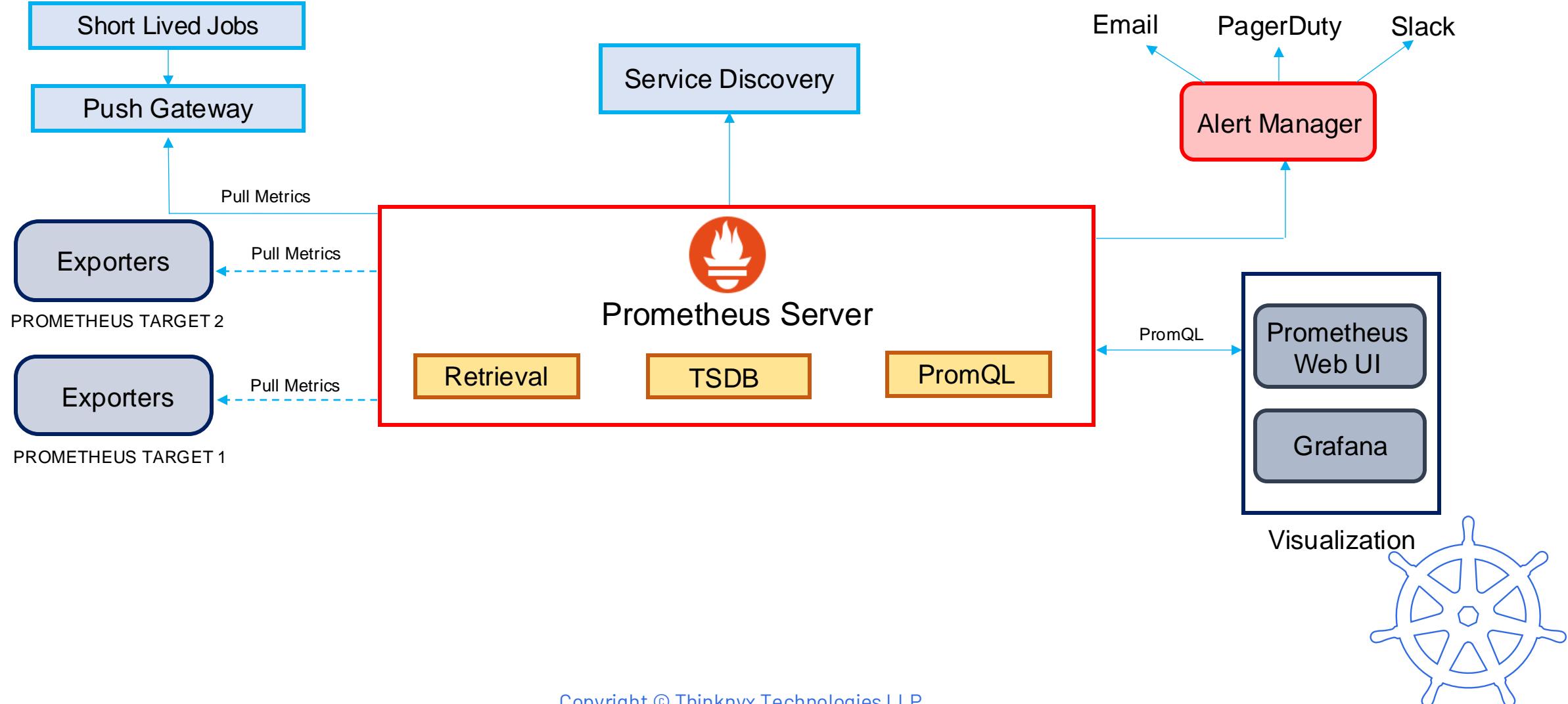
Prometheus Introduction



Prometheus is a monitoring system that is completely open source, CNCF-approved, platform-agnostic, and comes with the powerful PromQL query language and uses time series database (TSDB) for storing metrics



Prometheus Architecture



Prometheus Terminologies

Target

Job

PromQL

Functions

Alerting rules & Alerts

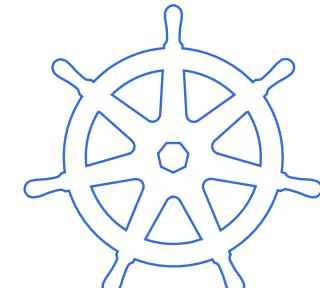
Exporter

TSDB

Data Types

Recording Rules

Client Libraries

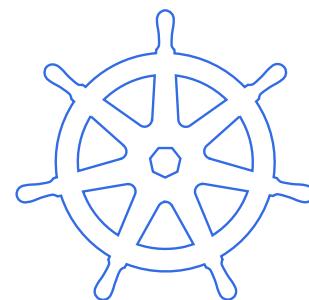
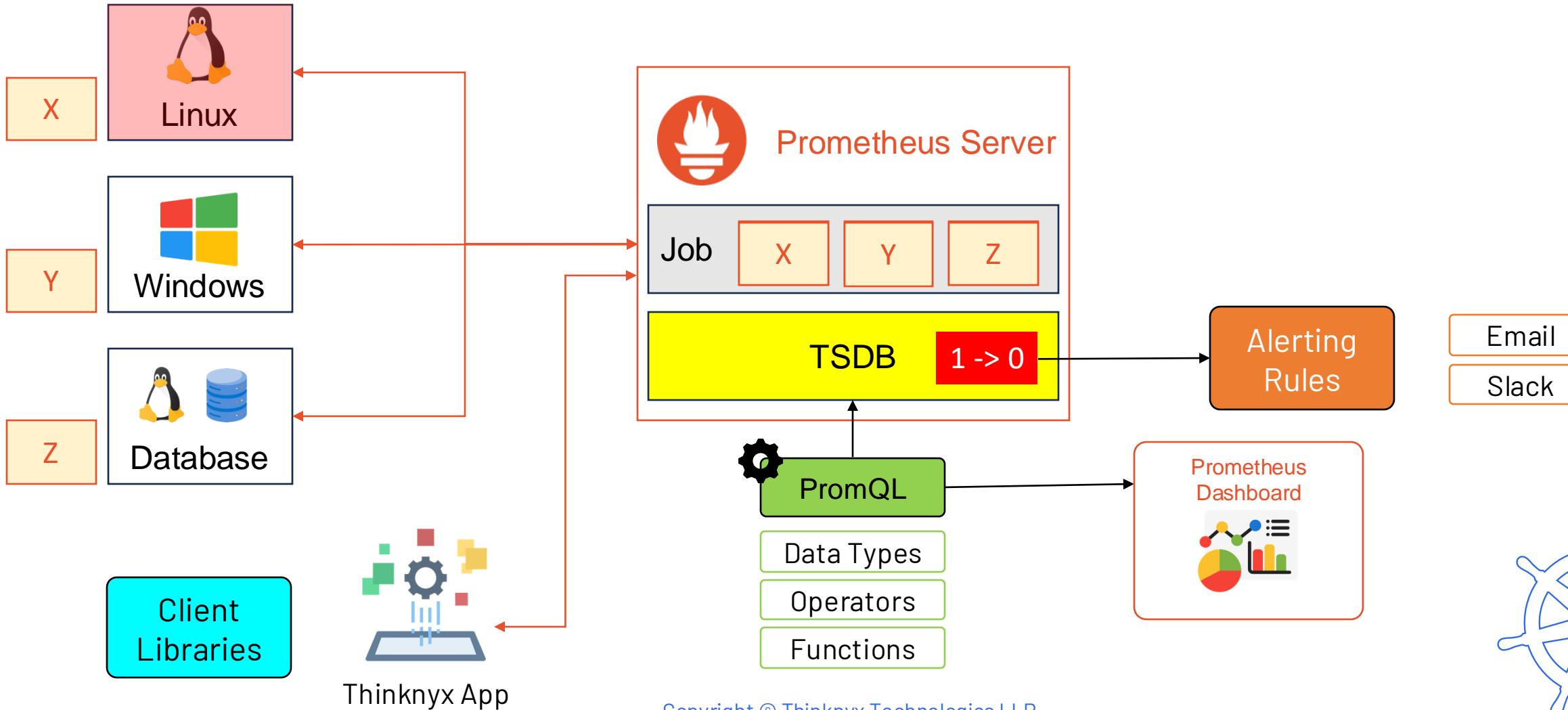


Node Exporter
Database Exporter
Network Exporter

Exporters

Total Memory
Used Memory
Free Memory

Recording Rules



Prometheus Terminologies

Target

TSDB

Recording Rules

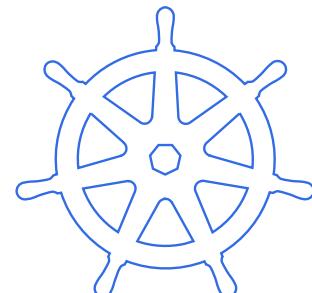
Exporter

PromQL

Alerting Rules



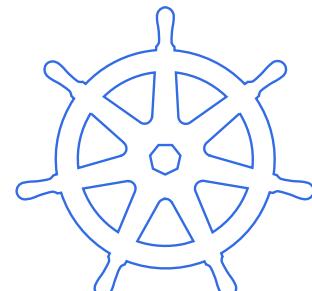
Client Libraries



Lecture Overview

Observability with Grafana

- ↳ **Grafana Introduction**
- ↳ **Grafana Architecture**
- ↳ **Grafana Terminologies**
- ↳ **Demonstration:**
 - **Grafana Installation**
 - **Checking Dashboard**
 - **Prometheus as a data source**
 - **Dashboard Creation**



Grafana Introduction

Grafana Introduction



Open-Source



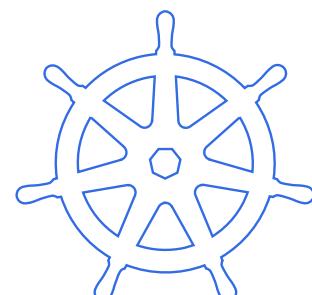
User friendly dashboard



Community-based Dashboard



Alert Management



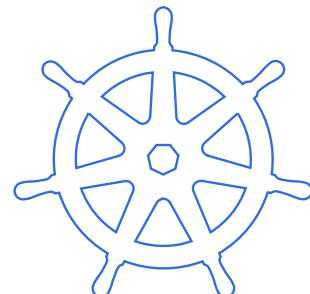
Grafana Introduction



Grafana OSS (Open
Source Software)

Grafana Enterprise

Grafana Cloud



Grafana Introduction



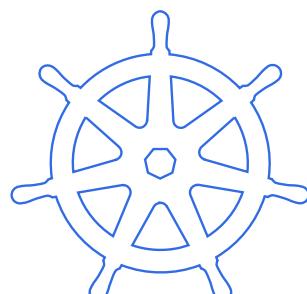
**Grafana
as a solution**

Data Analytics

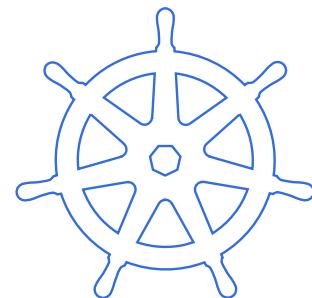
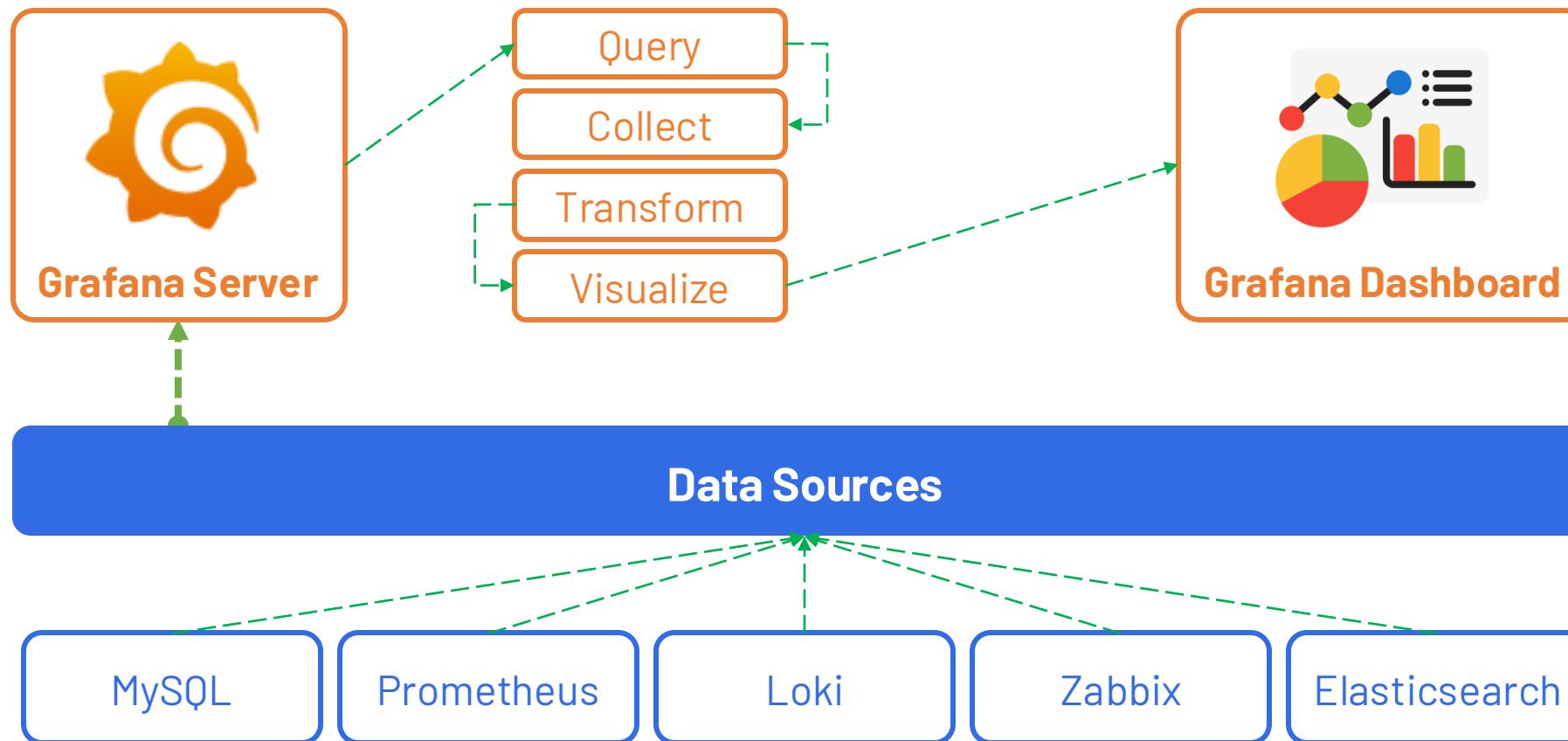
Pulling Up Metrics

Visualizing Data

Simple & User-friendly Dashboards



Grafana Architecture



Grafana Terminologies

Data Sources

Built in RBAC

Dashboards

Alerting

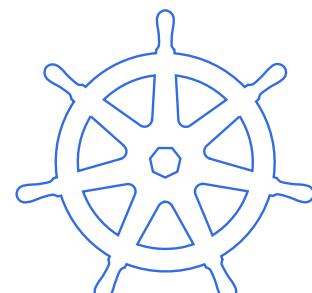
Explore

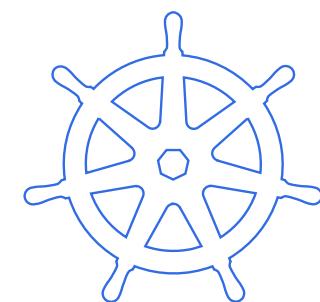
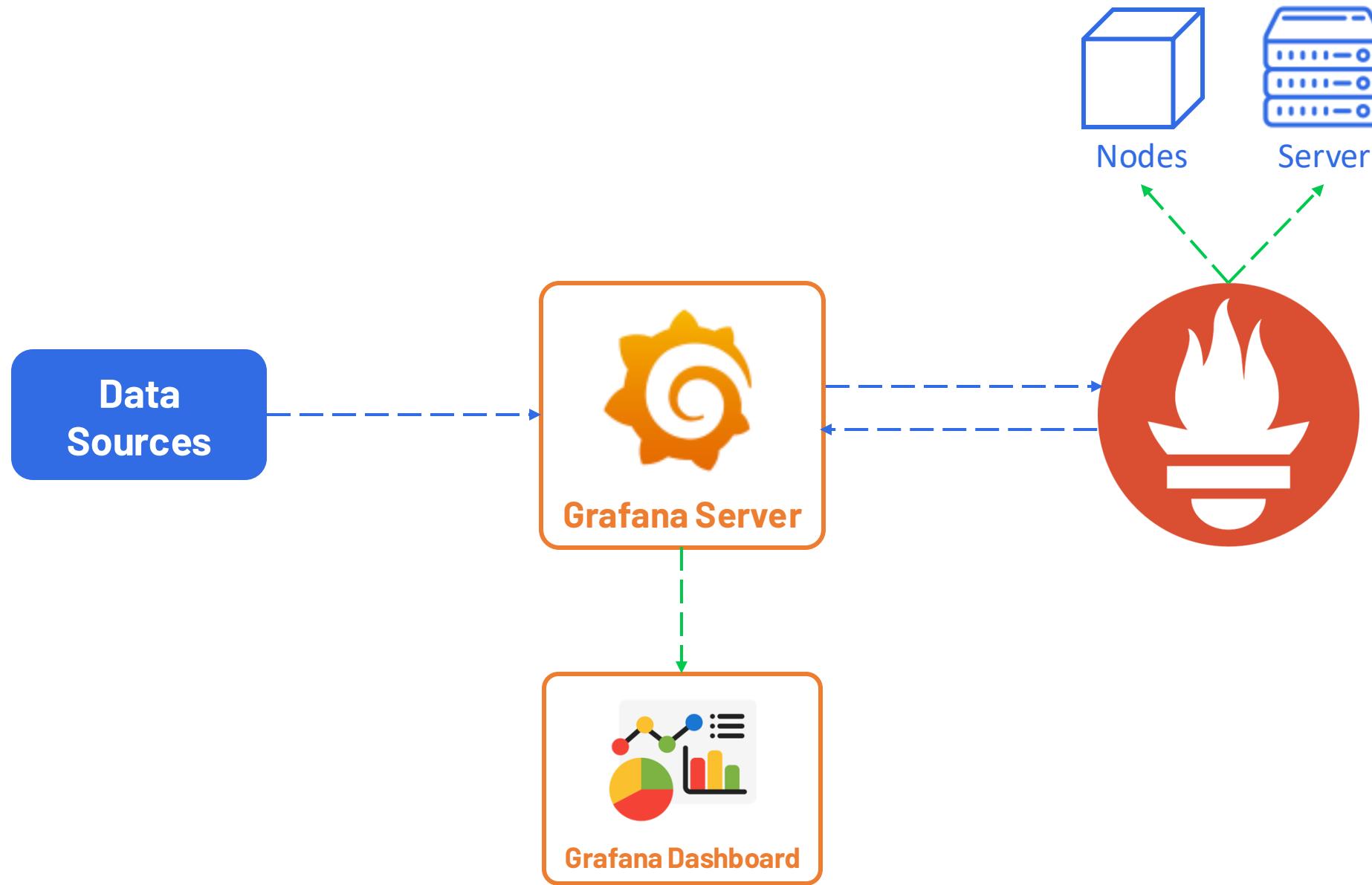
Visualization

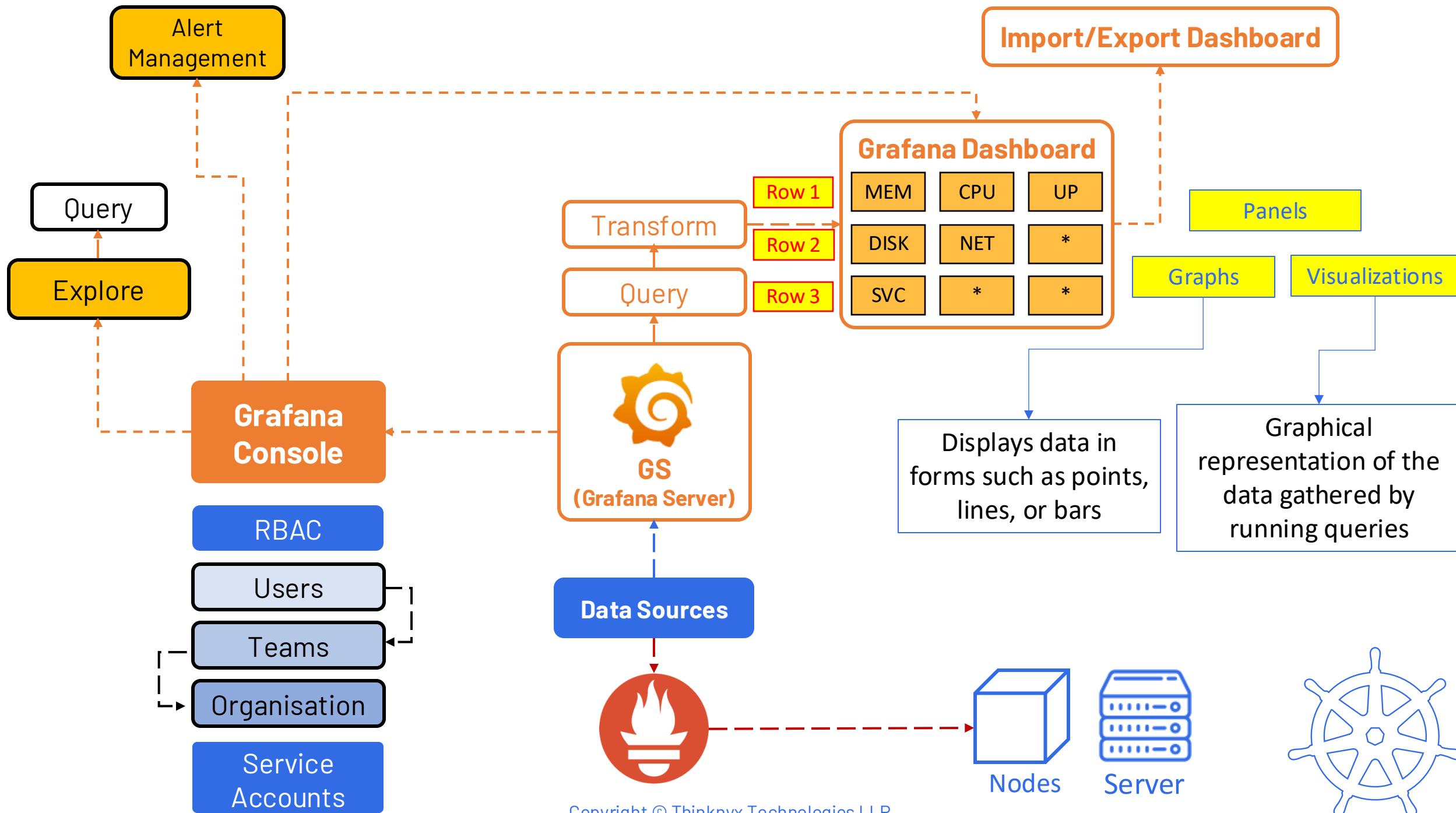
Import/Export Dashboard

Graphs

Panels







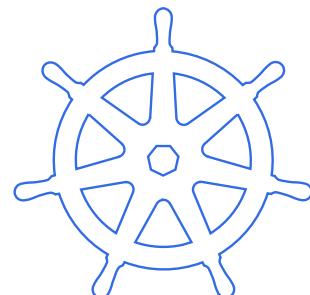
Cost Management



Cloud-native
Applications

Agility

Scalability

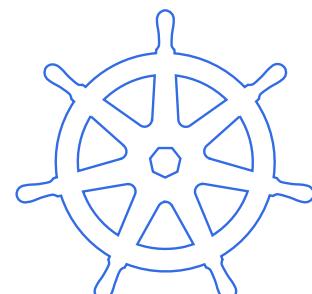


Cloud-native Cost Management

Monitor

Optimize

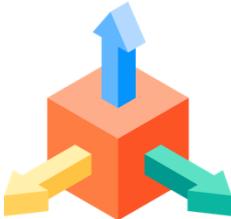
Control Cost



Cloud-native Cost Management



Rightsizing



Autoscaling



Continuous Monitoring



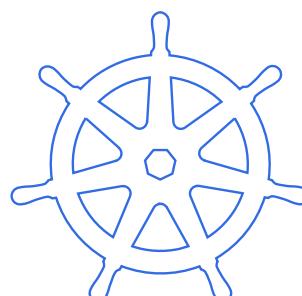
Cloud Anomaly Detection



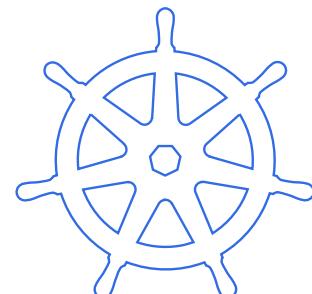
Cost Allocation & Tagging



Forecasting, Cost Alerts, & Budgeting



Cloud-native Cost Management



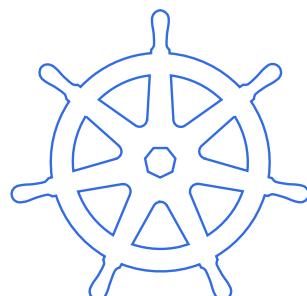
Cloud-native Cost Management

FinOps

Cost
Optimization

Cloud Governance &
Policies

Budgeting &
Forecasting

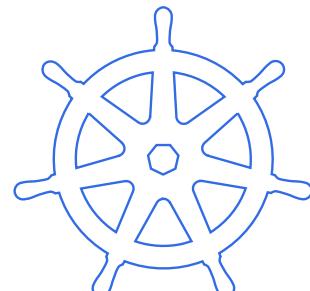


Section: 13

Cloud Native Application Delivery

Section Overview

- ↳ **Cloud-native application delivery**
- ↳ **GitOps**
- ↳ **Argo CD**



Cloud Native Application Delivery

Cloud Native Application Delivery

The practice of delivering and managing applications in cloud-based environments

Scalability

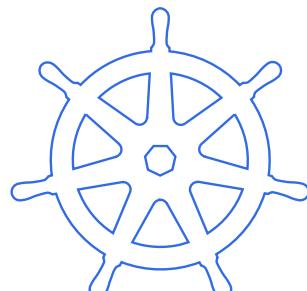
Resilience

Automation

Microservice
Architecture

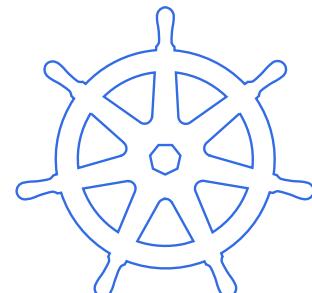
Containerization

CI/CD



Cloud Native Application Delivery

Application Delivery Fundamentals are the core concepts and best practices used to build and deliver cloud-native applications



Introduction to GitOps & Argo CD



Introduction to GitOps & Argo CD



Section Overview

- Introduction to GitOps
- Argo Project Ecosystem
- Introduction to Argo CD
- Argo CD Documentation

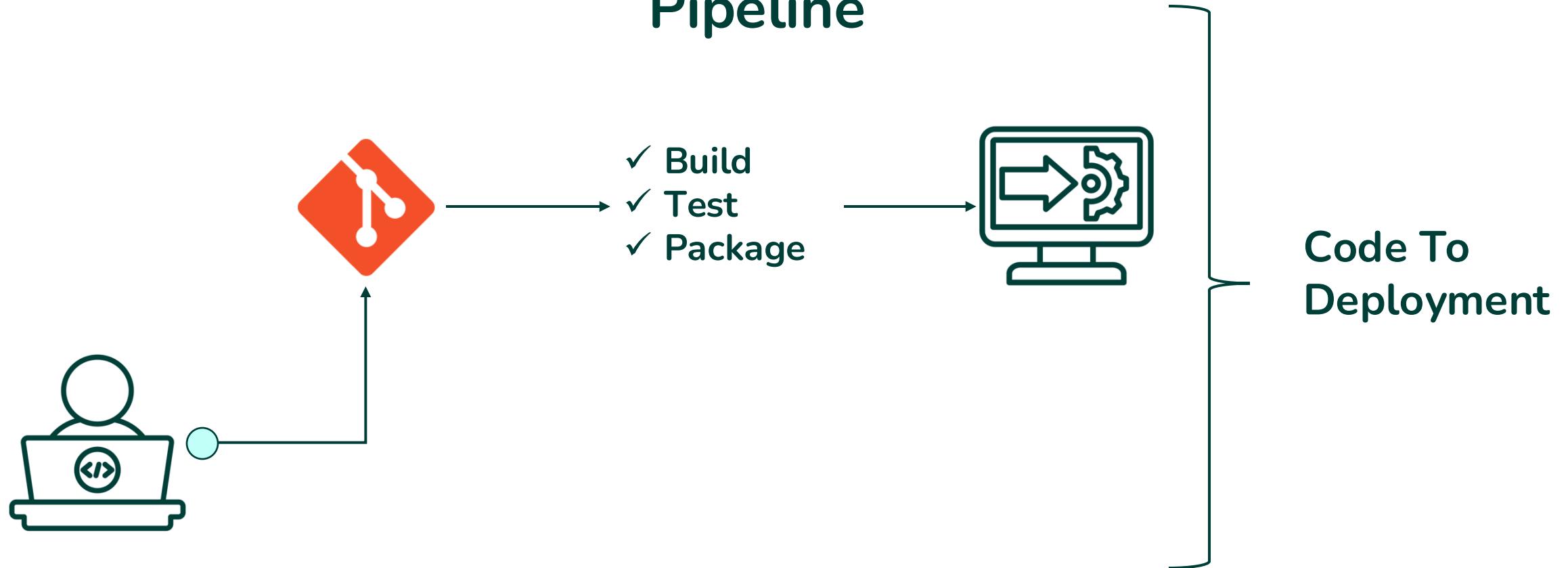


Introduction to GitOps

GitOps



CI/CD Pipeline





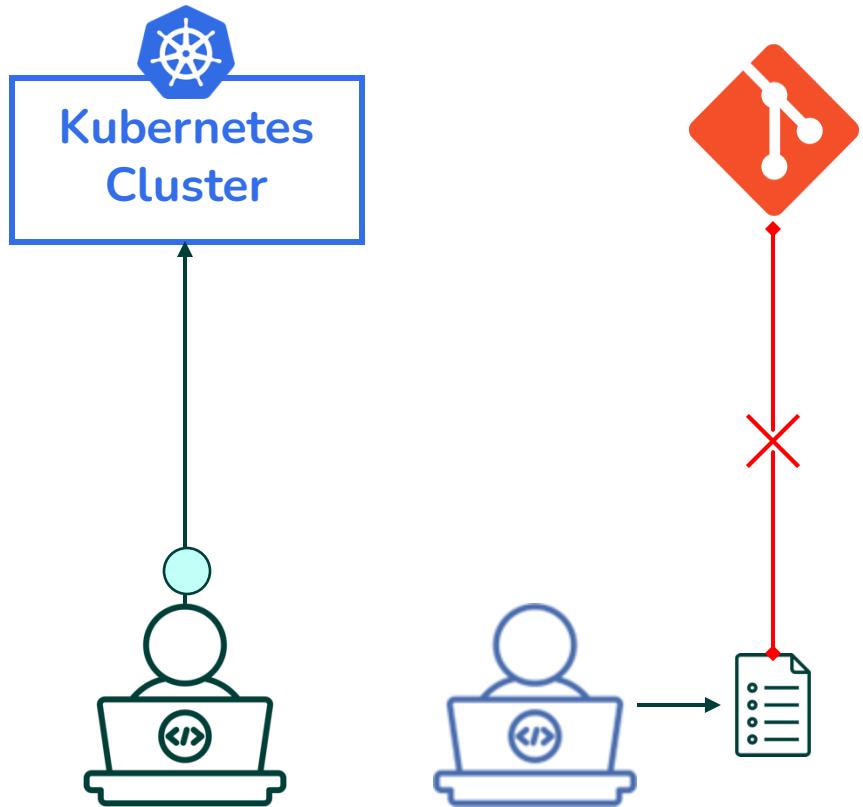
Who is managing the infrastructure configurations?



- Manually running **kubectl** commands
- Manually editing **YAML** files

How are changes to Kubernetes manifests or cluster configurations applied?





No Audit Trail

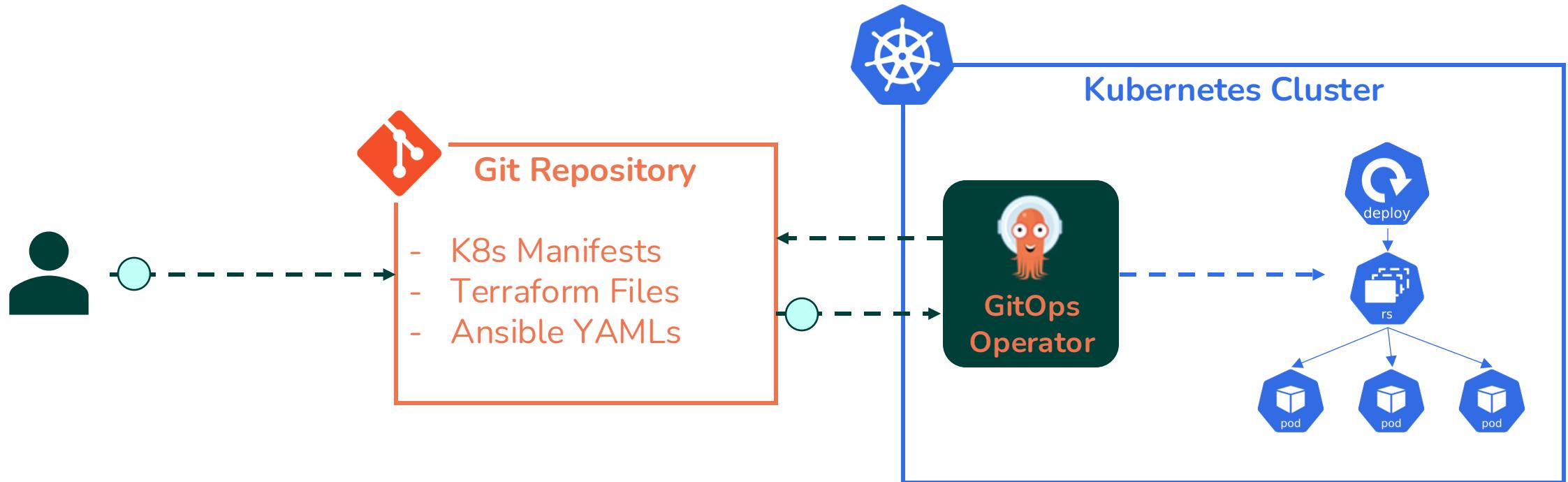
Configuration Drift

Disaster Recovery Nightmare



GitOps

Infrastructure & cluster configurations



GitOps

- ✓ Every change to cluster is stored in Git, creating a single source of truth
- ✓ Everything is automated and versioned
- ✓ In case of failure, you can restore the cluster by syncing with Git
- ✓ GitOps operator ensures the cluster's live state matches what's in Git, preventing unwanted changes

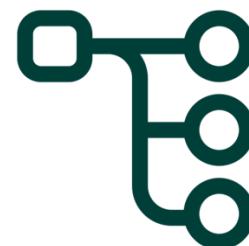


GitOps

GitOps is a modern approach to continuous deployment that uses Git as the single source of truth for your application code & infrastructure configurations



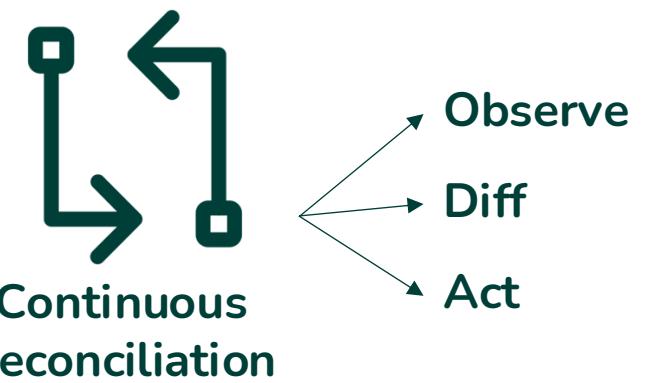
Declarative Configuration



Version Control



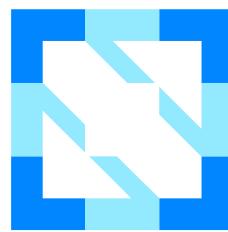
Automated Delivery



Overview of Argo Projects



Argo Projects



**CLOUD NATIVE
COMPUTING FOUNDATION**



Argo Projects



Argo
Workflows

Workflow engine for orchestrating tasks
like CI pipelines, data processing, and
batch jobs



Argo Projects



Argo CD

GitOps-based continuous delivery tool,
ensures Kubernetes clusters are synchronized
with Git repositories, automating deployment
and infrastructure management



Argo Projects



Argo Rollouts

Controller for advanced deployment strategies, including **Canary** releases, **Blue-Green** deployments, and **Progressive Rollouts**



Argo Projects



Event-driven automation framework that triggers workflows or actions based on external events

Argo Events



Argo Projects



Argo Workflows



Argo CD



Argo Rollouts



Argo Events

- ✓ **Workflows**
- ✓ **Deployments**
- ✓ **Event-driven automation**



Introduction to Argo CD



What is Argo CD?

Argo CD is a declarative, GitOps-based CD tool for Kubernetes that automates app deployments.
It syncs the Git-defined desired state with the actual state of the cluster.

- ✓ Monitors the **current state** of applications running in the cluster
 - ✓ Detects any **deviations** from the desired state stored in Git
 - ✓ Provides **visualizations** and **automated reconciliation**



Why Argo CD?

Declarative
& Version
Controlled

Automation
&
Remediation

Scalability
&
Flexibility



How Argo CD Works?



Single source of truth



How Argo CD Works?



Syncing
Applications



Monitoring
State



Reconciliation
Loop



Visual
Interface



Features of Argo CD



Features of Argo CD

- Automated Application Deployment
- Support for Multiple Configuration Tools
- Multi-Tenancy & RBAC
- Rollback & Roll-anywhere
- Web UI and CLI
- Audit Trails
- Multi-Cluster Management
- Single Sign-On (SSO) Integration
- Drift Detection & Synchronization
- Health Status Analysis
- Webhook & Automation Integration
- Observability & Metrics





Demo

Argo CD Documentation



Documentation Demo

Argo CD - Declarative GitOps C x +

argocd.readthedocs.io/en/stable/

Argo CD - Declarative GitOps CD for Kubernetes

Search GitHub v2.13.3 18.3k 5.6k

Argo CD - Declarative GitOps CD for Kubernetes

Overview

What Is Argo CD?

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes.

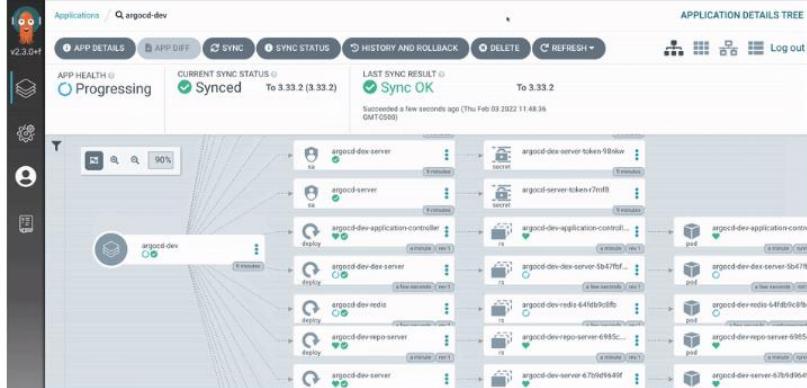


Table of contents

- What Is Argo CD?
- Why Argo CD?
- Getting Started
- Quick Start
- How it works
- Architecture
- Features
- Development Status
- Adoption

Getting Started

Quick Start

Copyright © Thinknyx Technologies LLP

Setting up Argo CD



Setting up Argo CD



Section Overview

- Argo CD installation options
- Prerequisites to install Argo CD





Argo CD Installation Types

Core Installation

This setup doesn't include UI, API server and minimizes resource usage

Multi-Tenant Installation

Non-H.A. Mode

- `install.yaml`
- `namespace-install.yaml`

H.A. Mode

- `install.yaml`
- `namespace-install.yaml`



`install.yaml`

Requires cluster admin access

Argo CD can deploy applications
on the same cluster where it runs

Access Requirements

Application Deployment Scope

`namespace-install.yaml`

Requires only namespace-level access

Argo CD cannot deploy apps in the
same cluster where it runs but can
manage and deploy apps in the other
clusters





Pre-Requisites for Installing Argo CD

Kubernetes cluster

kubectl CLI installed



Demo

Installing Argo CD on
Kubernetes



Demo

Argo CD
UI Walkthrough



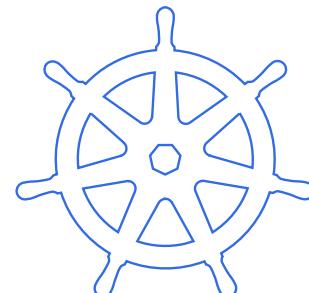
Conclusion



KCNA

(Kubernetes and Cloud Native Associate)

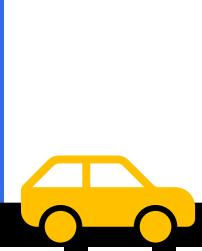
Certification Preparation



Kubernetes Fundamentals

Cloud-native Architecture

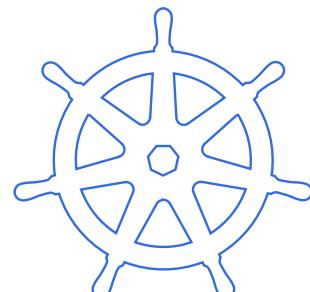
Cloud-native Application Delivery

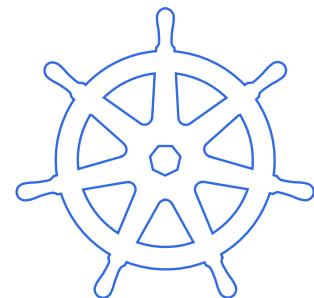


**Container
Orchestration**



**Cloud-native
Observability**







Follow us on:

 @thinknyx

 @thinknyx

 @thinknyx-technologies

 @thinknyx

 @thinknyx-technologies