Container Orchestration using Kubernetes

TECHNOLOGY

Core Concepts

A Day in the Life of a DevOps Engineer

You have recently started working as a DevOps engineer and are struggling to comprehend the functionality, different components, and features of Kubernetes.

The goal is to understand how the cluster is configured and how pods are created. Your team is looking for someone who can deploy applications, maintain their status, and provide all cluster details.

To achieve these and some additional features, you will learn a few concepts in this lesson that will help you find a solution for the given scenario.

Learning Objectives

By the end of this lesson, you will be able to:

- Distinguish between Docker and Containerd for effective container runtime selection
- Explore Containerd thoroughly and effectively apply its crictl commands for container management
- Comprehend the workings of container orchestration in Kubernetes for efficient container management
- Describe the functions of Etcd, Controller, Scheduler, and Kubelet for seamless container orchestration within Kubernetes

Learning Objectives

By the end of this lesson, you will be able to:

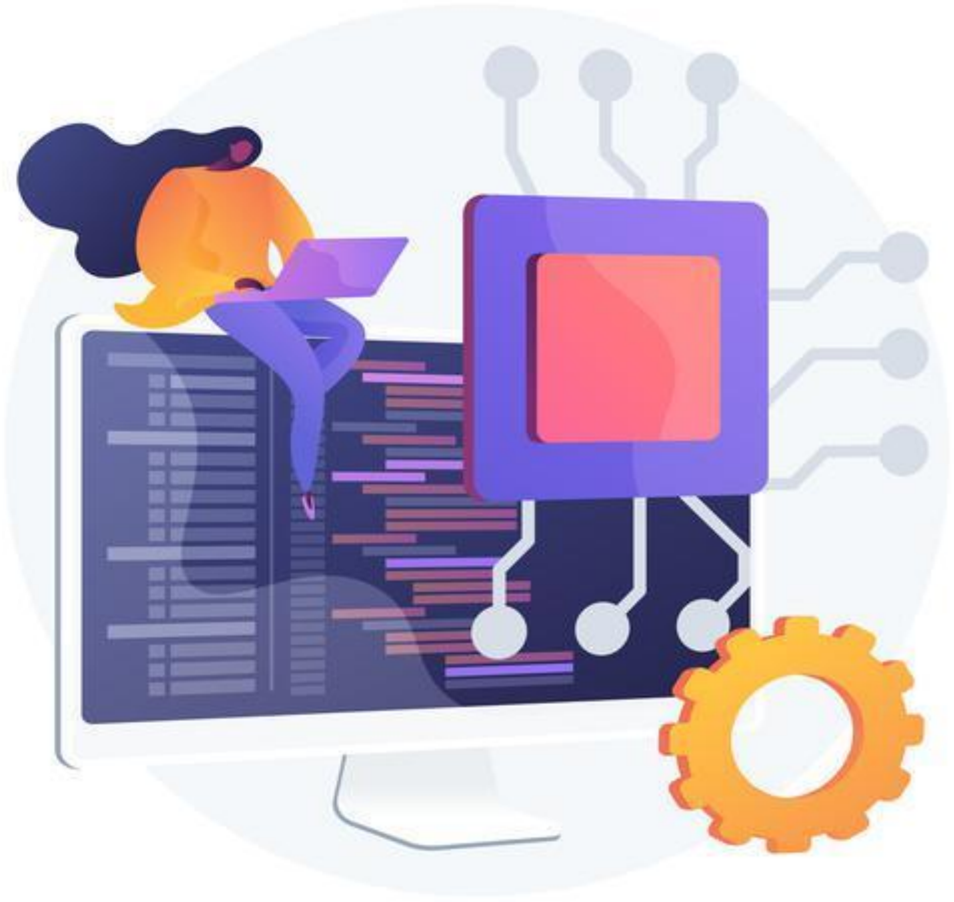
- Differentiate kube-proxy, pods, ReplicaSet, and deployment to manage Kubernetes clusters
- Classify common deployments use cases to facilitate organized management
- Summarize the concepts of containers and policies in Kubernetes to articulate their significance in practical use

TECHNOLOGY

Overview of Kubernetes

Kubernetes Design Principles

Kubernetes is a collection of building components. The components work together to offer mechanisms for deploying, maintaining, and scaling applications.

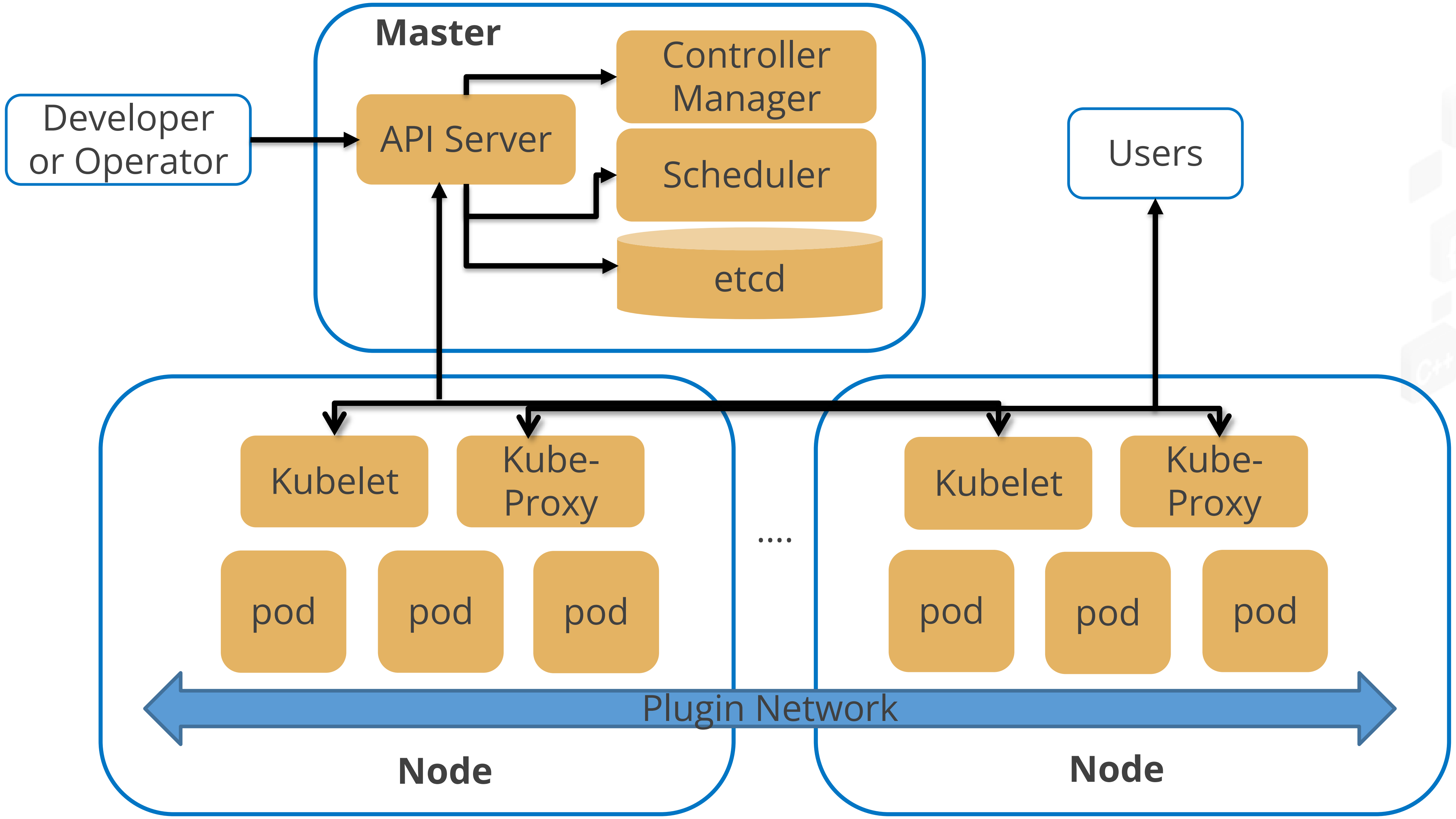


The platform gains control over computing and storage resources by defining resources.

Kubernetes is coupled and expanded to handle a variety of workloads and scheduling scenarios.

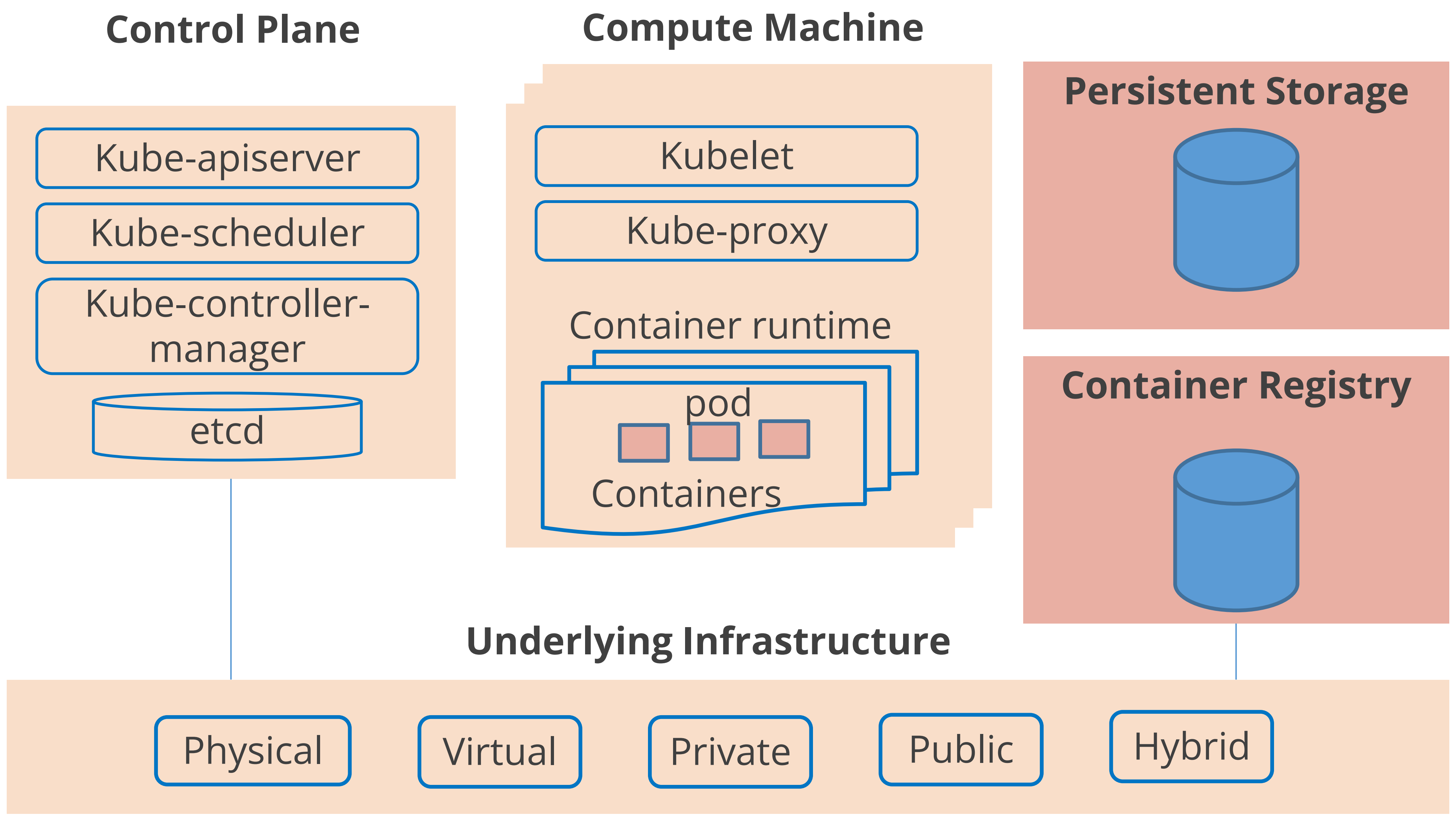
Kubernetes Architecture

It consists of a master (control plane), a distributed storage system (etcd) for maintaining cluster state consistency, and several cluster nodes.



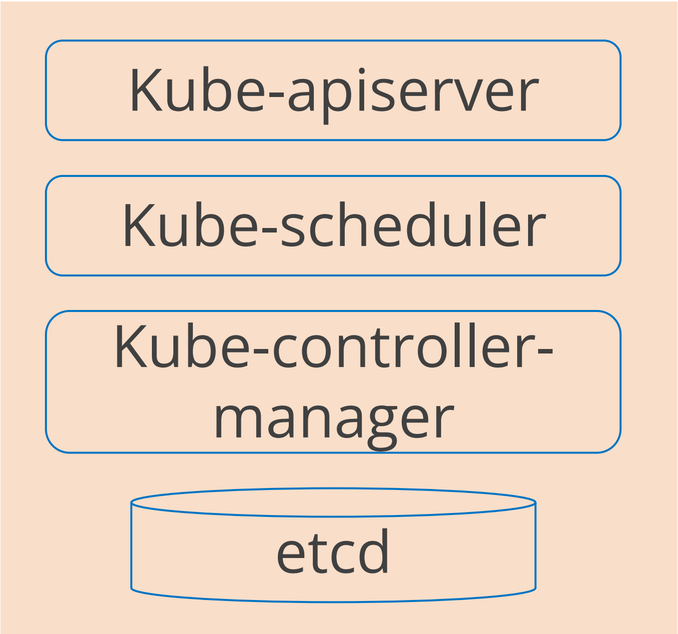
Kubernetes Cluster

It consists of a master node and a set of worker nodes.



Control Plane (Master)

It is in constant contact with the compute machines and ensures that the containers are running on the necessary resources.



Kube-apiserver handles internal and external requests.

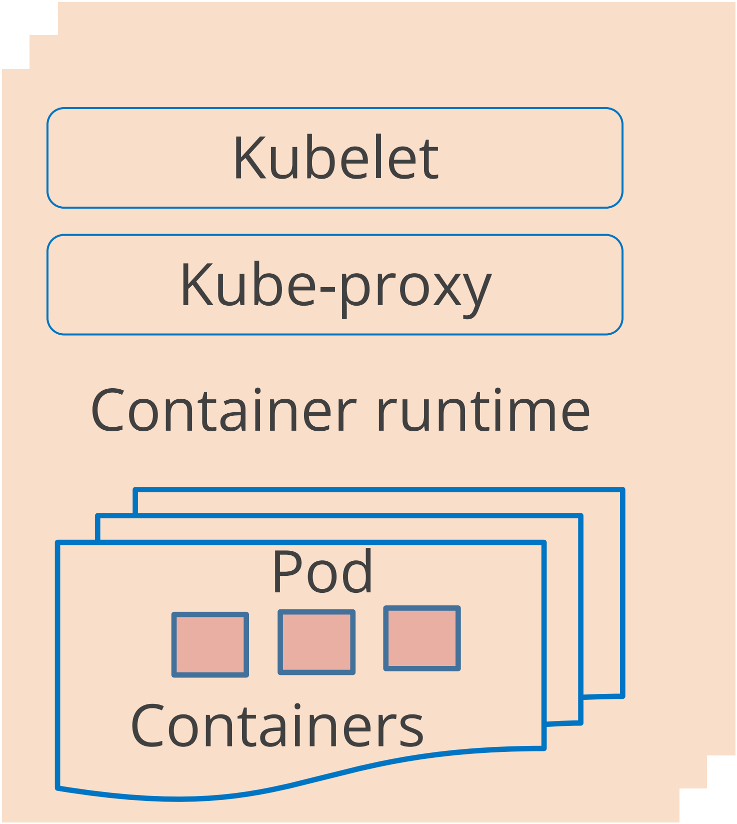
Kube-scheduler schedules the pod to an appropriate node.

Kube-controller-manager helps run the cluster.

etcd stores configuration data and information about the state of cluster lives.

Compute Machine (Worker)

A Kubernetes cluster requires at least one compute node. pods are scheduled and arranged to run on nodes. The control plane is not schedulable by default because of the default taint value.



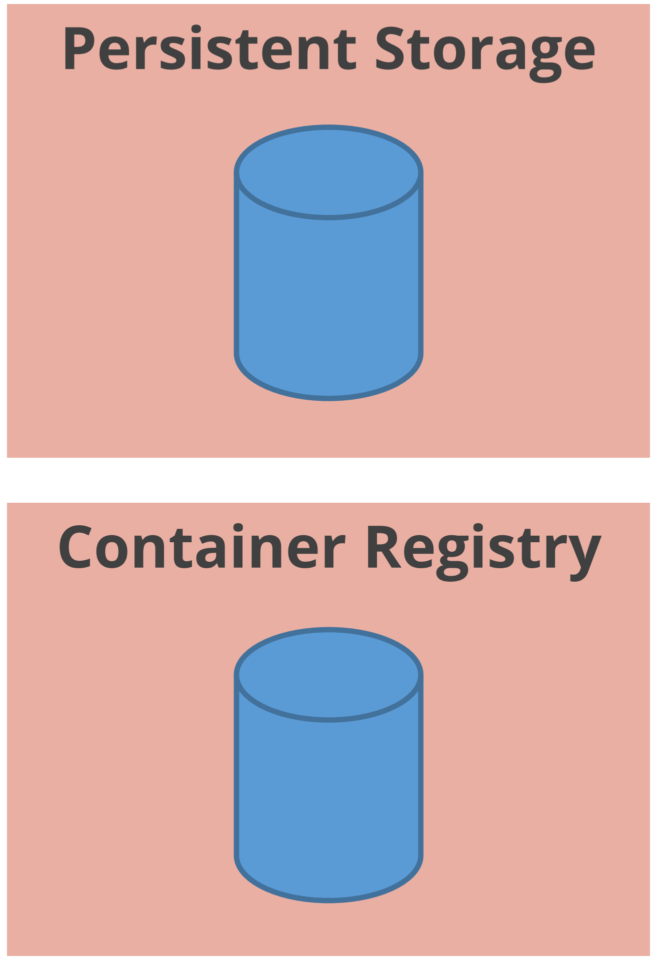
Kubelet communicates with the control plane.

Kube-proxy is a network proxy that facilitates Kubernetes network services.

Container runtime manages and runs the components required to run containers.

Pod represents a single instance of an application.

Storage and Registry



Persistent Storage manages application data attached to a cluster.

Container Registry stores the container images that Kubernetes relies on.

Features of Kubernetes

Kubernetes offers the following features to its users:

- Storage
- Secure configuration management
- Auto scaling
- Automated rollouts and rollbacks
- Service discovery and Load Balancing

Container

It is a standard unit of software that aids in the packaging of both application source code and dependencies.

It runs in four modes:

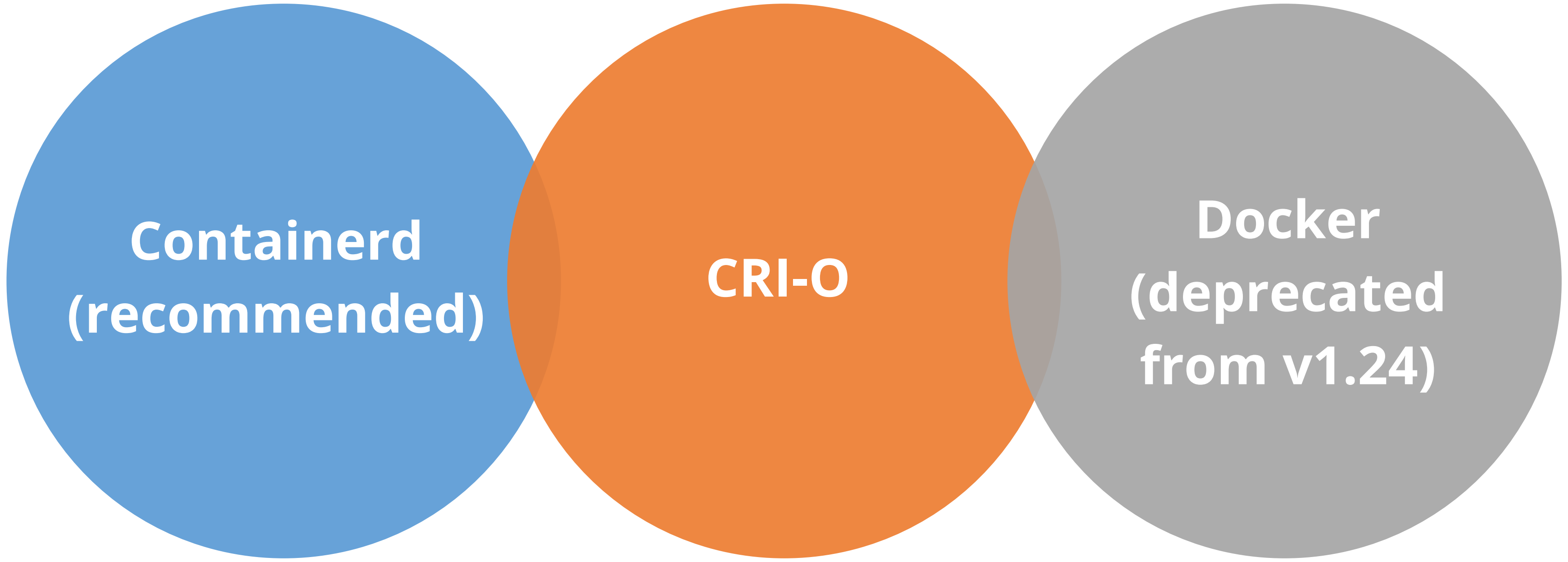
- Physical computers
- Virtual Machines
- Bare metal
- Clouds

Prepackaging allows the software to run fast and reliably from one computing environment to another. It can run on any compatible infrastructure.

Container Runtime

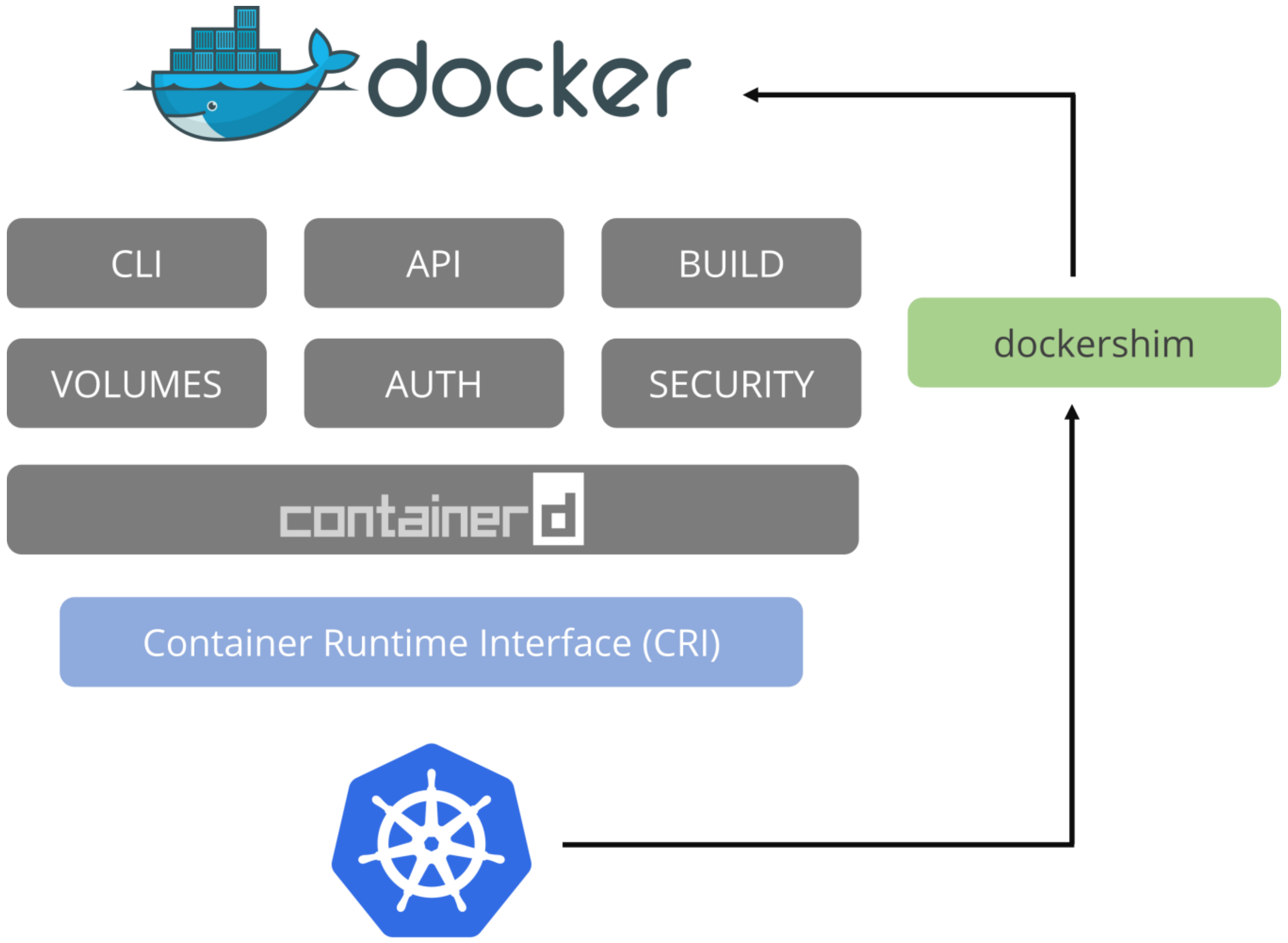
It is the software that is responsible for running containers.

Commonly used container runtimes with Kubernetes are:



Container runtime interface (CRI) is an API for container runtime to integrate with Kubelet.

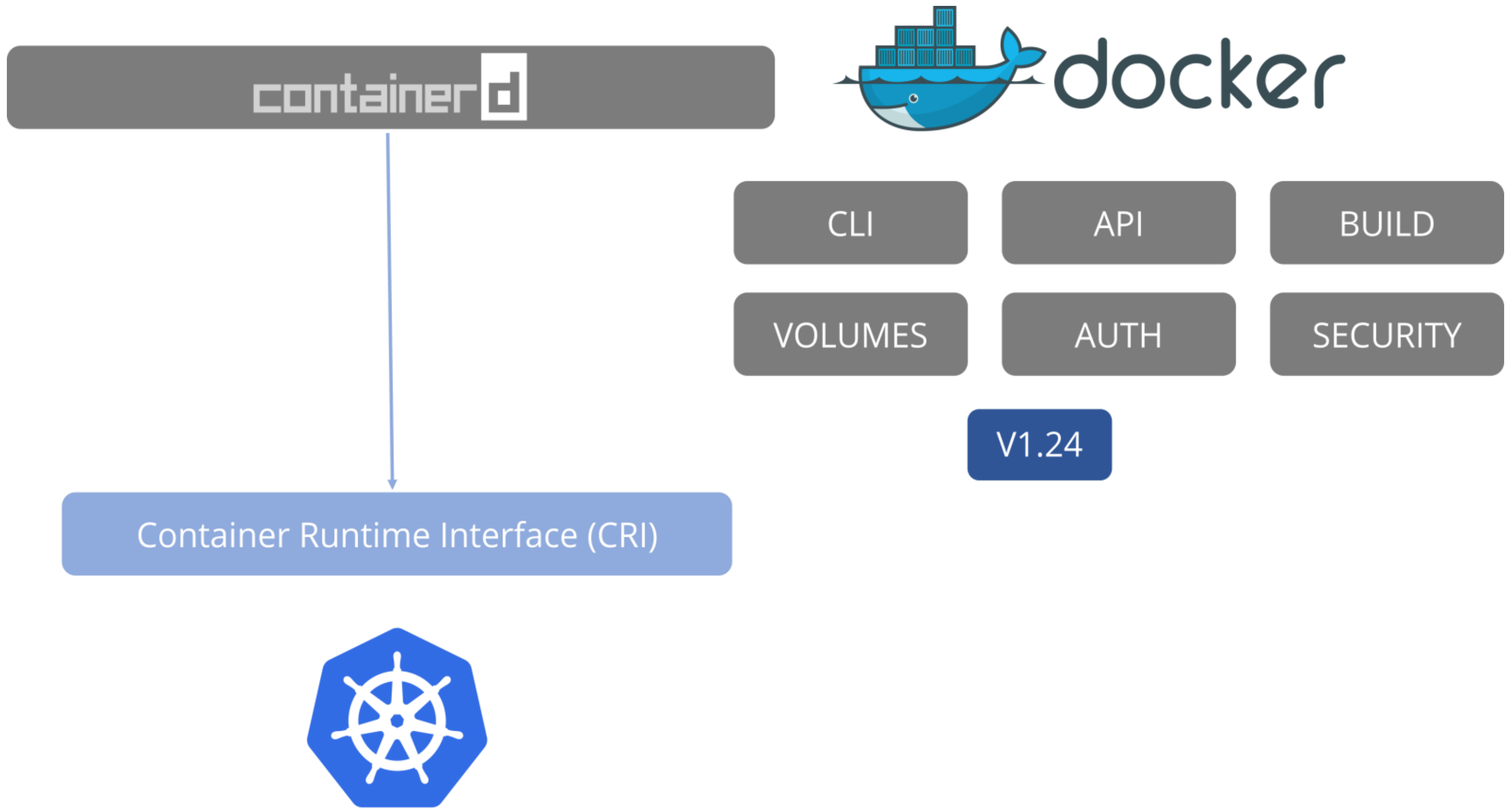
Docker vs. Containerd



Kubernetes releases before v1.24 included a direct integration with Docker Engine, using a component named dockershim.

That special direct integration is no longer part of Kubernetes (this removal was announced as part of the v1.20 release).

Docker vs. Containerd

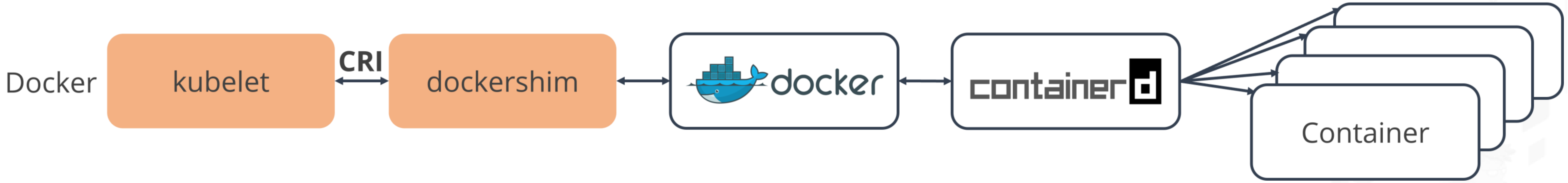


Containerd is CRI-compatible and can directly integrate with Kubernetes, like all other runtimes. Consequently, one can use containerd as a standalone runtime, separate from Docker.

Containerd initially started as an integral part of Docker but was later donated to the Cloud Native Computing Foundation (CNCF).

Docker vs. Containerd

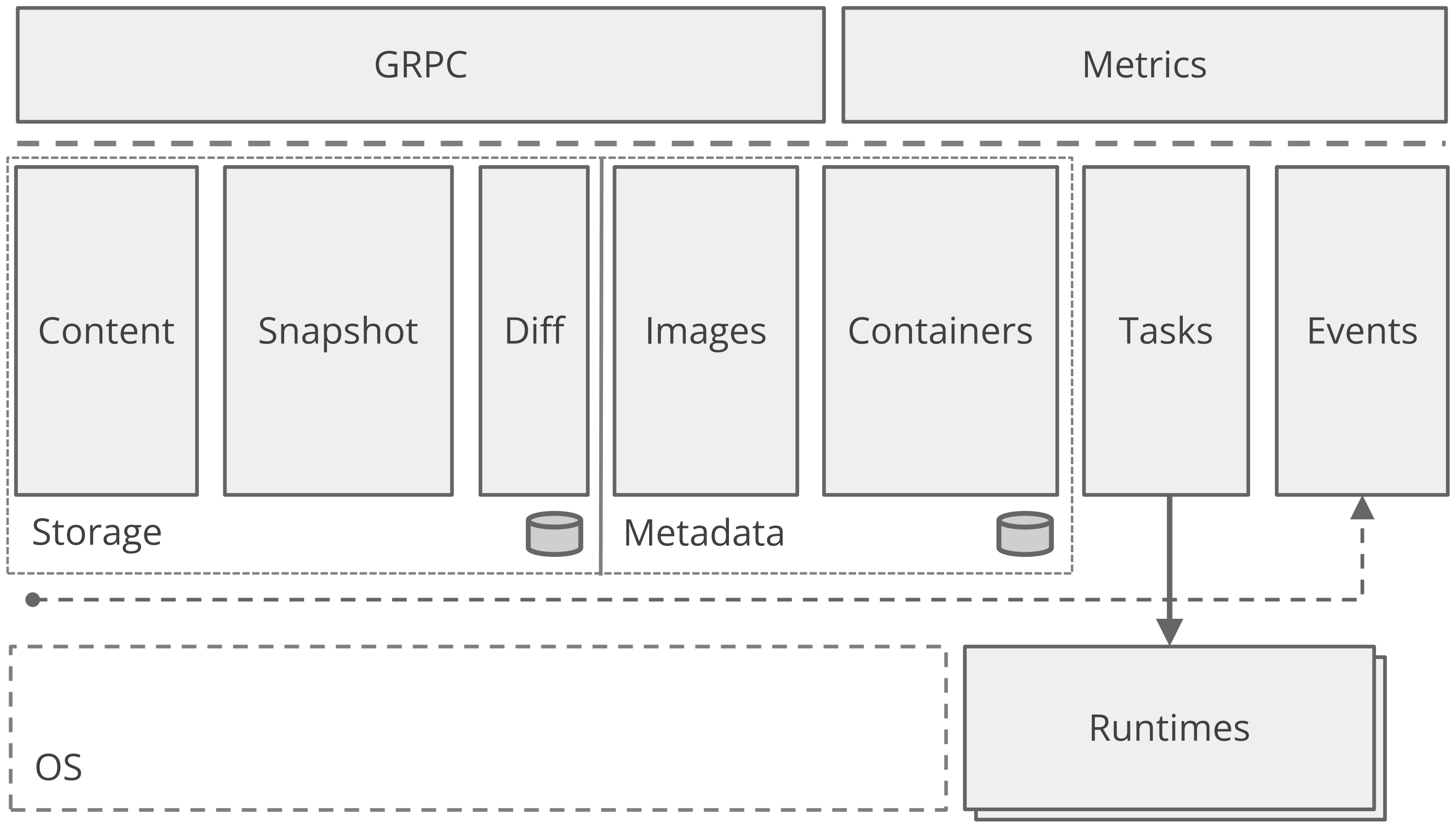
Before Kubernetes v1.24, Kubelet utilized Docker with containerd as an internal component to create containers.



Starting from Kubernetes v1.24, Kubelet directly employs containerd for creating containers.

Containerd

It is an industry-standard container runtime that prioritizes simplicity, robustness, and portability.



It serves as a Linux daemon capable of managing the entire container lifecycle of its host system.

This includes tasks such as image transfer and storage, container execution and supervision, low-level storage, and network attachments.

Containerd CLI

Ctr and nerdctl lack user-friendliness, while crictl commands closely resemble Docker commands.

containerd

kubernetes

Container Runtime Interface (CRI)

	ctr	nerdctl	crictl
Purpose	Debugging	General Purpose	Debugging
Community	ContainerD	ContainerD	Kubernetes
Works With	ContainerD	ContainerD	All CRI Compatible Runtimes

The *crictl* command also possesses pod awareness, allowing you to list pods by executing the *crictl pods* command, a capability Docker did not have.

Crictl

Crictl is a command-line interface for container runtimes that are compatible with CRI. It allows you to inspect and debug container runtimes and applications on a Kubernetes node.

The following commands are to retrieve debugging information:

docker cli	crictl	Description	Unsupported features
attach	attach	Attach to a running container	--detach-keys, --sig-proxy
exec	exec	Run a command in a running container	--privileged, --user, --detach-keys
images	images	List images	
info	info	Display system-wide information	

Crictl

docker cli	crictl	Description	Unsupported features
Inspect	inspect	Return low-level information on a container, image or task	
logs	logs	Fetch the logs of a container	--details
ps	ps	List containers	
stats	stats	Display a live stream of container(s) resource usage statistics	Column: NET/BLOCK I/O, PIDs
version	version	Show the runtime (Docker, ContainerD, or others) version information	

Crictl

The following commands are to perform changes:

docker cli	crictl	Description	Unsupported features
create	create	Create a new container	
kill	stop (timeout = 0)	Kill one or more running container	--signal
pull	pull	Pull an image or a repository from a registry	--all-tags, --disable-content-trust
rm	rm	Remove one or more containers	

Crictl

docker cli	crictl	Description	Unsupported features
rmi	rmi	Remove one or more images	
run	run	Run a command in a new container	
start	start	Start one or more stopped containers	--detach-keys
stop	stop	Stop one or more running containers	
update	update	Update configuration of one or more containers	--restart, --blkio-weight and some other resource limit not supported by CRI

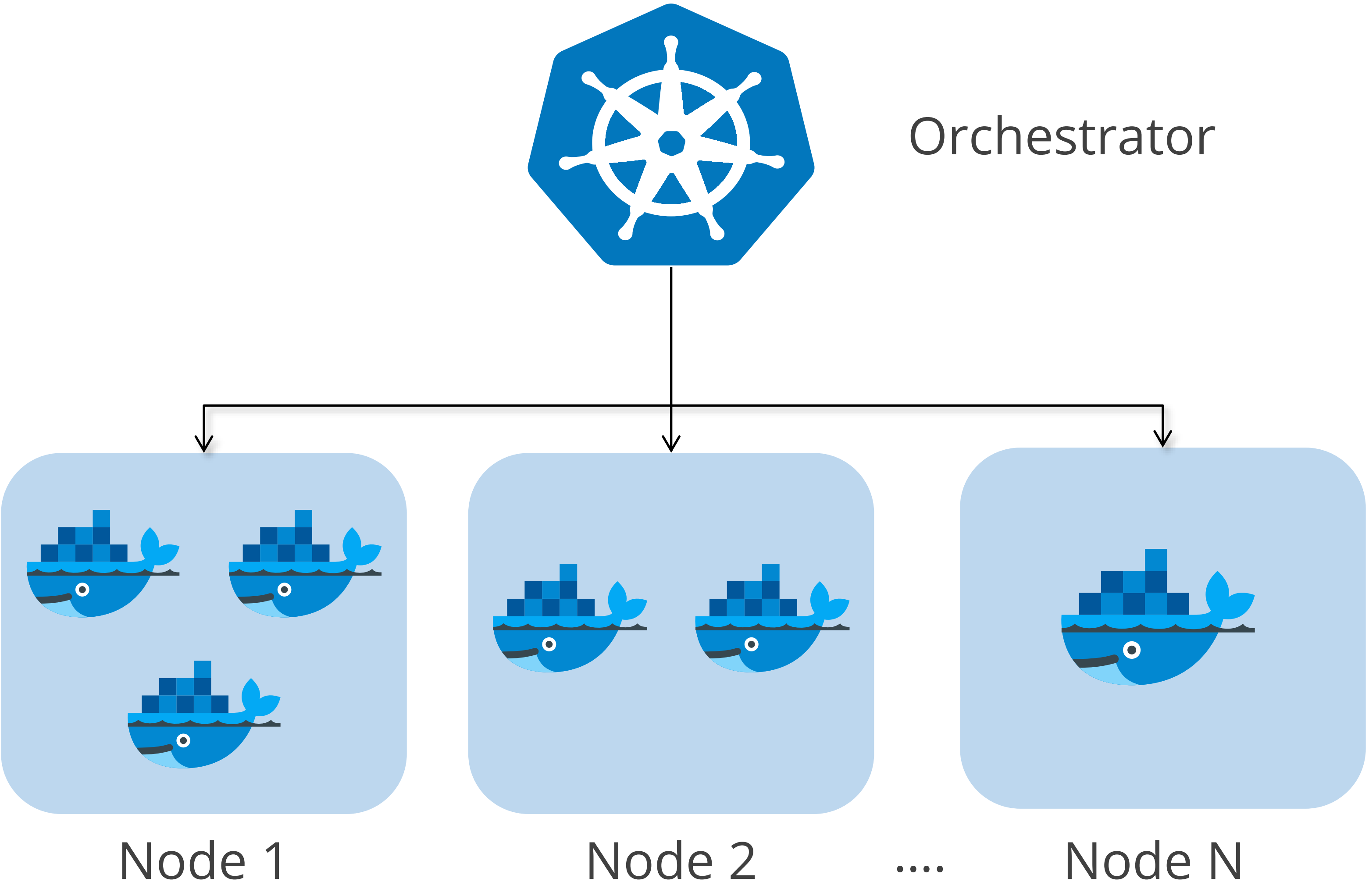
Crictl

The following commands are only supported only in *crictl*, not in Docker:

crictl	Description
imagefsinfo	Return image filesystem info
inspectp	Display the status of one or more pods
port-forward	Forward local port to a pod
pods	List pods
runp	Run a new pod
rmp	Remove one or more pods
stopp	Stop one or more running pods

Container Orchestration

It automates and simplifies the provisioning, deployment, and management of containerized applications.



Kubernetes: Overview

The aspects of Kubernetes are as follows:

Kubernetes Components

A Kubernetes cluster consists of components that represent the control plane and a set of machines called nodes.

They are divided into:

Master component

Node components

Master Components

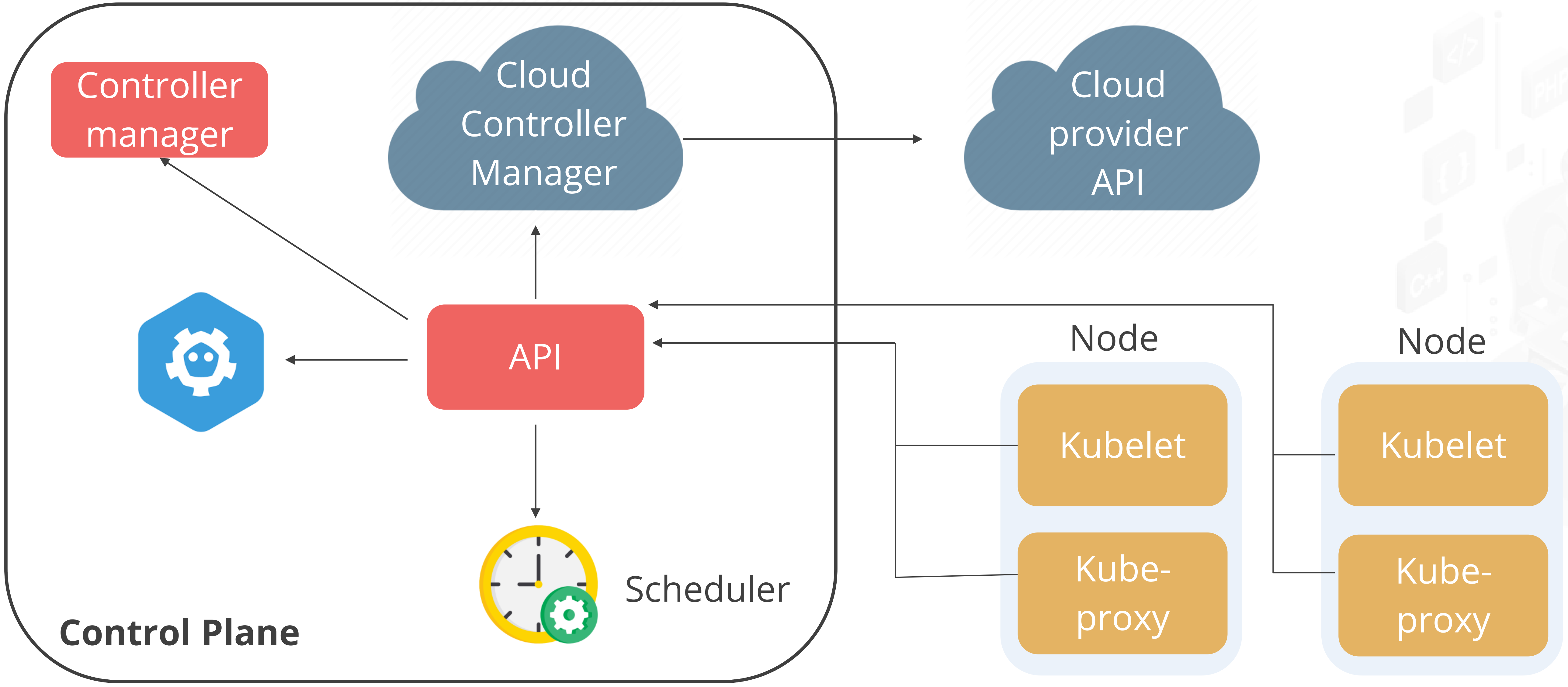
They are designed to monitor the cluster and respond to cluster events. These components are running as a pod on master in the kube-system namespace.

- Kube-apiserver
- Etcd
- Kube-Scheduler
- Kube Controller Manager
- Cloud-Controller Manager

Default config file is **/etc/kubernetes/admin.conf**

Cloud-Controller Manager

It helps to link clusters into the Cloud provider's API and separates components that interact with the Cloud platform.



Kubernetes API

It facilitates querying and manipulating the state of objects. The nucleus of the Control Plane in Kubernetes is the API server.

It helps to manipulate the state of API objects, such as:

- Pods
- Namespaces
- ConfigMaps and Events

Kubernetes Objects

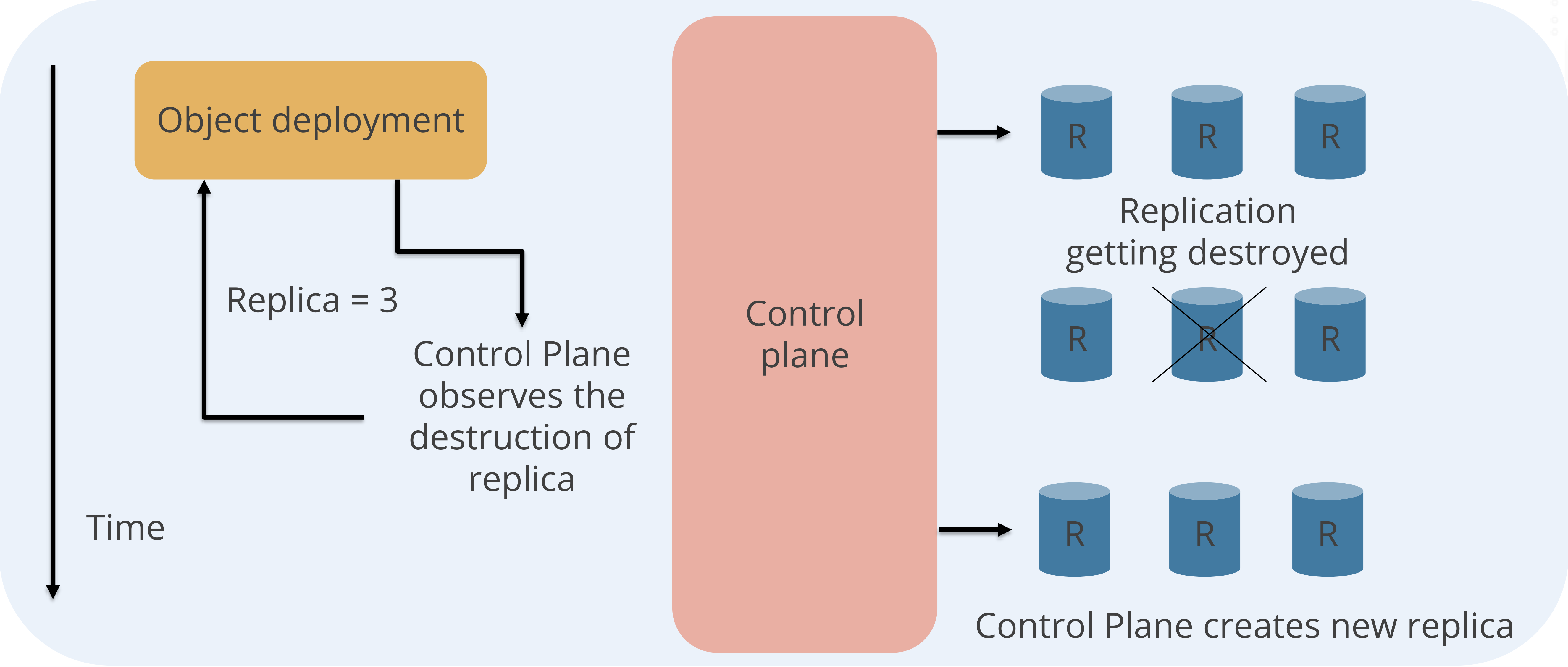
They are persistent entities in the Kubernetes system.

These objects can be described as:

- What containerized applications are running
- The resources available to those applications
- The policies around how applications behave, such as restart policies, upgrades, and fault-tolerance

Object Fields

Every Kubernetes object includes two nested object fields that govern the object's configuration, namely, object **spec** and **status**.



simplilearn

Describing Kubernetes Object

The object spec should be given while creating a Kubernetes object, which outlines the object's desired state as well as some basic information about it, such as a name.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-1-17
        image: k8s-master:31320/nginx:1.17
        ports:
        - containerPort: 80
```

Required Fields in .yaml File

In the .yaml file, a set of values creates a Kubernetes object for the following fields:

- apiVersion
- kind
- metadata
- spec

Creating and Configuring a Kubernetes Cluster

Duration: 20 mins

ASSISTED PRACTICE

Problem Statement:

You have been asked to create a Kubernetes cluster and add the nodes to it.

Assisted Practice: Guidelines

Steps to be followed:

1. Change the hostnames of all machines
2. Set up the master node
3. Join the worker nodes in the cluster

Demonstrating Crictl Commands for Container Runtime Operations

Duration: 20 mins

Problem Statement:

You have been asked to debug Kubernetes nodes with crictl CLI commands.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Configure and manage container runtime environment

TECHNOLOGY

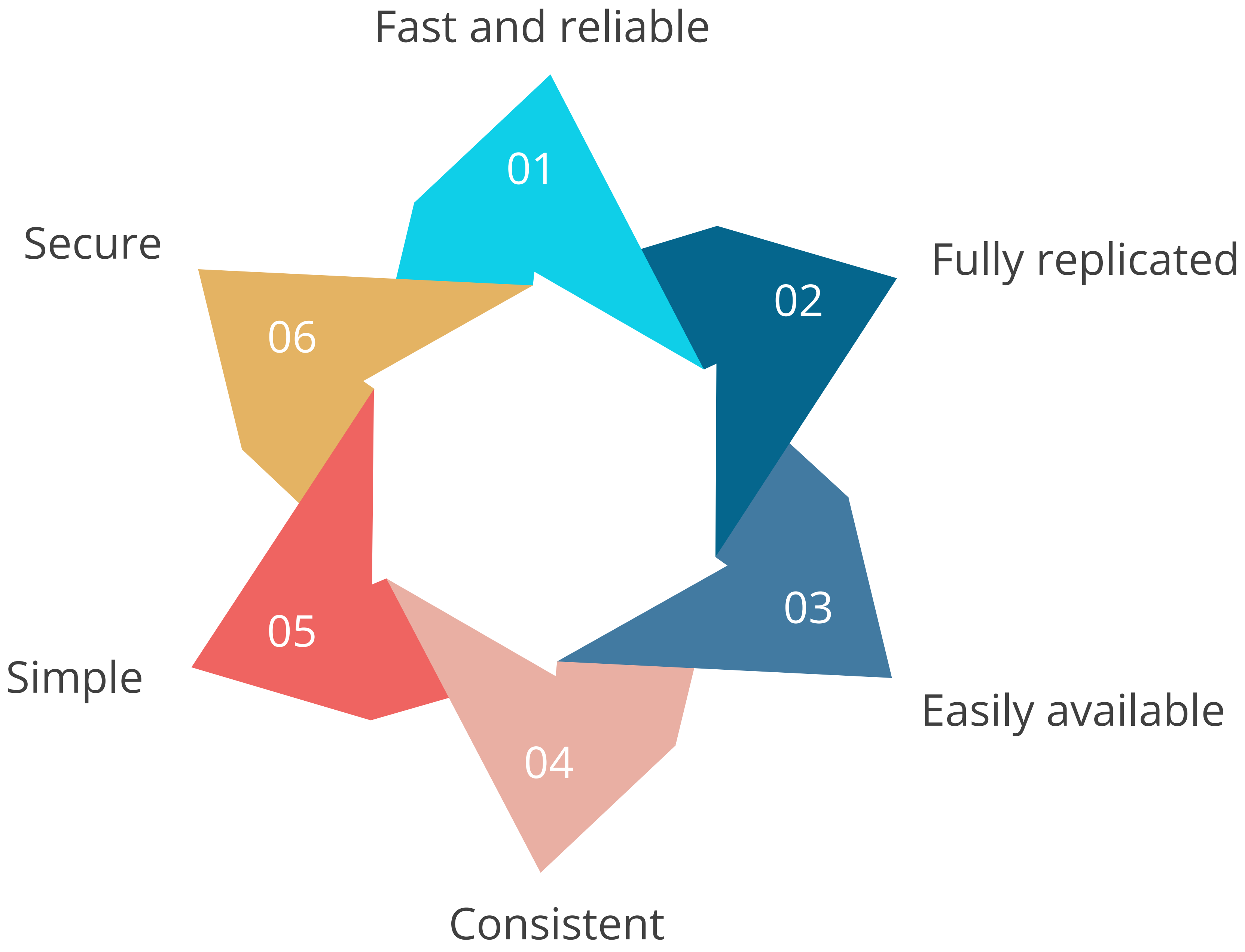
Etcd

Etcd

It is an open-source distributed key-value store for storing and managing important data that distributed systems require to function.

- Distributed key-value store that manages critical information
- Used in production environment by cloud service providers
- Used as a distributed datastore
- Used to manage configuration data, state data, and metadata

Features of Etcd



Working of Etcd

It is defined through three key concepts (Raft-based system):

Leaders	Handle all client requests that need consensus among clusters
Elections	Is used when the leader is down for any reason
Terms	Appear for multiple candidates

TECHNOLOGY

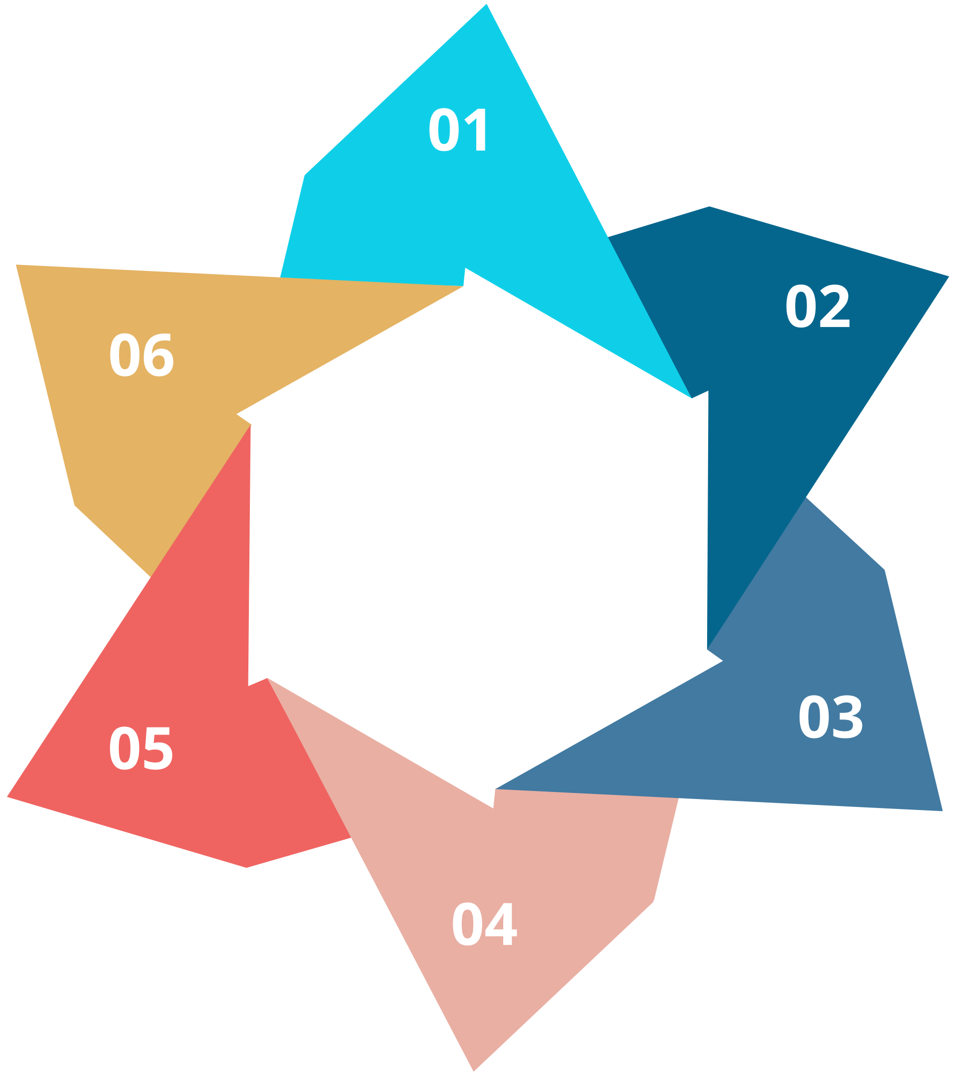
Controller

Controllers in Kubernetes

Built-in Controllers manage the state by interacting with the cluster API server.

Job Controller is a good example of a Kubernetes built-in controller.

Types of Controllers



DaemonSet: Ensures all nodes run a copy of a pod

ReplicaSet: Creates a stable set of pods

Deployment: Maintains ReplicaSets with a desired configuration

StatefulSet: Manages stateful applications

Job: Creates one or more short-lived pods

CronJob: Creates a job on schedule

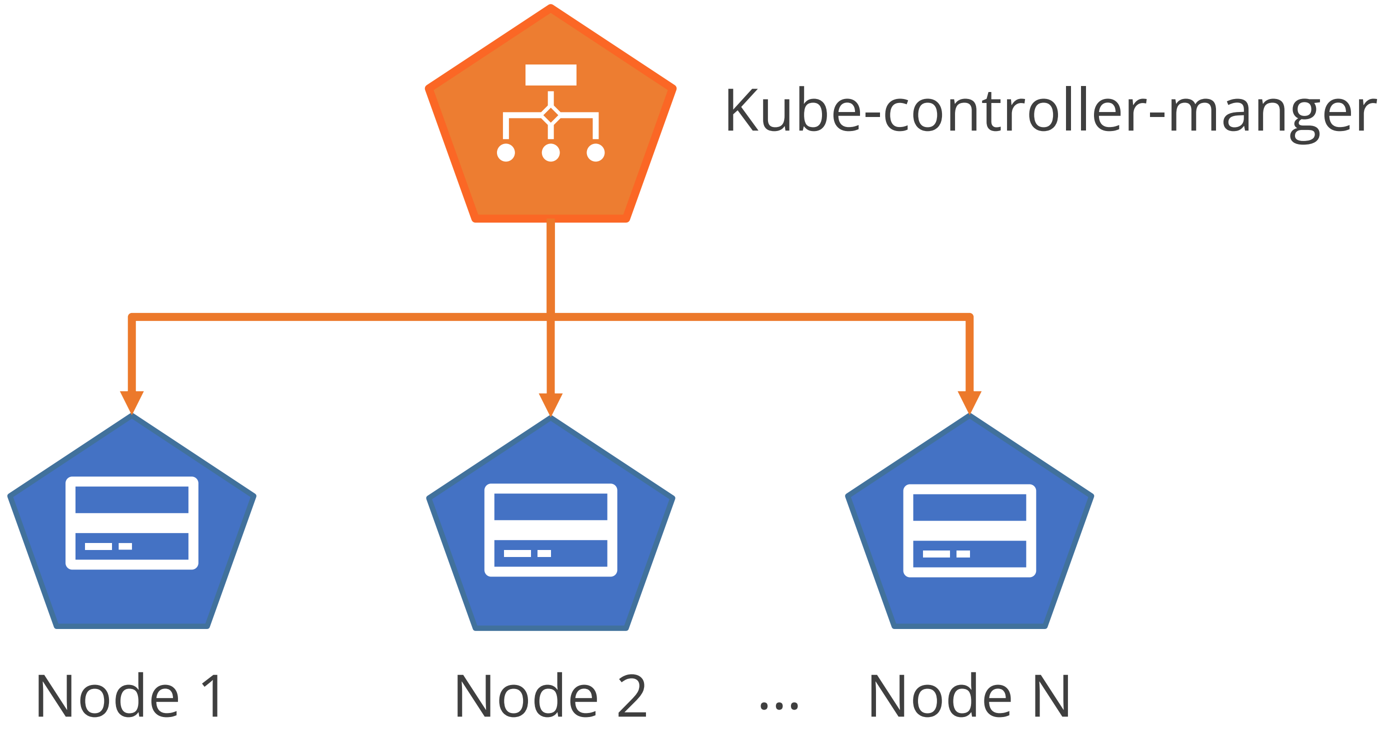
Controller Design

Kubernetes uses many controllers, each of which manages a specific feature of the cluster state.

- A specific control loop uses one kind of resource as its desired state.
- Kubernetes is designed to handle controller failures.

Working of Controllers

Kubernetes comes with a set of built-in controllers that run inside the Kube-controller-manager. These built-in controllers provide important core behaviors.



To extend Kubernetes, controllers in the system run outside the control plane.

Scheduler

Kubernetes Scheduler

Scheduling refers to ensuring that pods are matched to nodes so that a Kubelet can run them. A Scheduler watches for newly created pods that are not assigned to any nodes.

Pod:
Represents a set of running containers in a cluster

Node:
Is a worker machine in Kubernetes

Kube-scheduler

It is the default scheduler of any Kubernetes system and runs as a part of the control plane.

- Helps in writing scheduling components and using them
- Finds workable or feasible nodes for a pod and then runs a set of functions to score feasible nodes
- Provides optimal node for newly created pods

Node Selection in Kube-scheduler

The Kube-scheduler selects a node for the pod in a two-step operation:

Filtering	Finds the set of nodes where it is feasible to schedule the pod
Scoring	Ranks the nodes that remain to select the most suitable pod placement

Configuring Filtering and Scoring Behavior

There are two supported ways to configure the filtering and scoring behavior of a scheduler:

01 Scheduling policies

02 Scheduling profiles

Node Affinity and Anti-Affinity

Node affinity allows flexible decisions. YAML file configuration represents specific requirements.

Node anti-affinity is created to allow flexible decision-making processes.

TECHNOLOGY

Kubelet

Kubelet

It is a tiny application that communicates with the control plane. It makes sure that the containers are running in a pod.

It works in terms of a specification called PodSpec.

A PodSpec is a YAML or JSON object that describes a pod.

Providing Container Manifest to Kubelet

Apart from PodSpec, there are three ways to provide a manifest to Kubelet:

File

The path passed as a flag on the command line

HTTP endpoint

Endpoint passed as a parameter on the command line

HTTP Server

Listens for HTTP and responds to a simple API

TECHNOLOGY

Kube-proxy

simplilearn

Kube-proxy

It is a network proxy that runs on every node in a cluster, implementing the Kubernetes Service concept.

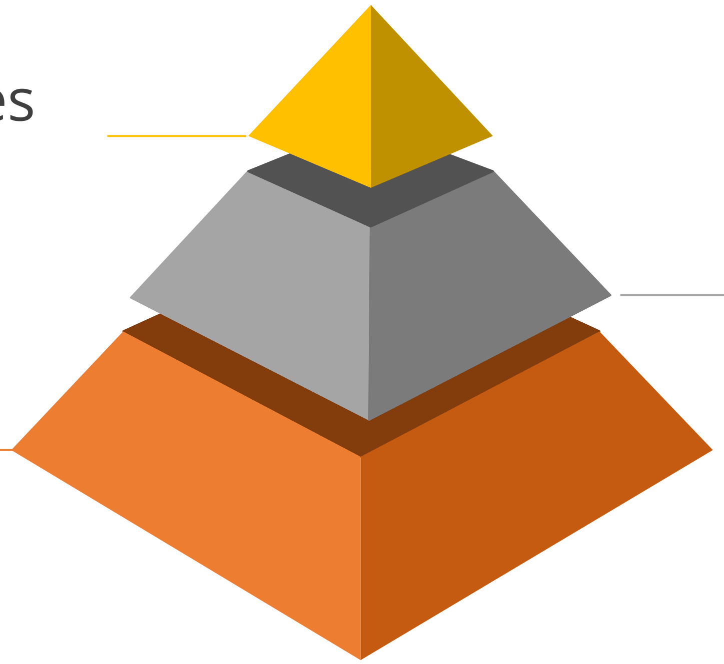
Maintains network rules on nodes

Manages forwarding of traffic addressed to the virtual IP addresses of the clusters

Operation Modes

Each node has a Kube-proxy container process. The Kube-proxy currently supports three different operation modes:

User space
Service routing takes place here



Iptables
Default for Kube-proxy on all platforms

IPVS (IP Virtual Server)
Built on Netfilter platform and supports load balancing

TECHNOLOGY

Containers

Introduction

A container image is a software package that is ready to use and contains everything needed to run an application:

- Libraries
- Binaries
- Additional configuration files

Containerized Application

They can be deployed without regard to the underlying infrastructure.

Containerized applications are isolated from each other, like virtual machines, increasing reliability and reducing problems resulting from inter-application interactions.

Benefits of Containers

Containers consume fewer resources than virtual machines or other outdated program architectures. As they are:

Lightweight

Scalable

Portable and consistent

Agile

Problems Containers Solve

Containers are versatile and solve a broad range of IT problems throughout an application's life cycle.

Ensures that software runs properly when moved between computing environments

Increases efficiency by eliminating the need for a separate hypervisor

Facilitates micro-service deployments

Eliminates conflicts and dependencies between multiple applications

Use Cases for IT Operations

Improve application security by isolating it from other applications

Facilitate seamless migration of applications

Improve IT efficiency by enabling multiple application containers to run on a single OS instance

Offer on-demand scalability

How Do Containers Work?

Containers isolate applications from one another.

A registry or repository transfers container images and the application container engine which turns the images into executable code.

Container repositories facilitate reusing commonly used container images.

Containers are created using the process of packaging applications.

Containers: Features

Namespaces provide access to the underlying operating system.

Control Groups implement resource accounting and resource limitation.

Union File Systems prevent data duplication.

TECHNOLOGY

Pods

Understanding Pods

Pods are the smallest deployable unit of computing, which can be created and managed in Kubernetes.

Pods in a Kubernetes Cluster are mainly used in two ways:

1. Pods that run a single container; the most common Kubernetes use case is the "one-container-per-pod" model.
2. Pods that run multiple containers, which should work in conjunction with each other.

Understanding Pods

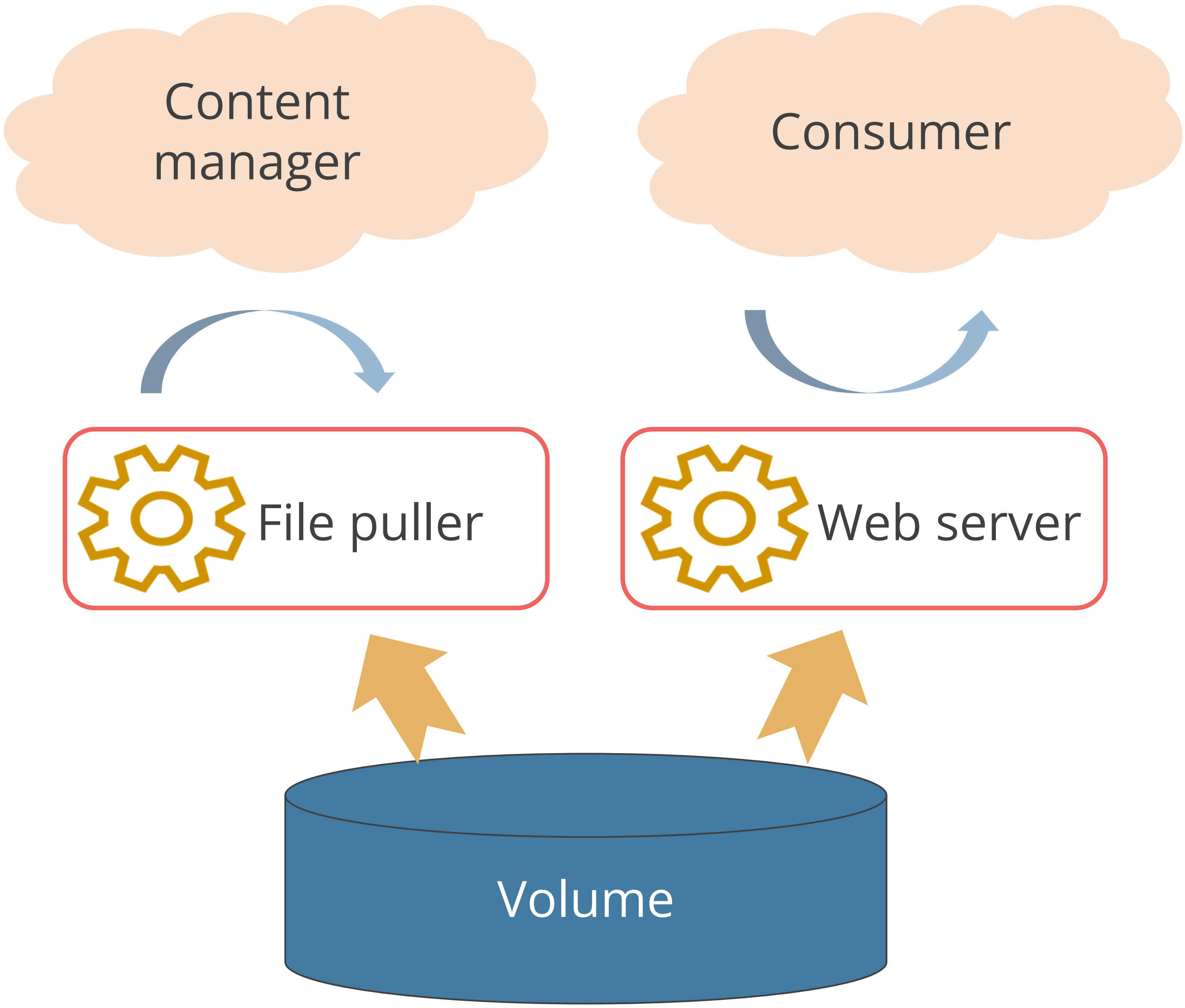
Pods also contain shared networking and storage resources for their containers:

Network:
Pods are automatically assigned unique IP addresses.

Storage:
Pods can specify a set of shared storage volumes that can be shared among the containers.

How Pods Manage Multiple Containers?

Pods are designed to support multiple cooperating processes (as containers) that form a cohesive unit of service.



Pods and Controllers

Workload resources create and manage one or more pods.

Examples of workload resources:

Deployment

StatefulSet

DaemonSet

Pod Template

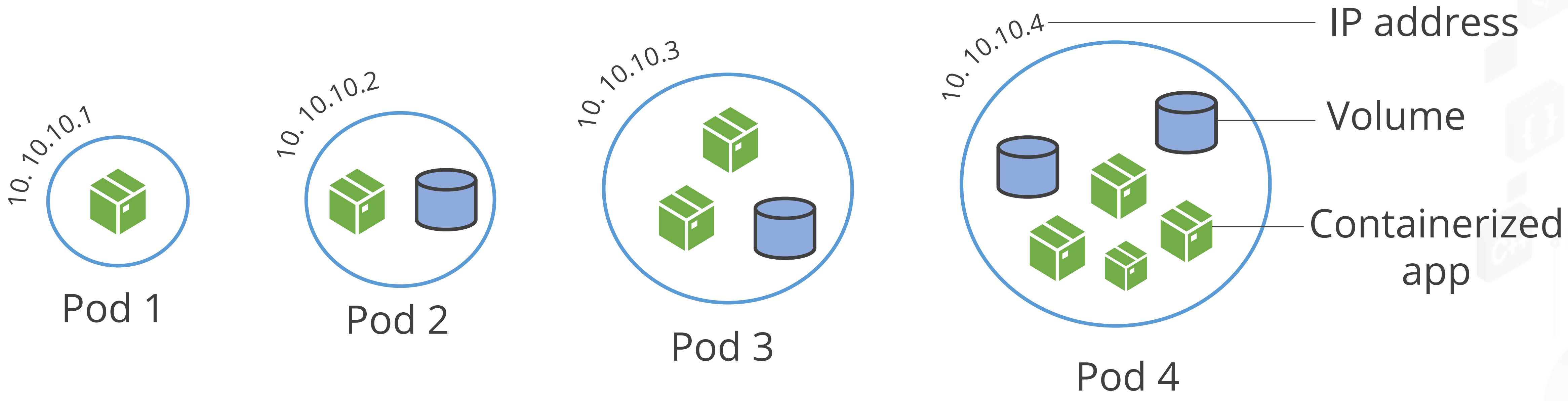
Pod templates are specifications for creating pods and are included in workload resources, such as deployments, Jobs, and DaemonSets.

Sample pod template:

```
apiVersion: v1
kind: pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Pod Update and Replacement

The controller does not update or patch existing pods when the pod template for a workload resource is updated or changed.



The controller creates new pods based on the updated template.

Pod Update and Replacement

Pod update operations like **patch** and **replace** have some limitations:

The metadata about a pod is immutable.

If the **metadata.deletionTimestamp** is set, no new entry can be added to the **metadata.finalizers** list.

Pod updates may not change fields.

When updating the **spec.activeDeadline** seconds field, two types of updates are allowed.

Resource Sharing and Communication

Pods enable data sharing and communication among their constituent containers using two methods:

Storage in pods: All containers in a pod have access to the shared volumes, allowing those containers to share data.

Pod Networking: When containers within a pod communicate with entities outside the pod, they must coordinate how they use the shared network resources.

Privileged Mode for Containers

Any container in a pod can get the Privileged Mode into working by utilizing the privileged flag on the security context of the container spec.

Privileged Mode is used for containers that use the operating system's administrative capabilities.

The processes in Privileged Mode have the same privileges as the processes outside a container.

Static Pods

The Kubelet daemon manages static pods directly on a specific node, without being observed by the API server.

The control plane manages most pods. The Kubelet supervises every static pod directly.

Configuring Pods in the Kubernetes Cluster

Duration: 15 mins

ASSISTED PRACTICE

Problem Statement:

You have been assigned a task to create, configure pods, and execute the Apache services.

Assisted Practice: Guidelines

Steps to be followed:

1. Configure and set up the pod files
2. Configure and set up the Service file
3. Execute the Apache services

TECHNOLOGY

ReplicaSets

ReplicaSets

It maintain a stable set of replica pods running at any given time.

It guarantees the availability of a specified number of identical pods.

It ensures that a specified number of pod replicas are running.

Operators to Use with ReplicaSets

There are three important operators that play a crucial role in managing and configuring ReplicaSets within a Kubernetes cluster:

01 In

02 Notin

03 Exists

Ensure that the selectors of one ReplicaSet do not match another's.

ReplicaSet Manifest

The following is an example of a ReplicaSet manifest:

```
apiVersion: apps/v1 # our API version
kind: ReplicaSet   # The kind we are creating
Metadata: # Specify all Metadata like name, labels
  name: some-name
  labels:
    app: some-App
    tier: some-Tier
Spec:
  replicas: 3 # Here is where we tell k8s how many replicas we want
  Selector: # This is our label selector field.
    matchLabels:
      tier: some-Tier
    matchExpressions:
      - {key: tier, operator: In, values: [some-Tier]} # we are using the set-based
operators
  template:
    metadata:
      labels:
        app: some-App
        tier: someTier
    Spec: # This spec section should look like spec in a pod definition
      Containers:
```

Working of ReplicaSet

A ReplicaSet is defined with fields, including a selector that specifies:

- Identification of pods
- The number of pods to be maintained
- Data in new pods

It ensures that a specified number of pod replicas are running at any given time.

TECHNOLOGY

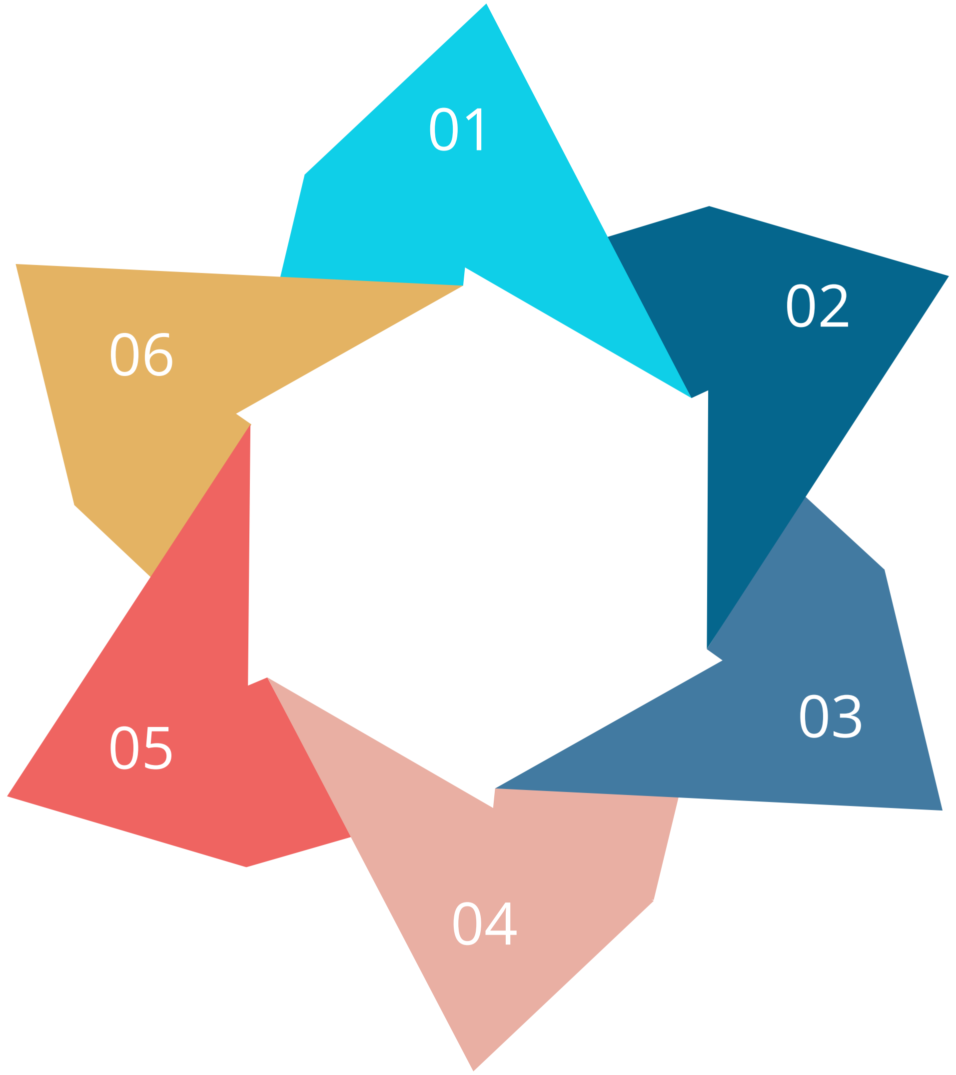
Deployments

Deployment

It provides declarative updates for pods and ReplicaSets.

It can be defined to create new ReplicaSets or remove existing Deployments.

Use Cases of Deployment



Use deployment status and clean up old ReplicaSets

Create a deployment to rollout a ReplicaSet

Use deployment to declare the new state of the pods

Rollback to an earlier deployment revision

Scale up the deployment to facilitate more load

Pause and resume the deployment

Creating a Deployment

The following is an example of a Deployment:

Example

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Updating and Rolling Back Deployment

Deployments can be updated by making changes to the pod template spec in the Deployment; it automatically generates an update rollout.

Rolling Back Deployment

```
kubectl rollout undo [deployment_name]

#Adding the argument

-to-revision=

#will roll back to that specific
version of the deployment
```

Scaling a Deployment

Deployments are useful for scaling the number of replicas as demand increases for a particular application.

Example:

```
# to scale a deployment up to 20 replicas

kubectl scale [deployment-name] –replicas 20
```

Pause and Resume

Multiple fixes can be applied between pausing and resuming without triggering unnecessary rollouts.

Example:

```
#Pause a deployment

kubectl rollout pause deployment.v1.apps/nginx-deployment

#Resuming a deployment

kubectl rollout resume deployment.v1.apps/nginx-deployment
```

Creating and Configuring the Deployment

Duration: 20 mins

ASSISTED PRACTICE

Problem Statement:

You have been assigned a task to create and configure deployment for an application.

Assisted Practice: Guidelines

Steps to be followed:

1. Create the deployment
2. Access the pod

TECHNOLOGY

Services, Load Balancing, and Networking

Services, Load Balancing, and Networking

Services and Load Balancing are the most important parts of Kubernetes networking, and they address four main concerns.

Containers in a pod use networking to communicate via loopback.

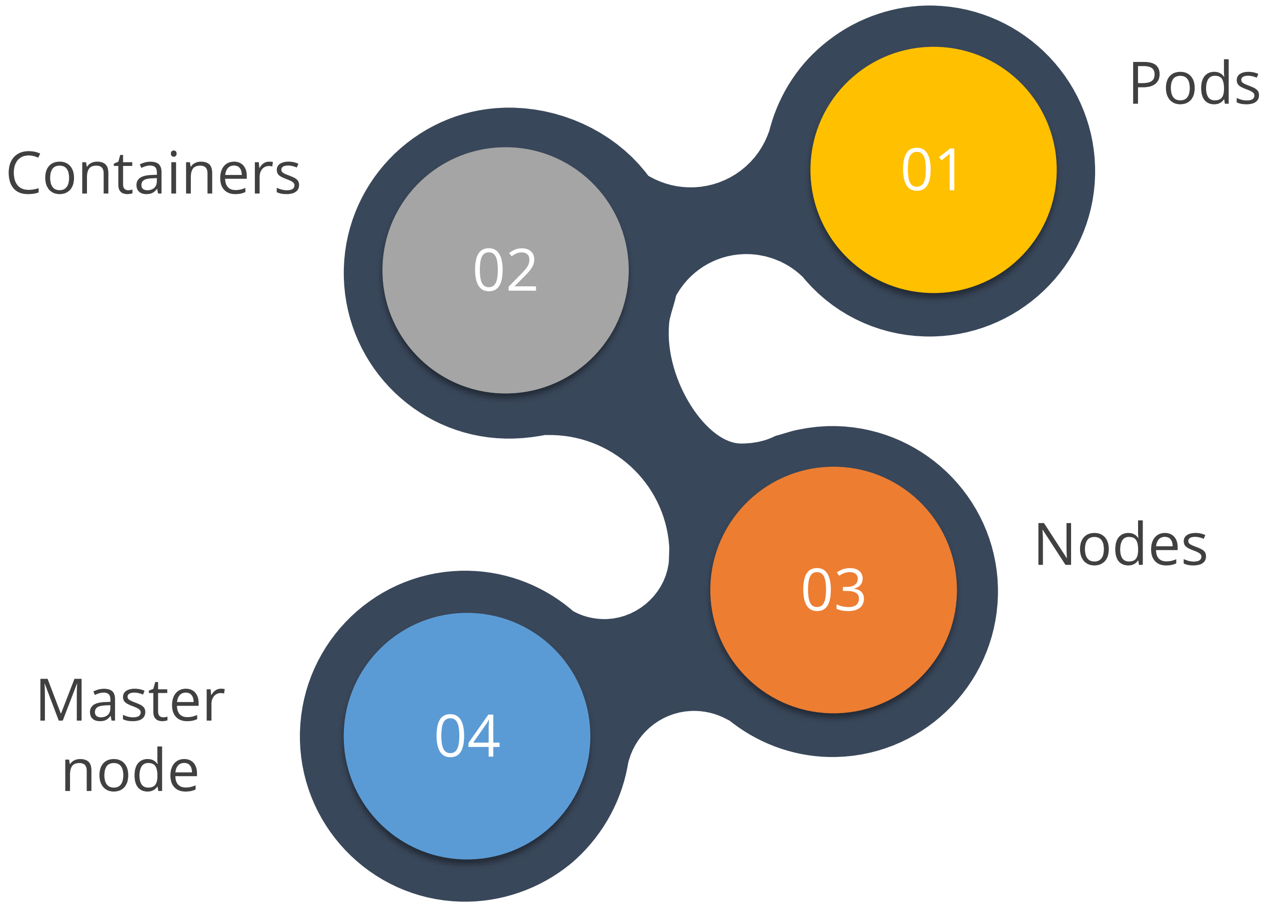
Cluster networking facilitates communication between various pods.

The Service resource lets exposing an application running in pods to be accessible from outside the cluster.

Services are used to publish services meant for consumption inside the cluster only.

Kubernetes Pod Network

A Kubernetes pod network connects several interrelated components including:



Networking in Kubernetes

Traffic that flows between nodes can also flow to and from nodes and an external physical machine or a VM.

There are four ways of getting external traffic into a Kubernetes cluster:

- LoadBalancer
- ClusterIP
- NodePort
- Ingress

Using Basic Commands of Kubernetes

Duration: 15 mins

ASSISTED PRACTICE

Problem Statement:

You have been asked to execute the basic commands used in Kubernetes.

Assisted Practice: Guidelines

Steps to be followed:

1. Create the deployment
2. Create the namespaces
3. Scale and delete the deployment

TECHNOLOGY

Policies

Overview

Policies define what end users can do on the cluster and possible ways to ensure that clusters comply.

Policies are applicable to network, volume, resource usage, resource consumption, access control, and security.

A constraint is a declaration that expects a system to meet a set of requirements.

Policy enablement helps organizations take control of Kubernetes operations.

Key Benefits of Policies

Simplified operations

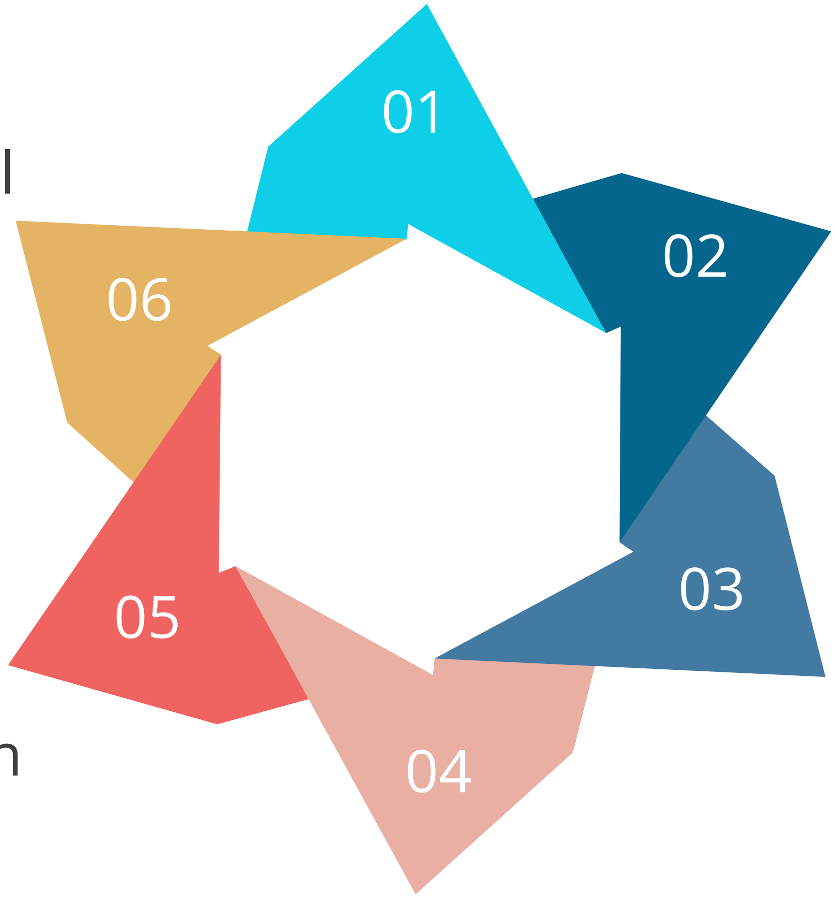
Ease of policy enforcement

Automated discovery of violations and conflicts

Better flexibility to changing requirements

Policy Restrictions

On a Kubernetes cluster, containers run with unbounded compute resources by default. To limit or restrict, appropriate policies must be implemented in the following ways:



- Security policies
- Network policies
- Volume policies
- Resource usage policies
- Resource consumption policies
- Access control policies

Key Takeaways

- Containers are lightweight, standalone, executable software packages that include everything required to run an application: code, runtime, system tools, system libraries, and settings.
- Etcd is a reliable and highly available key-value store that serves as the backup store for all cluster data in Kubernetes.
- Kube-proxy is a network proxy that runs on every node in a cluster, implementing the Kubernetes Service concept.
- Policies define what end users can do on the cluster and ways to ensure that clusters comply.

Fetching Cluster Specific Configuration

Duration: 25 Min.

Description: Your team lead has asked you to access the Kubernetes cluster and report the following cluster-specific details:

- Available nodes and their IP addresses
- Supported API versions on the server
- Status of the control plane and CoreDNS
- Status of pods with the kube-system namespace

Steps to Perform:

1. List the available nodes and their IP addresses
2. Identify supported API versions
3. Examine the control plane and CoreDNS status
4. Review the status of the pods in the kube-system namespace

LESSON-END PROJECT

TECHNOLOGY

Thank You

simplilearn