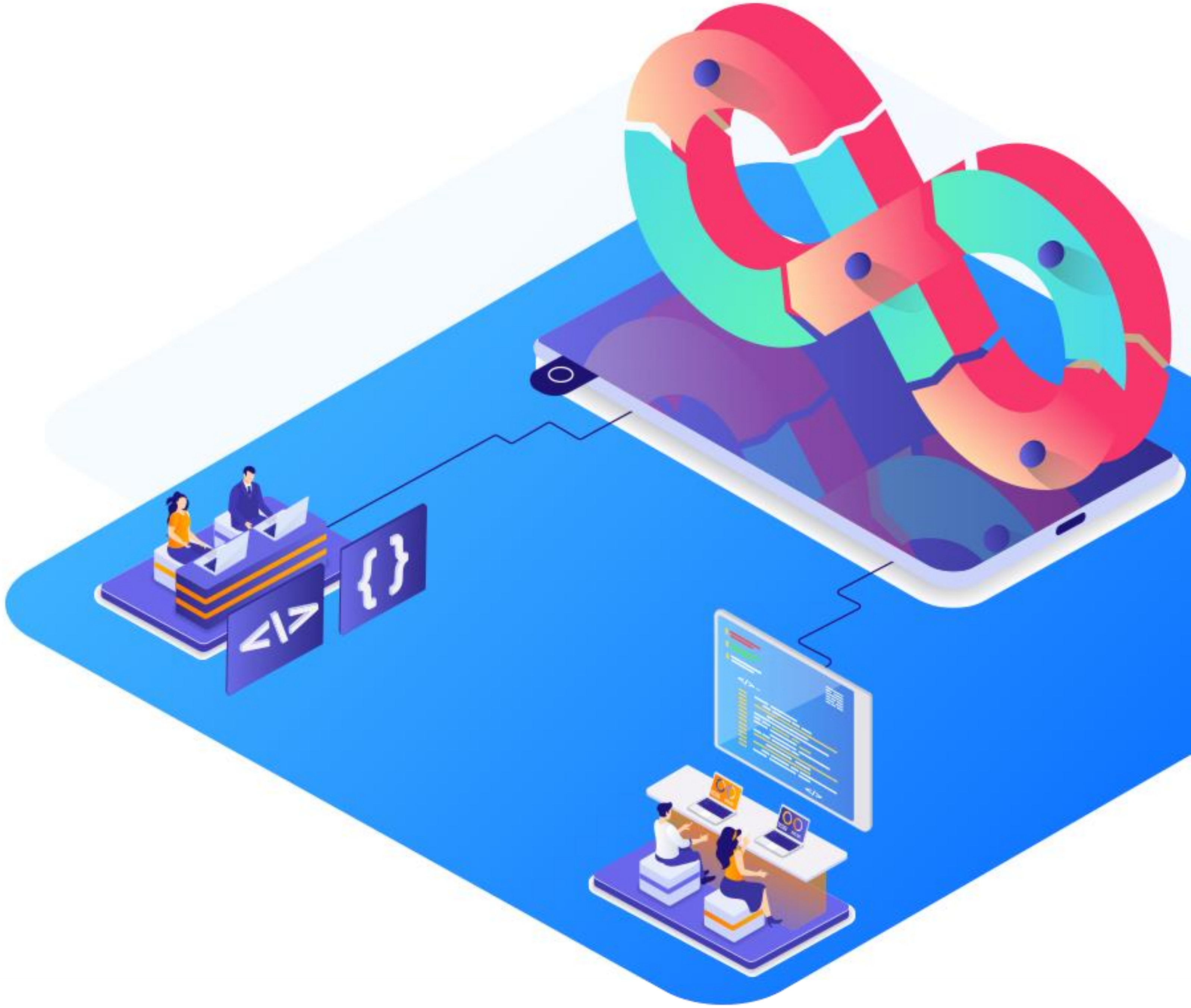
Container Orchestration Using Kubernetes



Core Concepts of Kubernetes

Learning Objectives

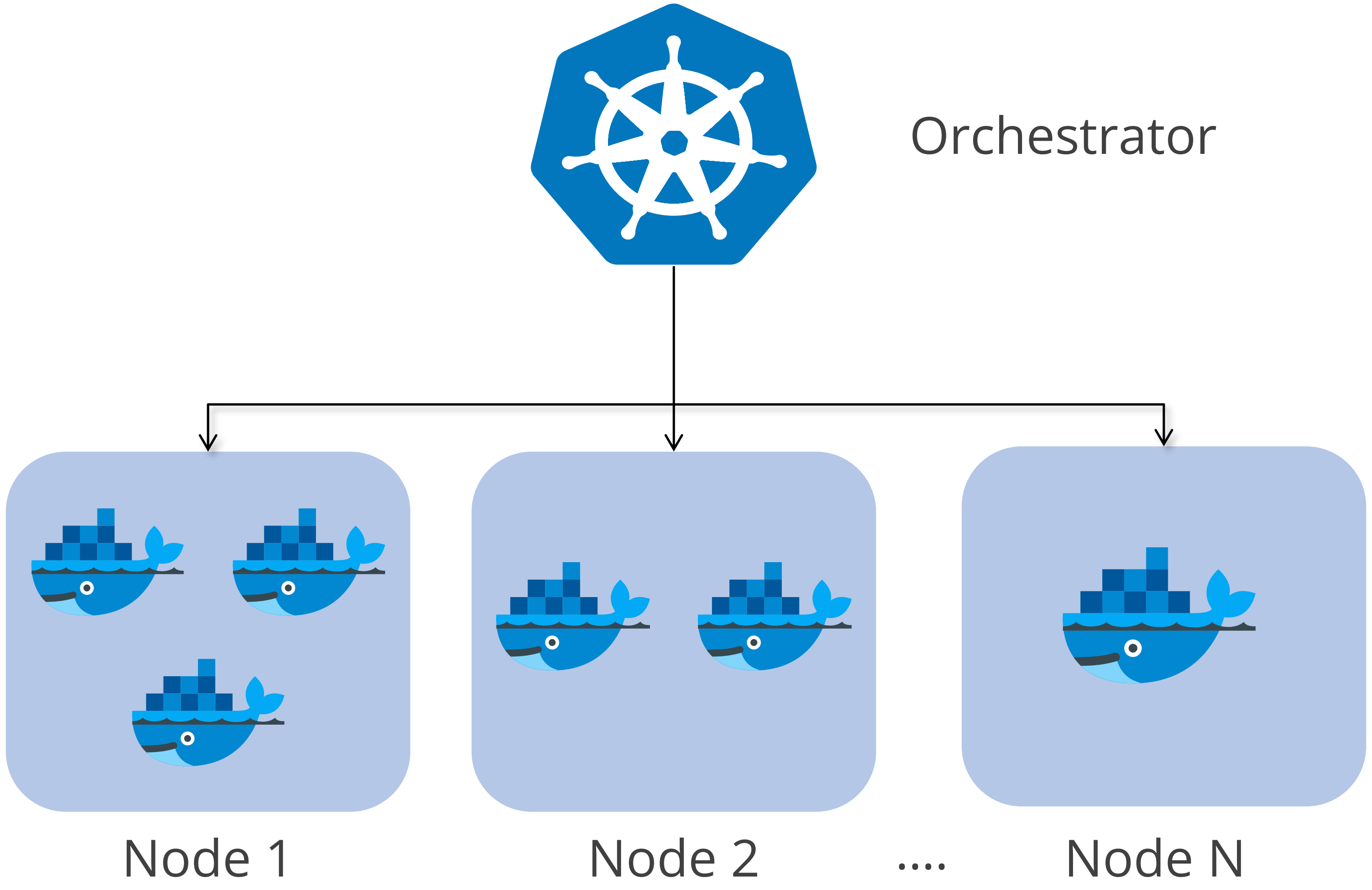
By the end of this lesson, you will be able to:

- Analyze container orchestration for managing and scaling applications
- Identify the core components of Kubernetes for orchestrating containers
- Analyze the Kubernetes scheduler's process of assigning pods to nodes based on resource requirements
- Identify the responsibilities of the Kubelet in managing pod for ensuring proper container execution
- Analyze the role of Deployments in simplifying rolling updates and rollback processes to maintain application stability and minimize downtime

Overview of Container Orchestration

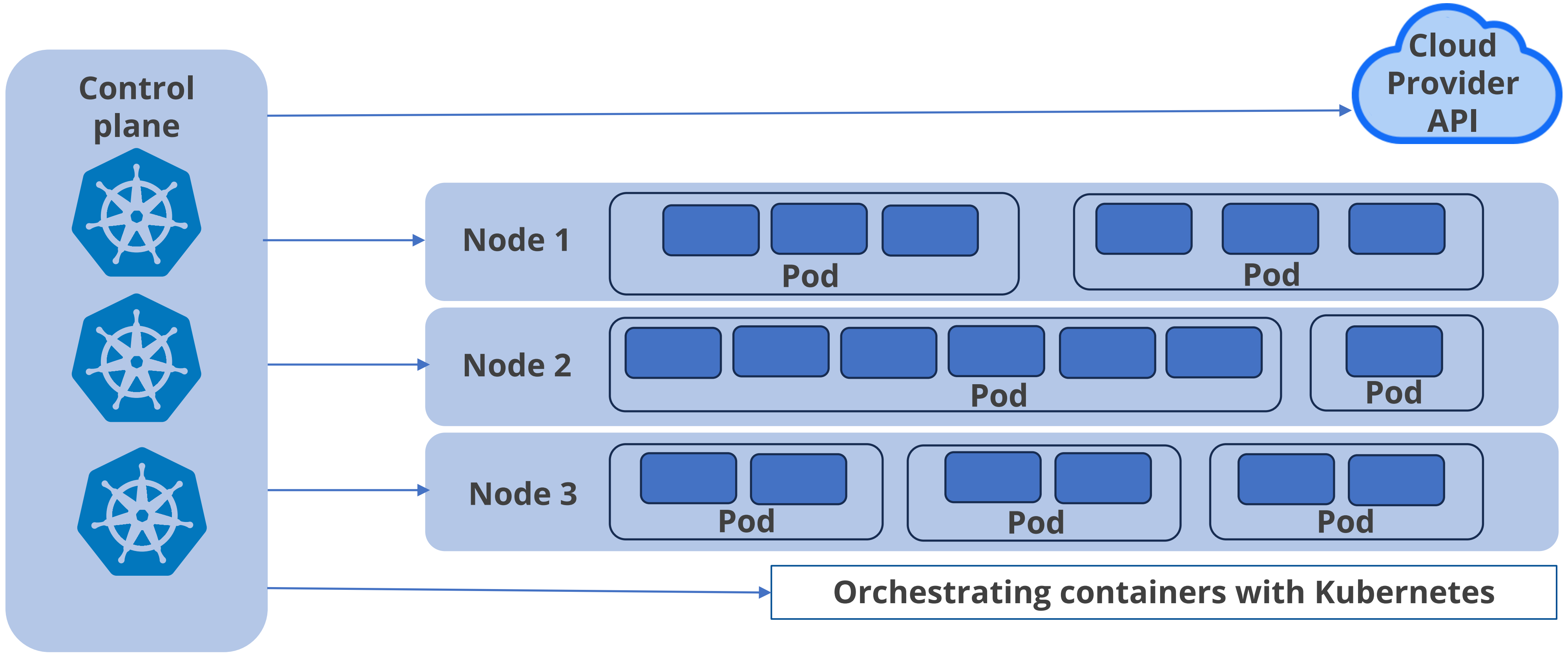
Container Orchestration

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



Container Orchestration Using Kubernetes

It enables organizations to efficiently orchestrate containers across clusters of machines, ensuring applications run seamlessly, scale automatically, and recover from failures.



Orchestrating containers with Kubernetes

Where Is Container Orchestration Used?

It is widely used in several areas to streamline operations and improve efficiency. Some of them are:

Microservices architecture

Manages and scales microservices, enabling independent build, deploy, and update processes

Cloud-native applications

Automates tasks like scheduling, networking, and load balancing in cloud environments

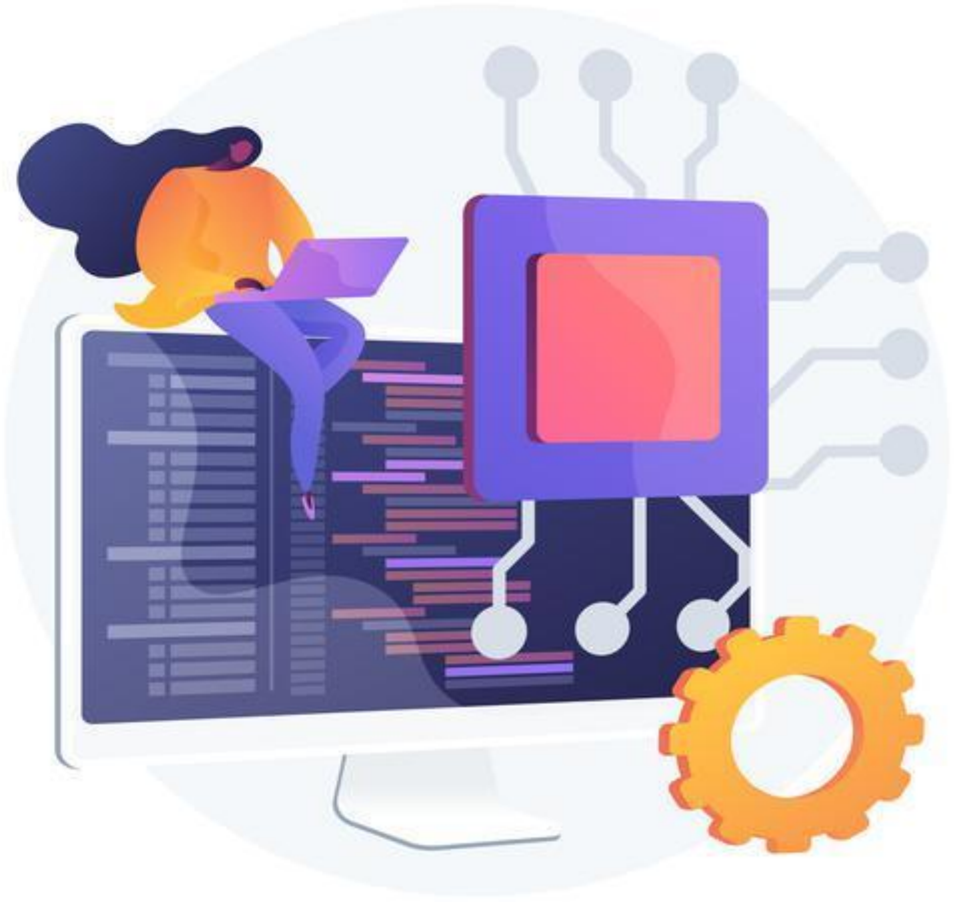
DevOps automation

Supports CI/CD by automating testing, scaling, and updating of containerized applications

Overview of Kubernetes

Kubernetes

It is an open-source platform for automating the deployment, scaling, and operations of application containers across clusters of hosts.



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.

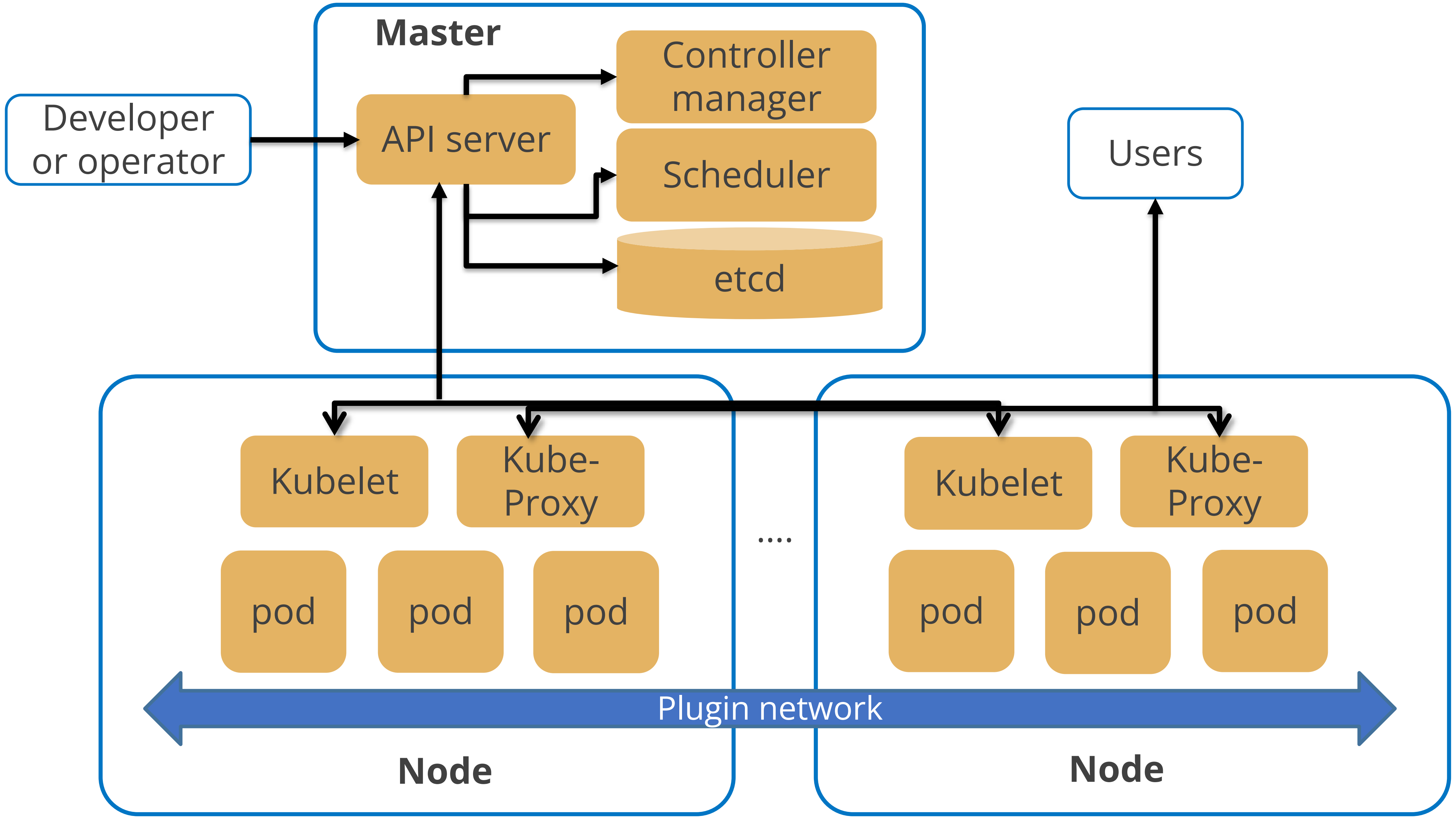
Features of Kubernetes

Kubernetes offers the following features to its users:

- Storage
- Secure configuration management
- Autoscaling
- Automated rollouts and rollbacks
- Service discovery and load balancing

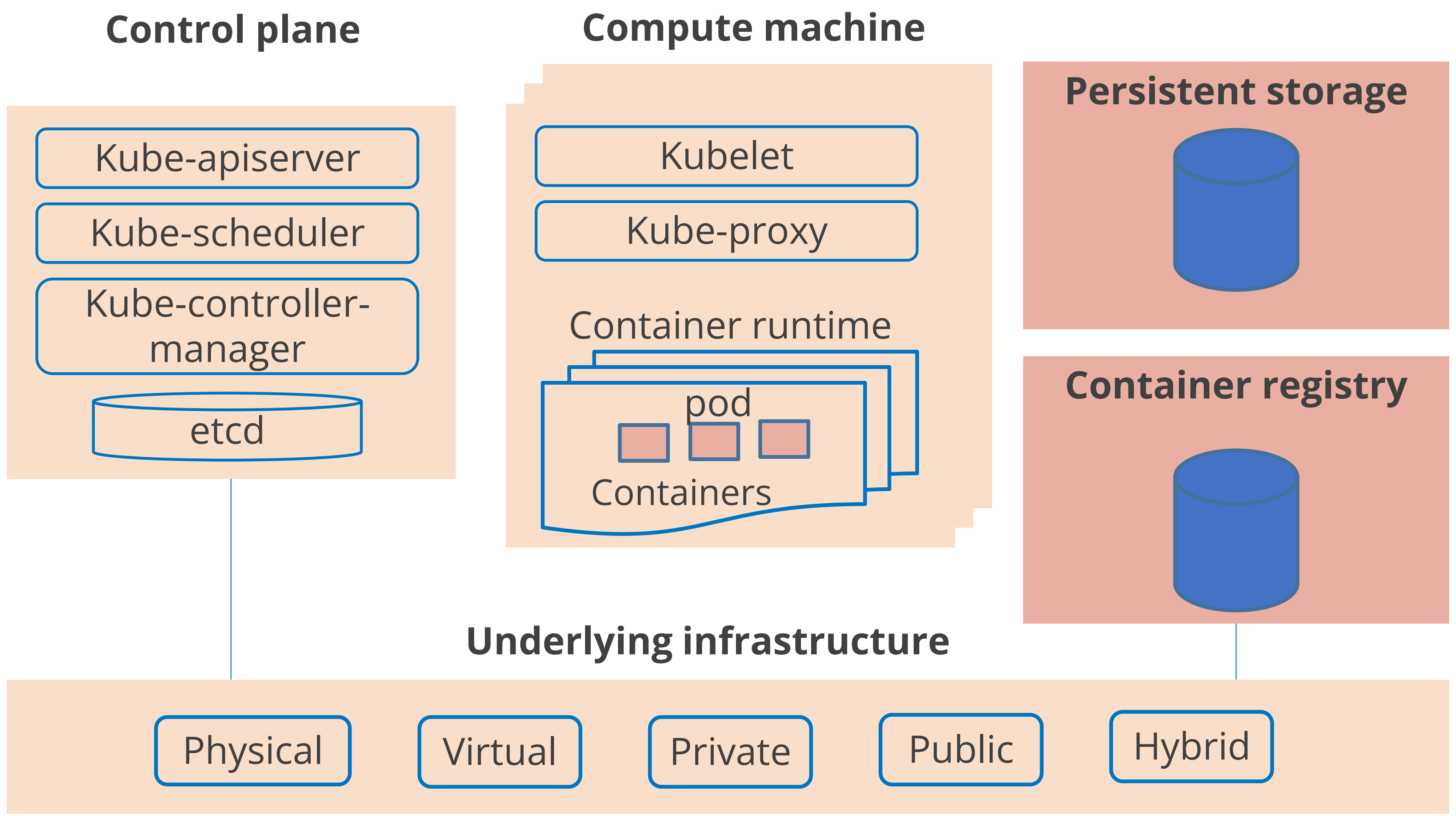
Kubernetes Architecture

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



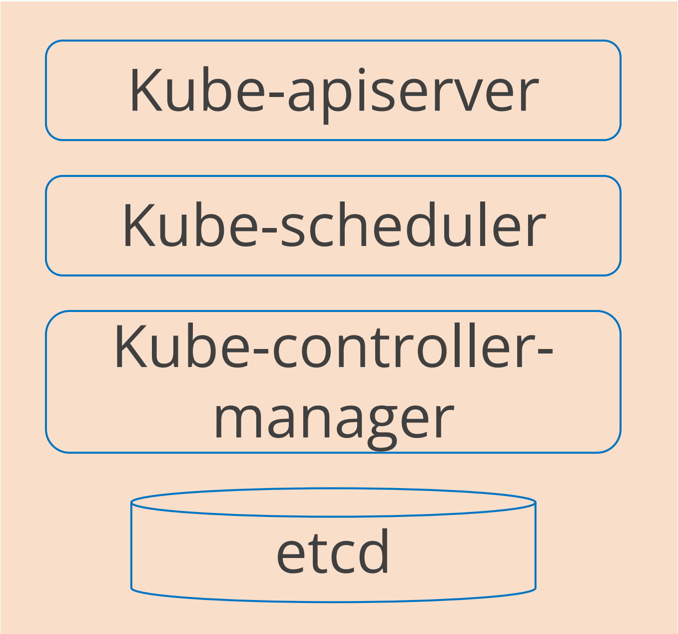
Kubernetes Cluster

It consists of a master node and a set of worker nodes that work together to automate the deployment, scaling, and management of containerized applications.



Control Plane (Master Node)

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.

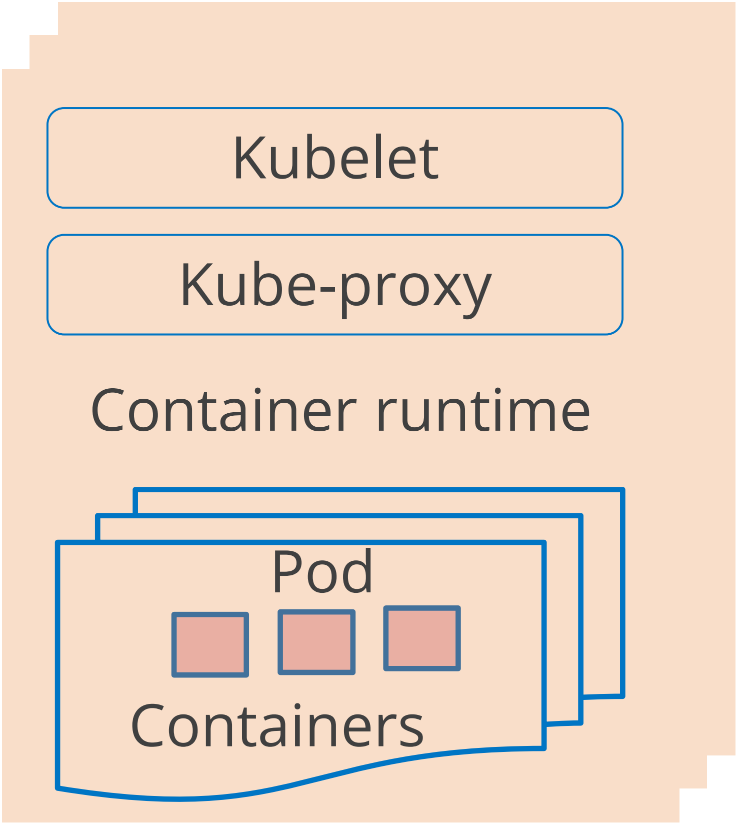
Kube-scheduler schedules the pod to an appropriate node.

Kube-controller-manager helps run the cluster.

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

Compute Machine (Worker Node)

A Kubernetes cluster relies on at least one worker node (compute machine), where pods are scheduled and containers are managed.



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.

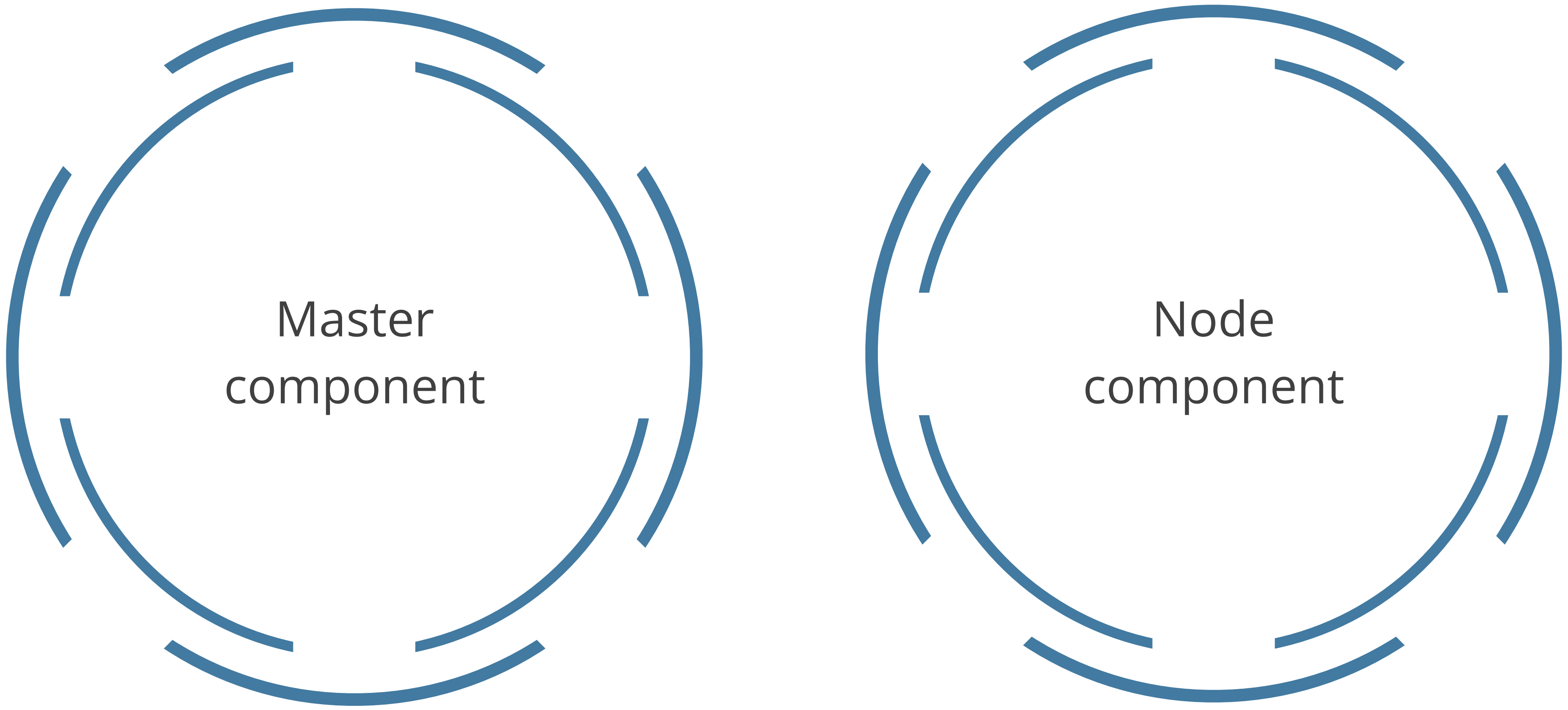
Kubernetes Aspects

The aspects of Kubernetes are as follows:

Kubernetes Components

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

They are divided into:



Master Component

They manage the entire cluster, handling the coordination for deploying, scaling, and maintaining applications and workloads.

Master components include:

- Kube-apiserver
- etcd
- Kube-Scheduler
- Kube Controller Manager
- Cloud-Controller Manager

The default config file is **/etc/kubernetes/admin.conf**.

Node Component

They are responsible for running workloads (containers) in the cluster. Each node includes the necessary services and components to manage and execute pods.

Node components include:

- kubelet
- kube-proxy
- Container runtime

Node components work together to ensure that the applications deployed in the Kubernetes cluster run efficiently and can communicate properly across the cluster.

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:

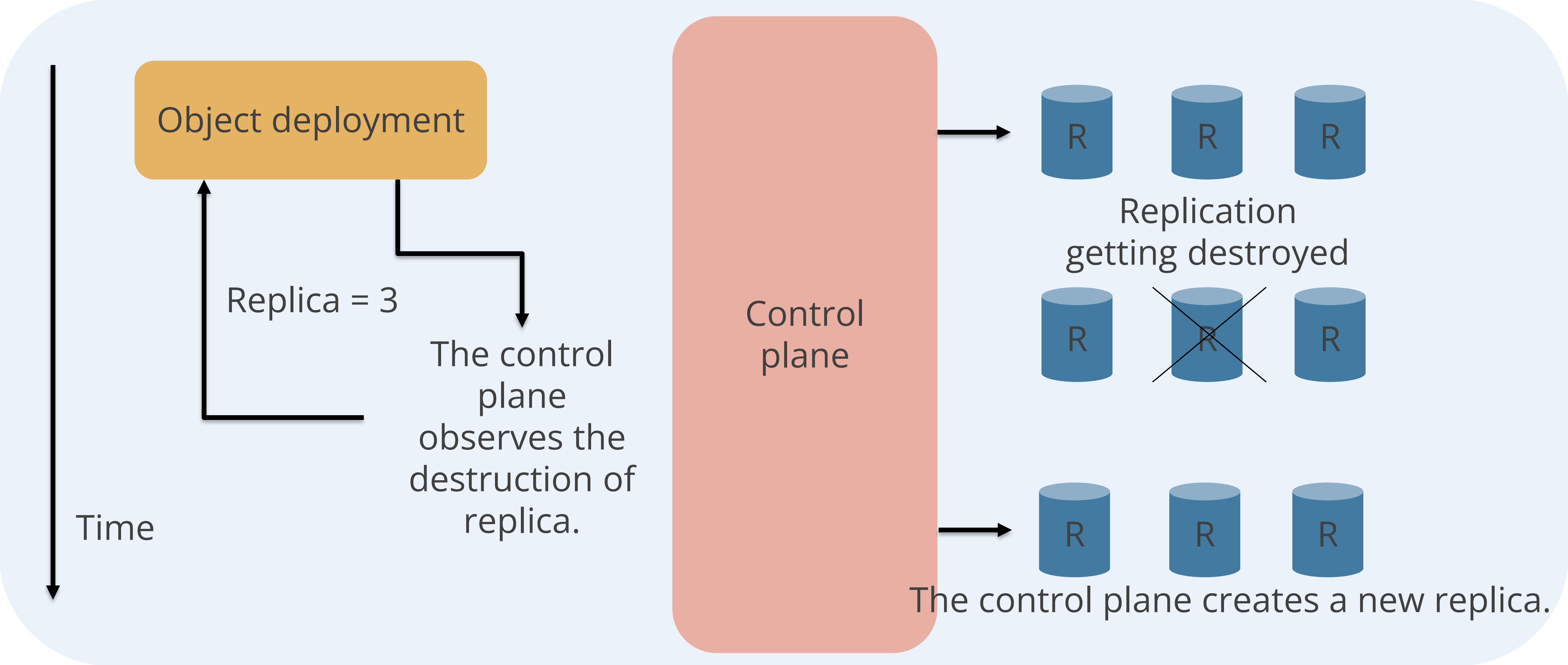
Kubernetes objects define what containerized applications are currently running.

These objects also specify the resources that are available to those applications.

They include policies governing how applications behave, such as restart policies, upgrade procedures, and fault tolerance measures.

Object Fields

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



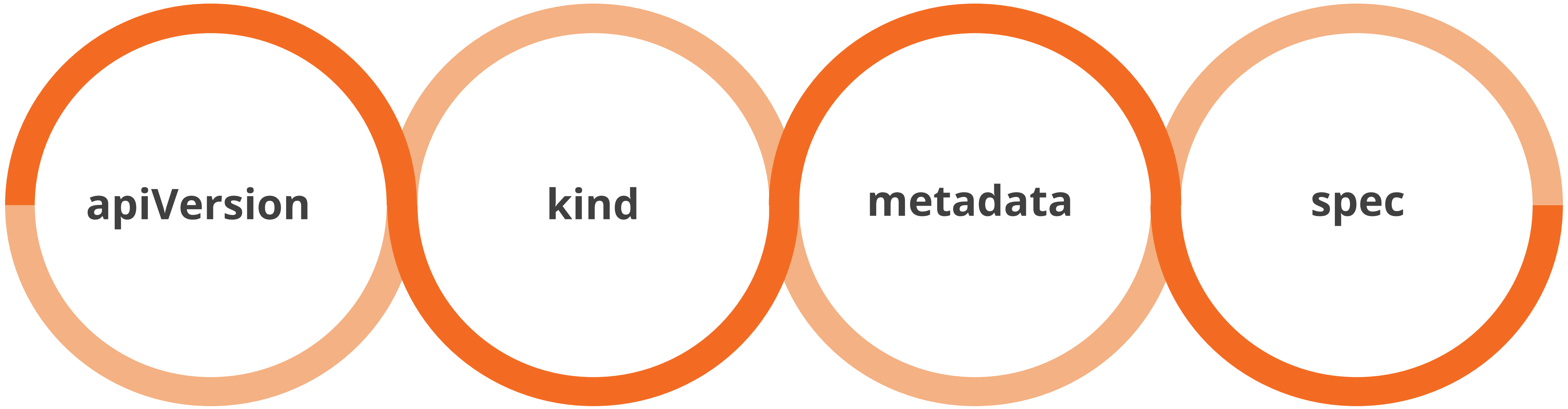
Describing Kubernetes Object

When creating a Kubernetes object, the user must provide the object specification, which defines the desired state of the object and includes essential information like its 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:



Business Benefits of Using Kubernetes

1) Scalability

Enhances scalability by allowing businesses to efficiently manage and scale applications without manual intervention

2) Cost efficiency

Reduces infrastructure costs through optimized resource allocation and dynamic scaling based on demand

3) Faster deployments

Increases deployment speed by automating the deployment and management of containerized applications across different environments

Business Benefits of Using Kubernetes

4) Operational efficiency

Improves operational efficiency by streamlining application maintenance, updates, and rollbacks with minimal downtime

5) Multi-Cloud Flexibility

Supports multi-cloud environments, enabling businesses to run applications seamlessly across various cloud platforms or on-premises infrastructure

6) Developer productivity

Boosts developer productivity by providing consistent environments and simplifying the process of managing microservices architecture

Organizations Using Kubernetes

Notable companies leveraging Kubernetes are as follows:

Google, the creator of Kubernetes, uses it extensively for managing services and applications on Google Cloud Platform (GCP). Kubernetes is key to scalable and efficient infrastructure management at Google.

IBM utilizes Kubernetes across its cloud and AI solutions for efficient application management.

Pinterest employs Kubernetes to manage infrastructure, ensuring performance and scalability for its platform.

Airbnb relies on Kubernetes to manage its containerized applications, which helps the company scale its services efficiently and maintain high availability for its global user base.

Quick Check

You are managing a Kubernetes cluster and need to know which component handles API requests. Which of the following is the control plane component that manages internal and external requests?

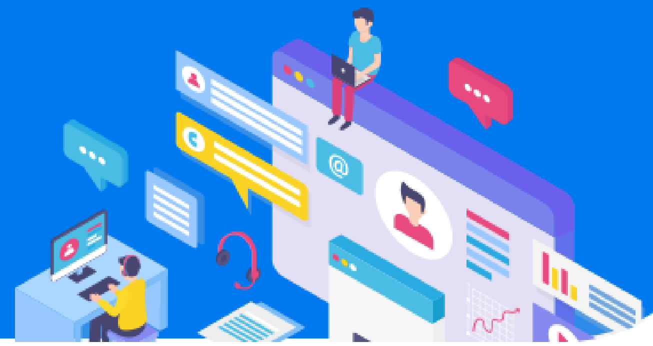
A. Kube-scheduler

B. Kube-apiserver

C. Kube-controller-manager

D. etcd

Assisted Practice



Creating and Configuring a Kubernetes Cluster **Duration: 20 Min.**

Problem statement:
You have been asked to create a Kubernetes cluster and add the nodes to it for deployment and management of containerized applications.

Outcome:
By completing this demo, you will be able to set up a Kubernetes cluster and add nodes to it for managing containerized applications. This will enable you to orchestrate containers and ensure application scalability and reliability.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

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

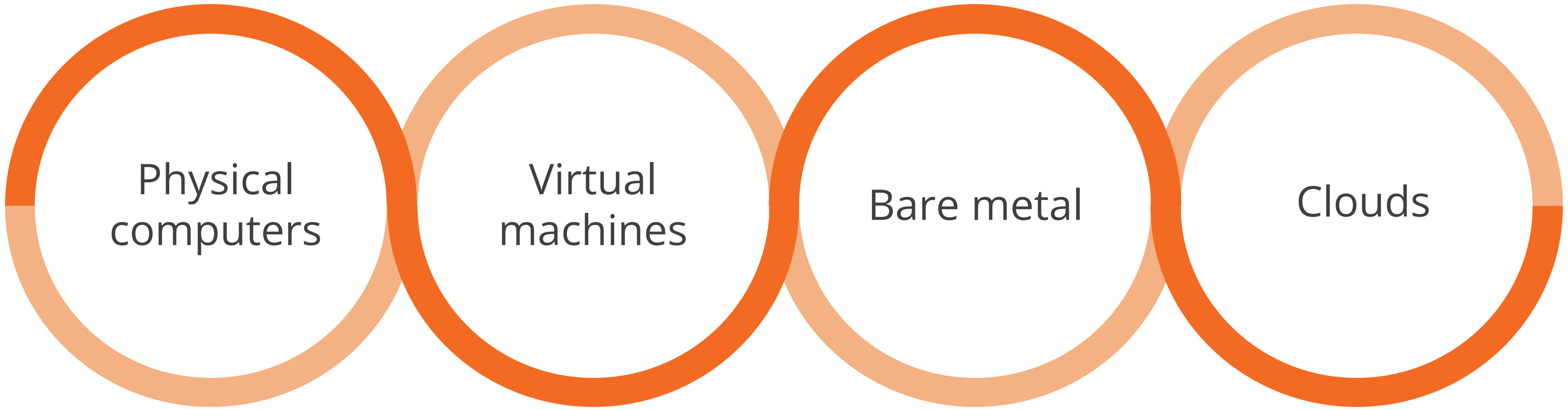
ASSISTED PRACTICE

Overview of Container

Container

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

Containers can run efficiently in four different environments:



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

Container Image

It is a software package that is ready to use and contains everything needed to run an application, such as:

- 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

Lightweight

Containers consume fewer resources than virtual machines, making them highly efficient.

Scalable

It can be quickly replicated to handle increased workloads, offering seamless scalability.

Portable and consistent

It ensure applications run consistently across different environments, enhancing portability.

Agile

It enable faster updates and deployments, supporting rapid development cycles.

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.

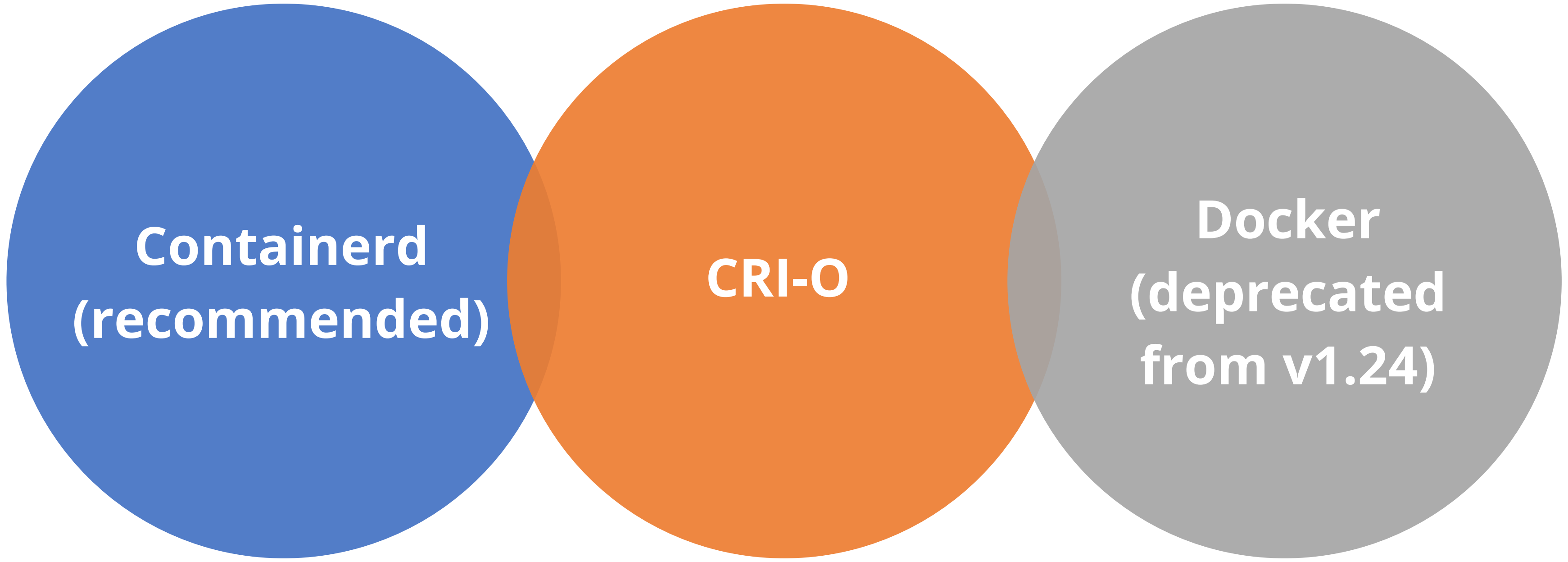
Control groups implement resource accounting and resource limitation.

Union file systems prevent data duplication.

Container Runtime

It runs on Kubernetes worker nodes. It pulls container images from a registry, starts and stops containers, and manages their lifecycle on the node.

Commonly used container runtimes with Kubernetes are:

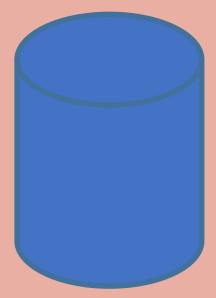


Container runtime interface (CRI) is an API for container runtime to integrate with Kubelet.

Storage and Registry

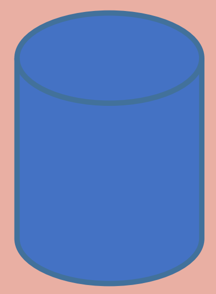
In a Kubernetes cluster, persistent storage and container registries are essential components that manage application data and store container images for deployment.

Persistent storage



A persistent volume or storage is a piece of storage in a cluster that an administrator has provisioned.

Container registry



A container registry stores and shares container images by allowing developers to push images to the registry and pull them into another system.

Quick Check

You are deploying an application on Kubernetes and need to choose the right container image. The image must include everything required for the application to run properly in any environment. Which image option should you select?

A. Image with only the application code

B. Image with configuration files only

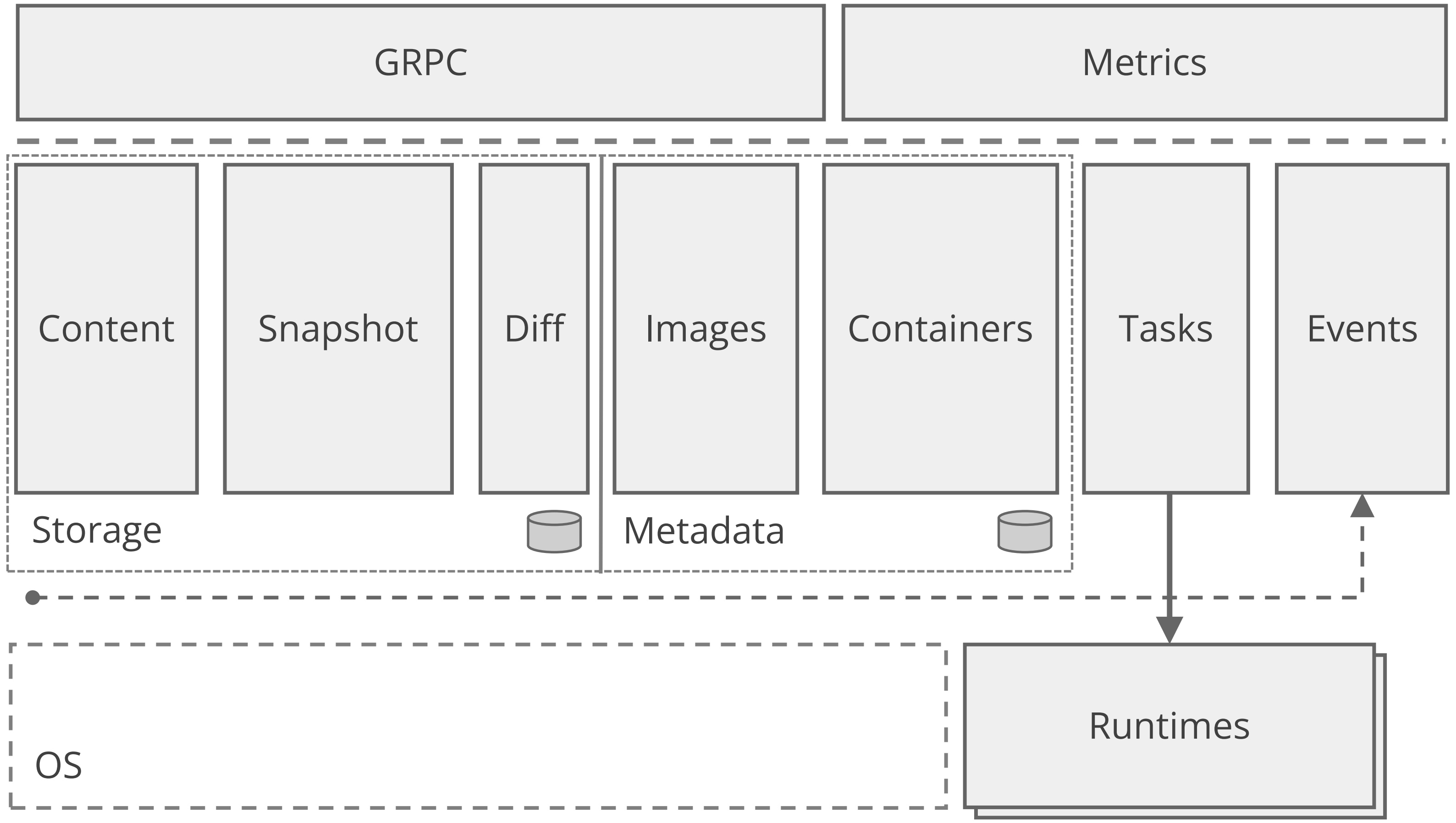
C. Image with just the libraries

D. Image with OS, libraries, and binaries

Overview of Containerd

Containerd

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

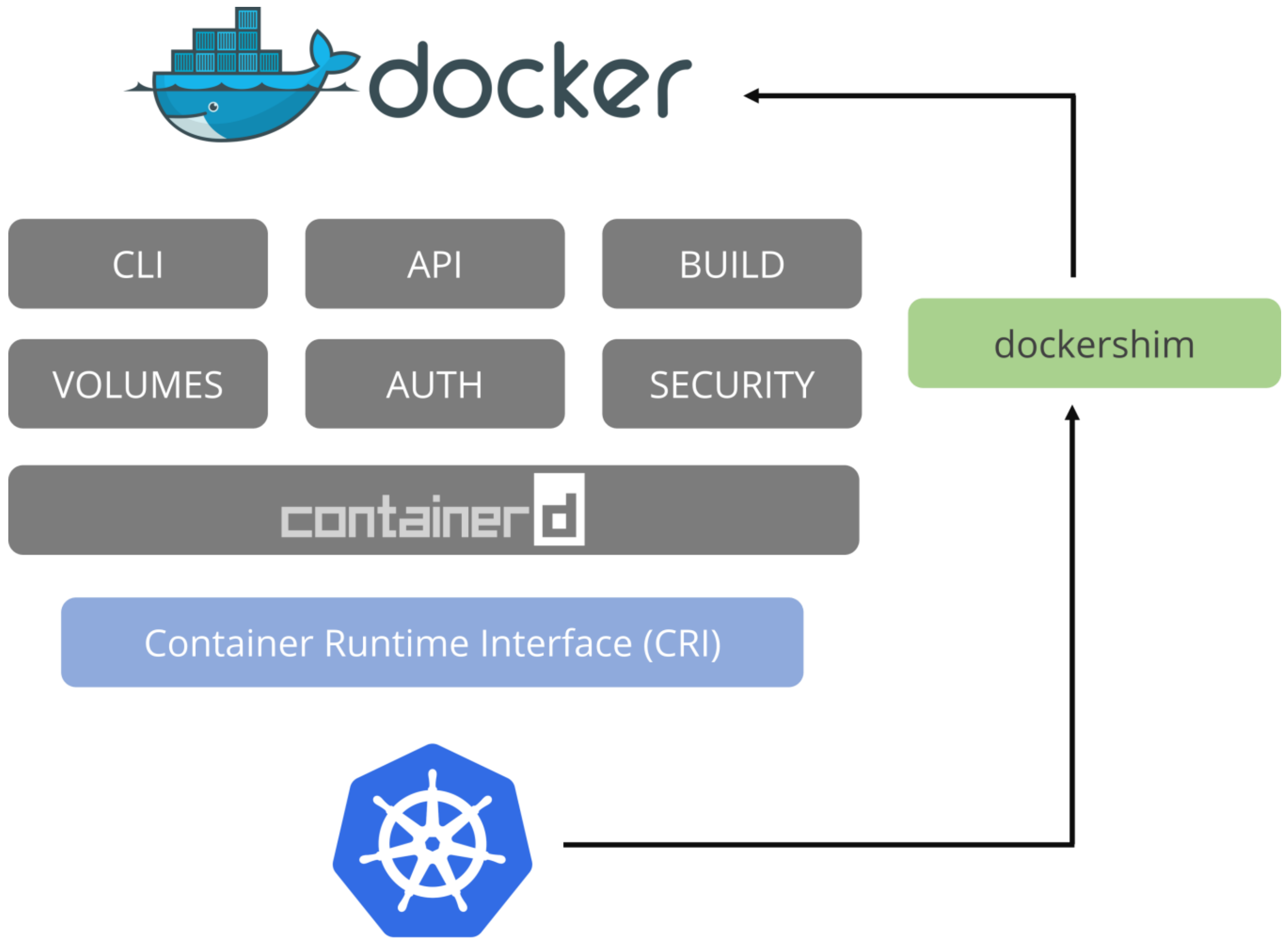


It serves as a Linux daemon capable of managing the entire container life cycle of its host system.

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

Transition from Docker to Containerd in Kubernetes

With Kubernetes v1.24, the direct integration with Docker via Dockershim was removed, transitioning Kubernetes to use containerd for container runtime management.

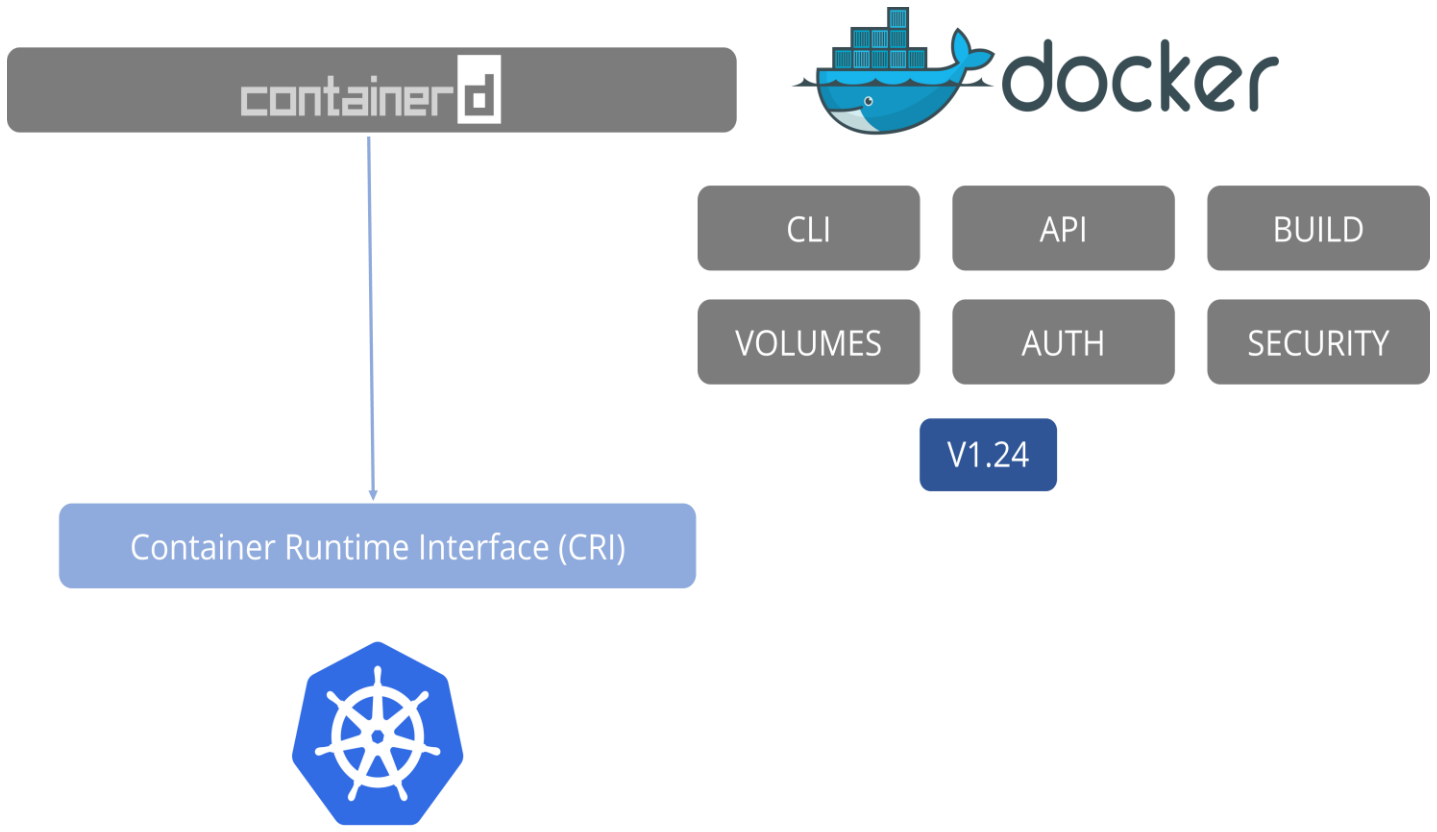


Before v1.24, Kubernetes relied on Dockershim to integrate with Docker Engine, allowing Docker to be used as the container runtime.

After v1.24, Kubernetes replaced Docker with containerd, a lightweight, CRI-compatible runtime for improved performance and simpler container management.

Transition from Docker to Containerd in Kubernetes

With Kubernetes v1.24, the shift from Docker to containerd streamlined container management, enhancing performance and compatibility with Kubernetes.

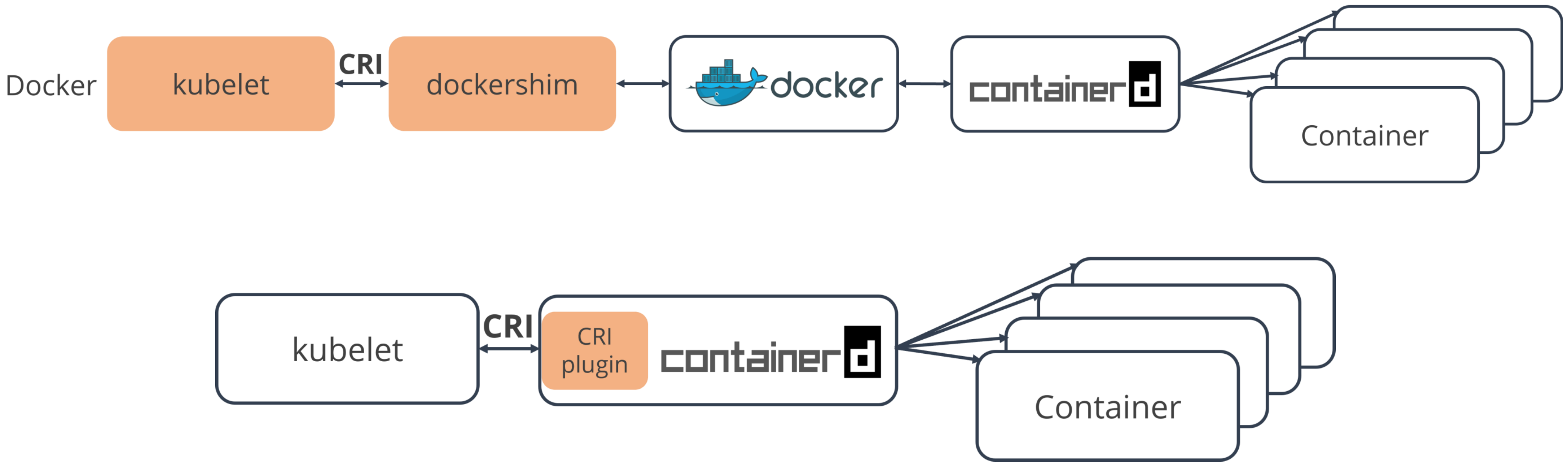


Containerd is CRI-compatible and can directly integrate with Kubernetes, like all other runtimes. Consequently, one can use containerd as a standalone runtime, separate from Docker.

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

Transition from Docker to Containerd in Kubernetes

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 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 users to list pods by executing the crictl pods command, a capability Docker did not have.

crictl

It is a command-line interface for container runtimes that are compatible with CRI. It allows users 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: Commands

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: Commands

The following commands are used to perform changes related to container life cycle management, such as creating, stopping, pulling images, and removing containers:

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: Commands

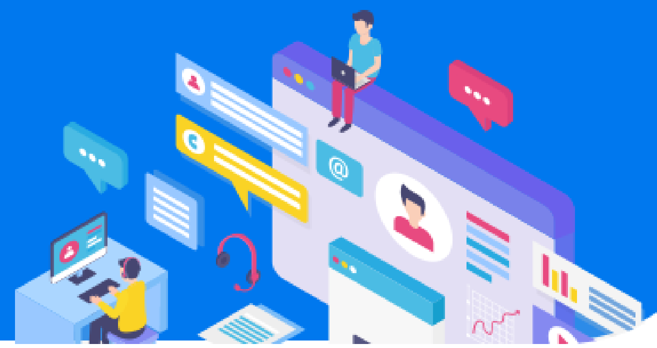
docker cli	crictl	Description	Unsupported features
rmi	rmi	Remove one or more images	
run	run	Run a command in a new container	
start	start	Start one or more stopped containers	--detach-keys
stop	stop	Stop one or more running containers	
update	update	Update configuration of one or more containers	--restart,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

crictl: Commands

The following commands are only supported in crictl:

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

Assisted Practice



Demonstrating crictl Commands **Duration: 20 Min.**

Problem statement:
You have been asked to debug Kubernetes nodes with crictl CLI commands for identifying and resolving issues in container runtimes and applications.

Outcome:
By completing this demo, you will be able to use crictl commands to inspect, manage, and debug Kubernetes nodes, enhancing your troubleshooting skills and ensuring cluster stability.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Configure and manage container runtime environment

ASSISTED PRACTICE

Quick Check

You need a lightweight, portable runtime for managing container life cycle tasks like image storage and execution that integrates with Kubernetes. Which runtime is the best fit for this?

A. Containerd

B. Docker CLI

C. Kubelet

D. Hypervisor

Overview of etcd

etcd

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

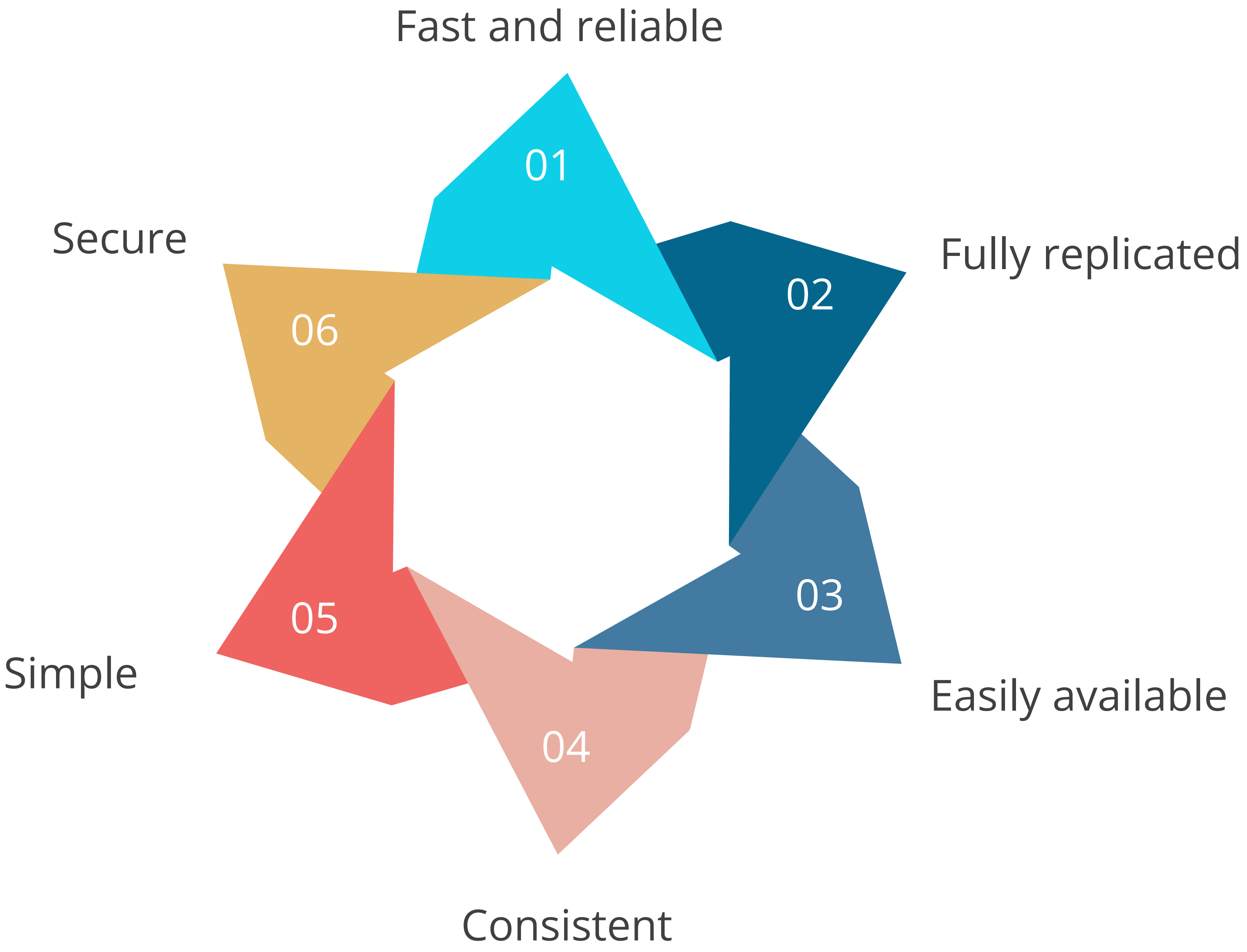
- Distributed key-value store that manages critical information
- Used in production environment by cloud service providers
- Used as a distributed datastore
- Used to manage configuration data, state data, and metadata

etcd : Key Concepts

etcd operates on three core concepts in its Raft-based system:

Leaders	Handle all client requests that require consensus across the cluster
Elections	Occur when the leader becomes unavailable, ensuring a new leader is chosen
Terms	Represent leadership periods, with new terms starting when elections are held

etcd: Features



Quick Check

You are managing a distributed system using the Raft consensus algorithm and need to ensure client requests are handled with leader election in case of failure. Which key etcd concept handles this?

A. Sharding

B. Followers

C. Nodes

D. Leaders

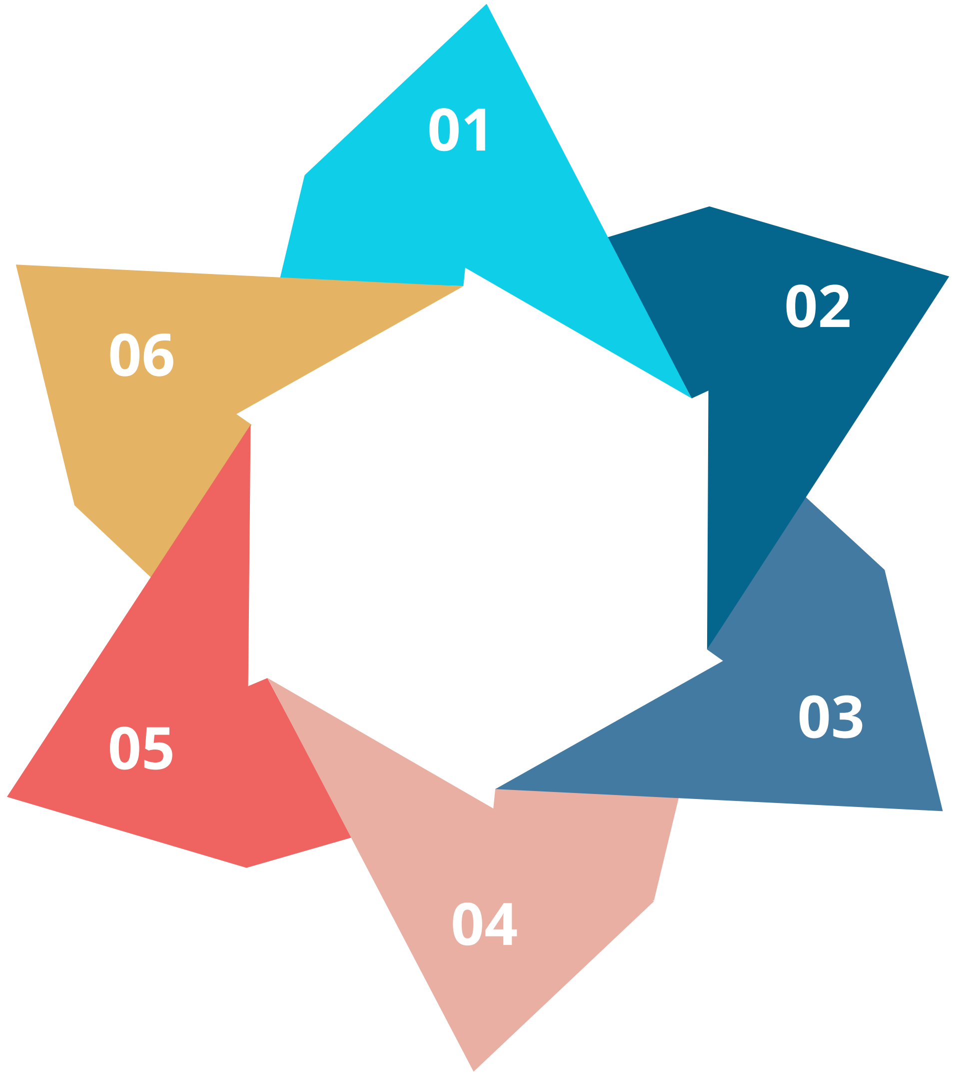
Controller

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

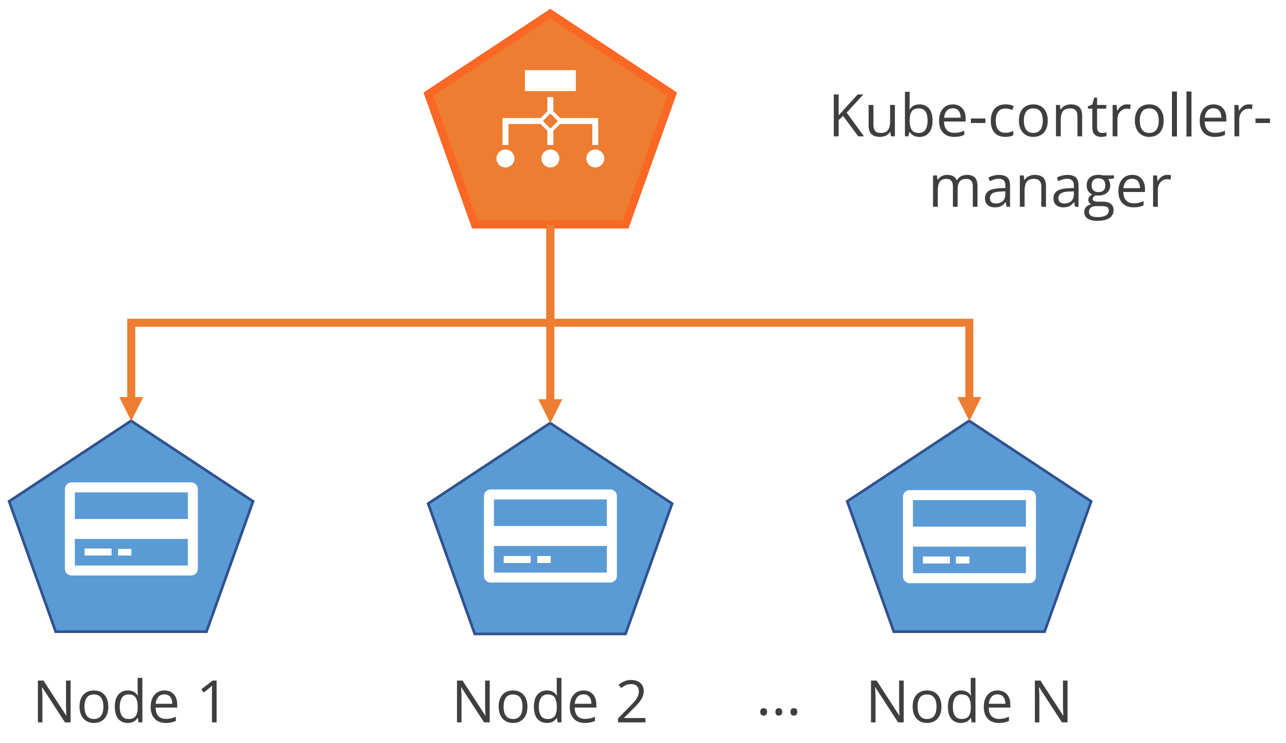
StatefulSet: Manages stateful applications

Job: Creates one or more short-lived pods

CronJob: Creates a job on schedule

Role of Controllers in Kubernetes

The Kube-controller-manager runs built-in controllers that manage key Kubernetes functions like node health, pod management, and resource allocation across all nodes.



Additional controllers can operate outside the control plane to extend Kubernetes functionality and ensure more scalable and customizable operations.

Quick Check

You need to run a pod on every Kubernetes node for logging and ensure it automatically updates as nodes are added or removed. Which Kubernetes controller will help achieve this?

A. Deployment

B. ReplicaSet

C. DaemonSet

D. CronJob

Scheduler

Kubernetes Scheduler

It 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 in 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 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

Kubernetes uses node affinity and anti-affinity to control where pods are placed by specifying preferred or excluded nodes based on defined rules.

Node affinity allows flexible pod placement by specifying preferred nodes based on certain conditions defined in the YAML configuration.

Node anti-affinity ensures flexible decision-making by avoiding the placement of pods on certain nodes, based on specific criteria.

Quick Check

You are deploying pods in a Kubernetes cluster and need them to be automatically assigned to the best nodes based on available resources. Which Kubernetes component handles this task?

A. Kubelet

B. DaemonSet

C. Kube-scheduler

D. Deployment

Kubelet

What Is 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

In addition to PodSpec, Kubelet can receive a container manifest through three other methods:

File

Pass the file path as a flag on the command line

HTTP endpoint

Pass the endpoint as a parameter on the command line

HTTP Server

Listen for HTTP requests and respond using a simple API

Kube-proxy

What Is Kube-proxy?

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

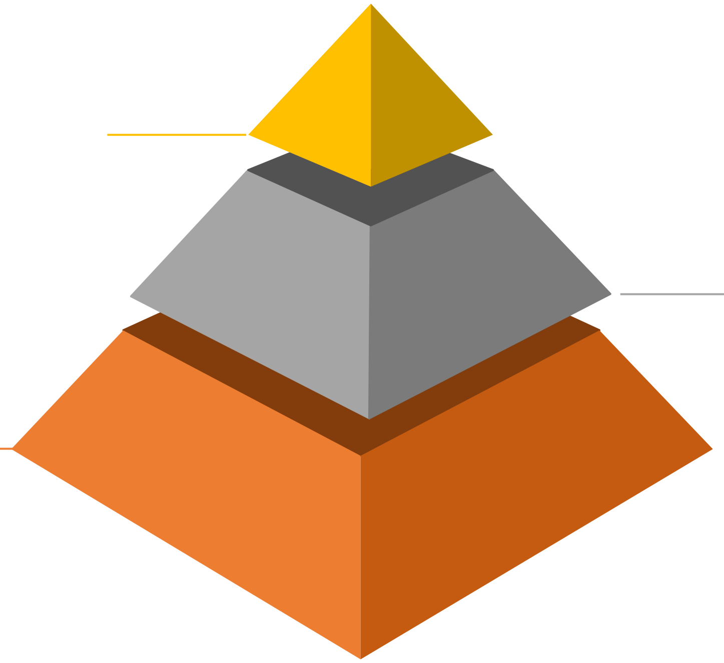
Maintains network rules on nodes

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

Kube-proxy: Operation Modes

It supports three different operation modes for managing service routing across nodes in Kubernetes:

User space
Handles service routing to backend pods



Iptables
Routes traffic as the default mode on all platforms

IPVS (IP Virtual server)
Manages traffic with advanced load balancing using Netfilter

Quick Check

You need a kube-proxy mode that supports service routing and efficient load balancing in your Kubernetes cluster. Which kube-proxy operation mode is best for this?

A. User space

B. IPVS

C. Iptables

D. TCPProxy

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

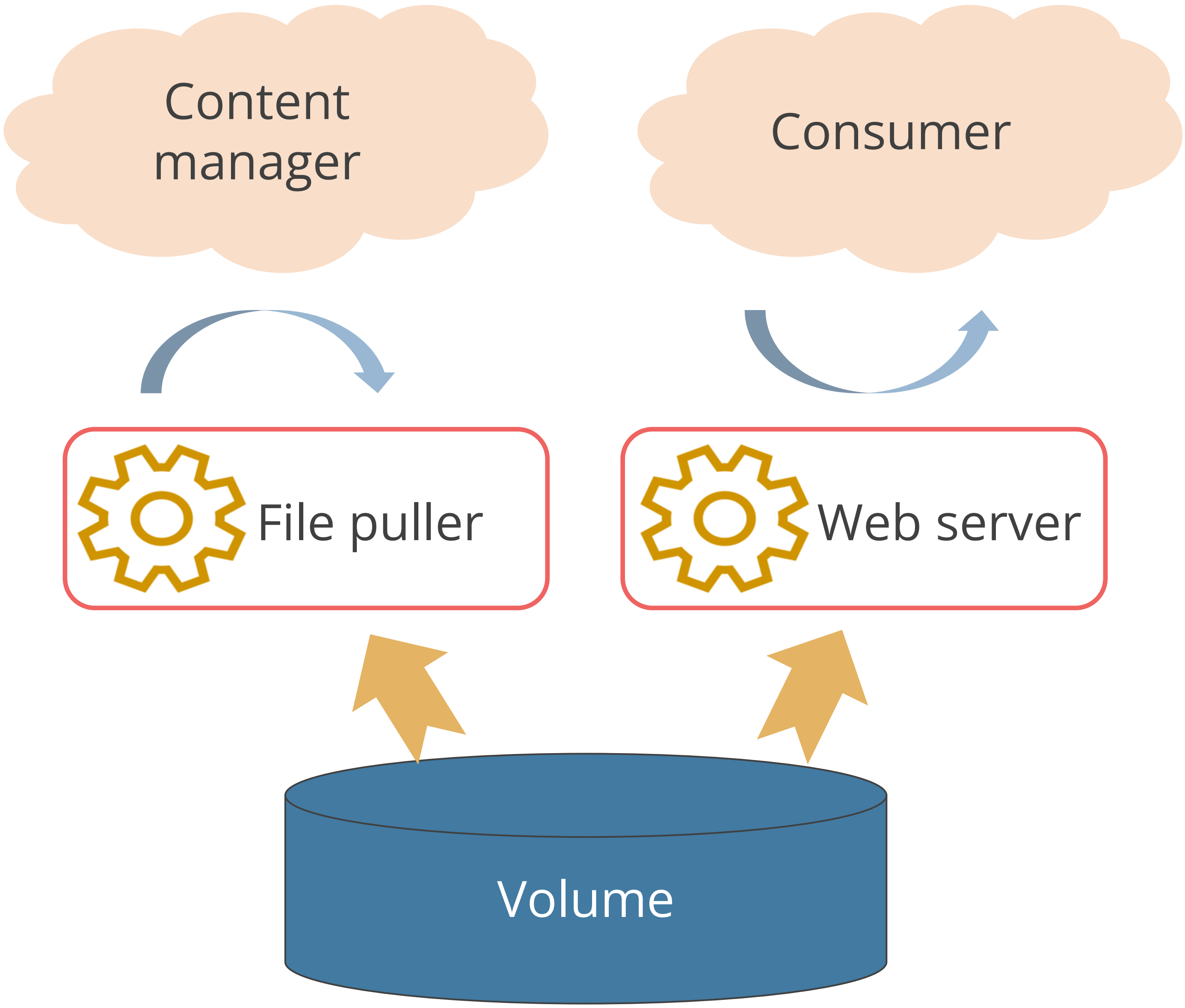
Pods provide shared networking and storage resources for the containers they host.

Network:
Each pod is automatically assigned a unique IP address.

Storage:
Each pod can define shared storage volumes accessible by its containers.

How Pods Manage Multiple Containers?

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



Multiple Pods: Workload Resources

Workload resources create and manage one or more pods.

Examples of workload resources:

Deployment

StatefulSet

DaemonSet

Pod Template

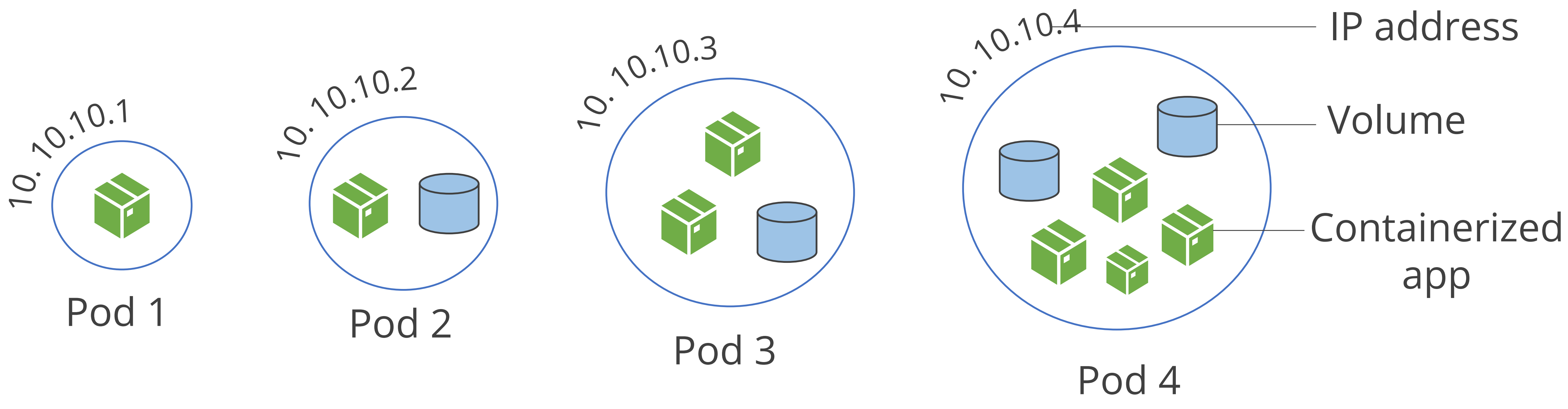
They are the specifications for creating pods and are included in workload resources, such as deployments, jobs, and DaemonSets.

Sample pod template:

```yaml
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: Limitations

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.

Assisted Practice

Configuring Pods in the Kubernetes Cluster **Duration: 15 Min.**

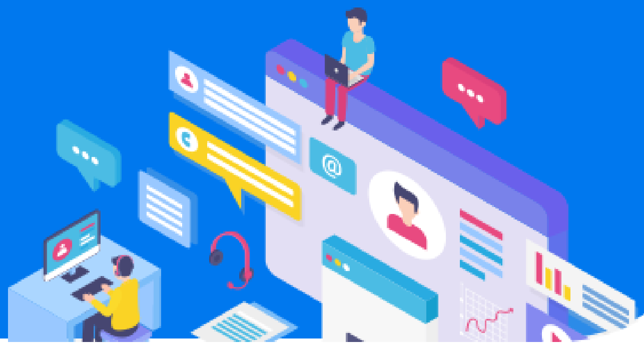
Problem statement:
You have been assigned a task to create, configure pods, and execute the Apache services for a web server deployment in a Kubernetes cluster.

Outcome:
By completing this demo, you will be able to deploy, configure, and manage Apache services in a Kubernetes cluster.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

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

Quick Check

You need to define specific configurations like the image, name, and ports for pods in a Kubernetes deployment. Which resource should you use to define these pod specifications?

A. Pod Template

B. ConfigMap

C. Service

D. Secret

ReplicaSets

ReplicaSets

It maintains the desired number of identical pod replicas running at any given time.

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

It ensures that a defined number of pod replicas are consistently running.

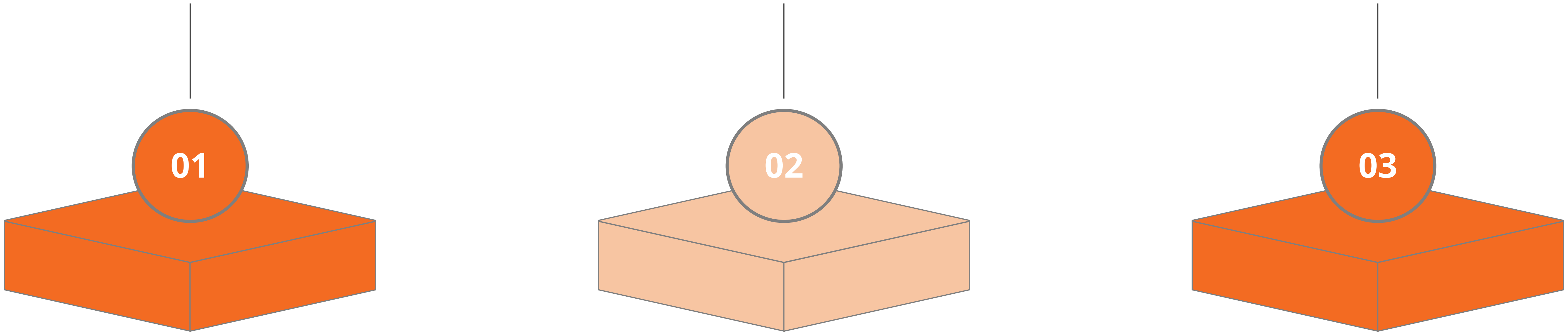
Operators to Use with ReplicaSets

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

In

Notin

Exists



One must 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.

Quick Check

You need to configure ReplicaSets in Kubernetes to manage pods based on label selectors, allowing you to include, exclude, or check for specific labels. Which operators are key for defining pod label selectors in ReplicaSets?

A. And, Or, Exists

B. Equals, NotEquals, Contains

C. Match, Select, Exclude

D. In, NotIn, Exists

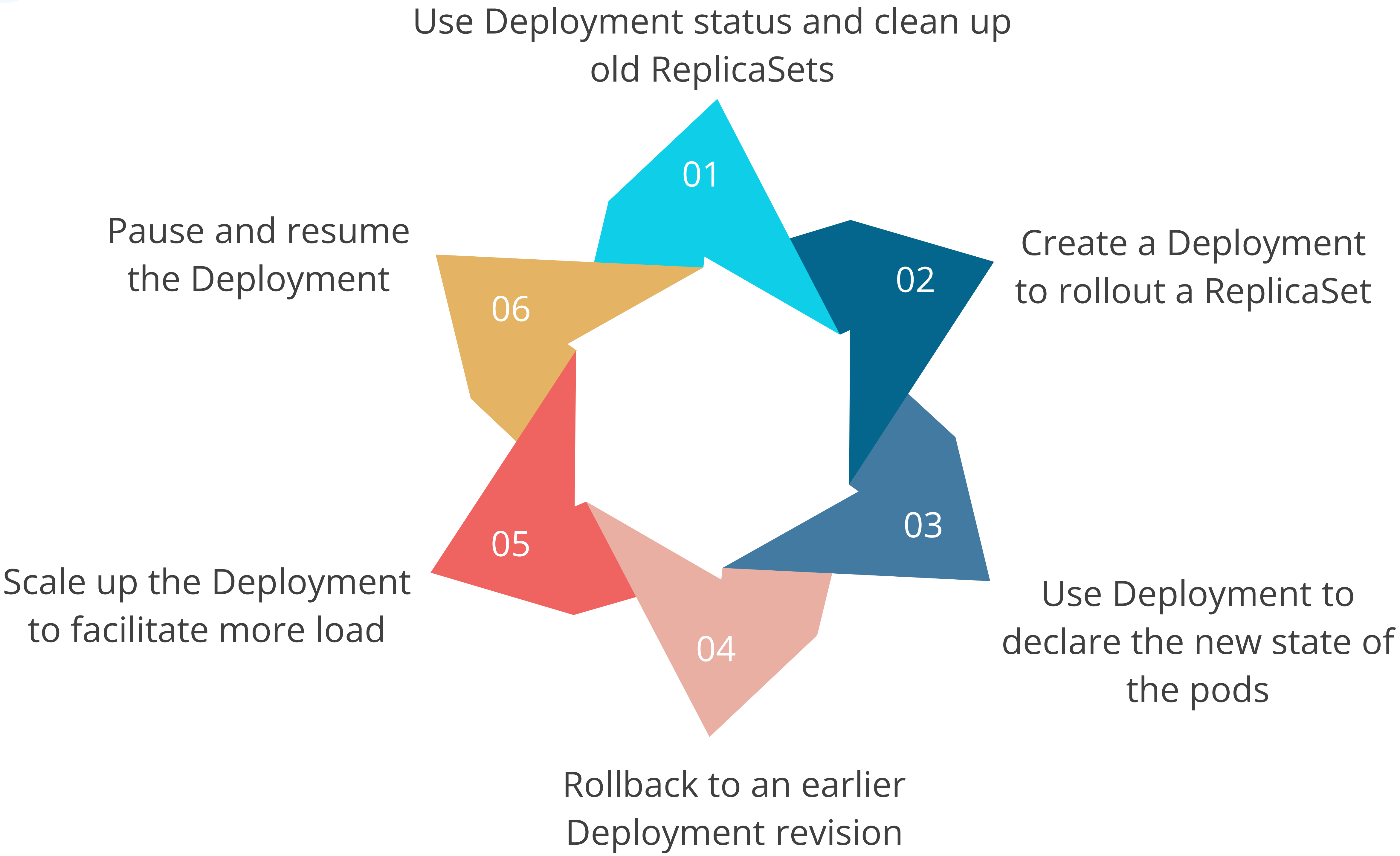
Overview of Deployment

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



Creating a Deployment

The following is an example of a Deployment:

Example

```yaml
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
```

Deployment Scaling

Deployments are useful for scaling the number of replicas as demand increases for the 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
```

Assisted Practice

Creating and Configuring the Deployment **Duration: 15 Min.**

Problem statement:
You have been assigned a task to create and configure deployment for an application for a production environment in a Kubernetes cluster.

Outcome:
By completing this demo, you will be able to deploy, configure, and manage applications in a Kubernetes production environment, ensuring high availability, scalability, and efficient resource utilization.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create the Deployment
2. Access the pod

ASSISTED PRACTICE

Quick Check

Your web application in Kubernetes needs more replicas to handle increased traffic. Which command will scale up the number of replicas in a Kubernetes deployment?

A. kubectl resize deployment [deployment-name] --replicas=20

B. kubectl create deployment [deployment-name] --replicas=20

C. kubectl scale deployment [deployment-name] --replicas=20

D. kubectl increase deployment [deployment-name] --replicas=20

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.

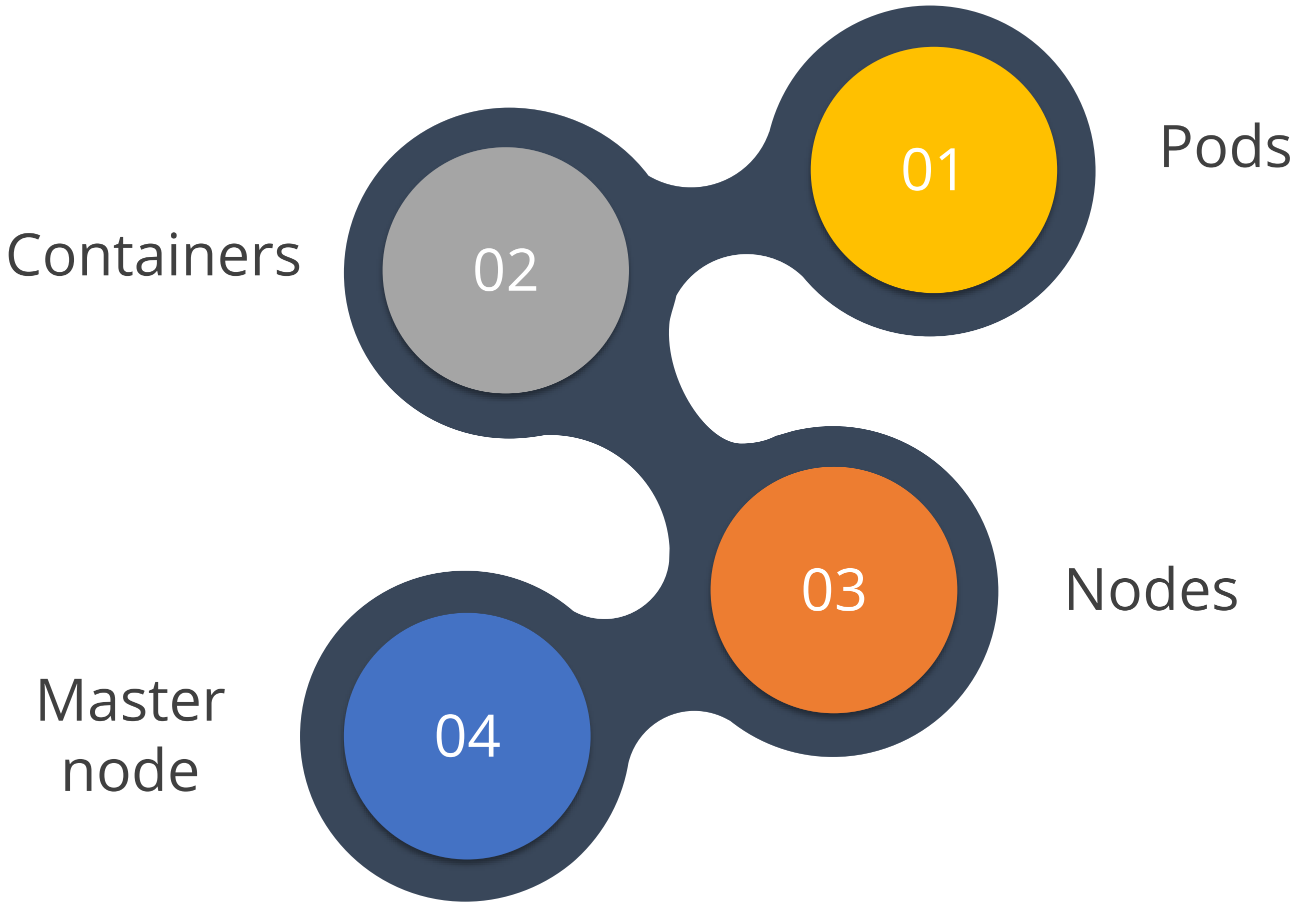
Cluster networking facilitates communication between various pods.

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

It 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

Assisted Practice

Using Basic Commands of Kubernetes

Duration: 15 Min.

Problem statement:

You have been asked to execute the basic commands used in Kubernetes for managing and interacting with cluster resources.

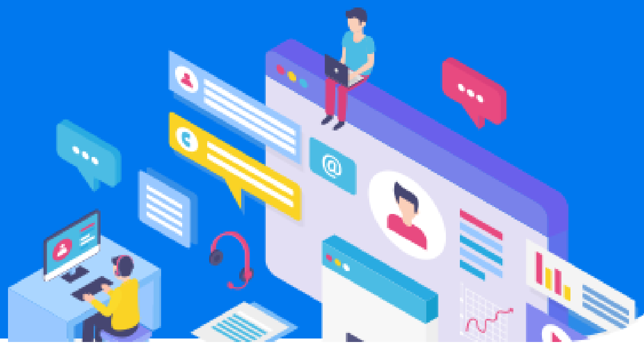
Outcome:

By completing this demo, you will be able to efficiently use Kubernetes command-line tools to manage pods, services, deployments, and other cluster resources.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

Assisted Practice: Guidelines



Steps to be followed:

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

ASSISTED PRACTICE

Overview of Policies

Policies

It defines 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

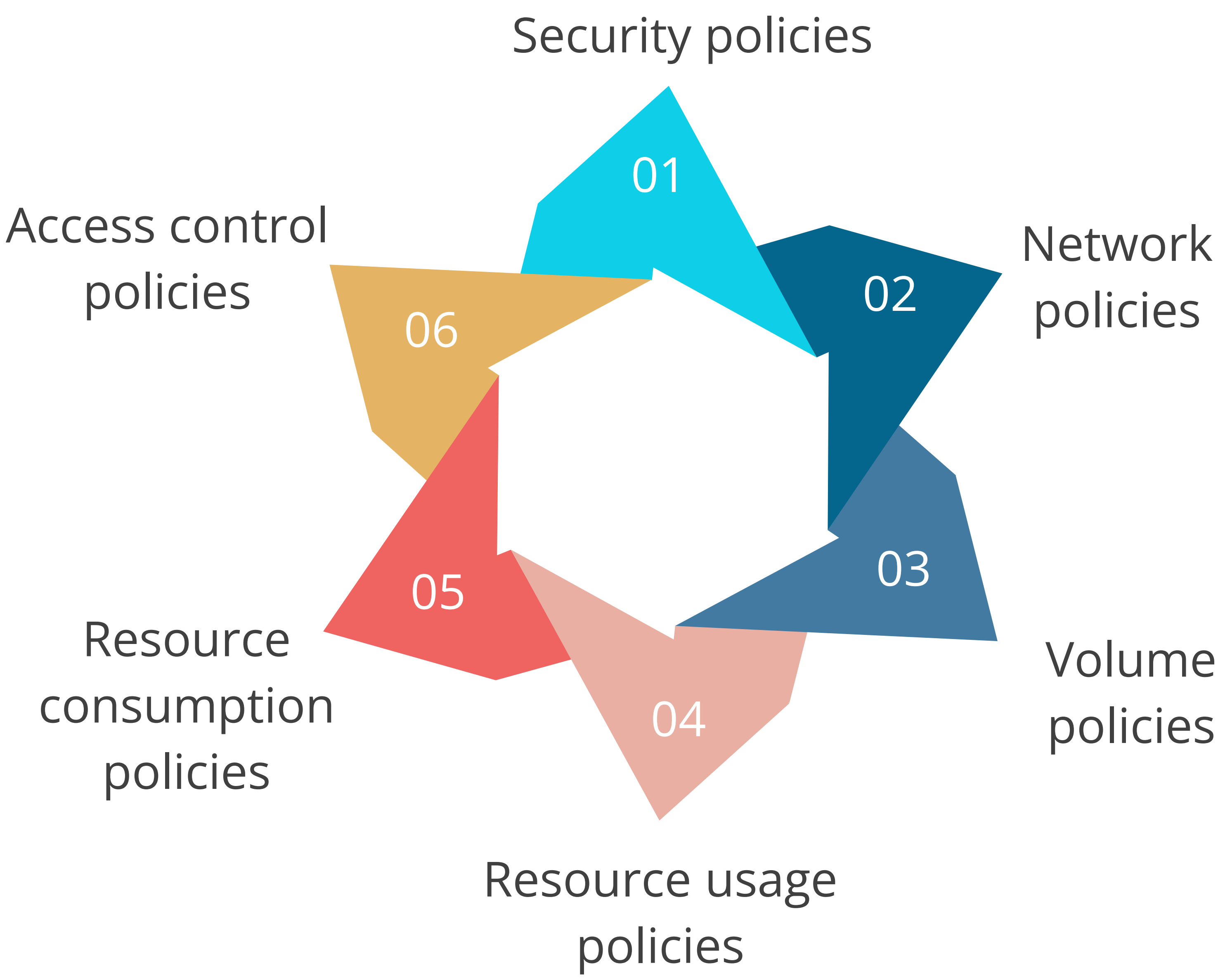
Ease of policy enforcement

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:



Quick Check

You need to control user actions and enforce rules for resource usage and security in your Kubernetes cluster. Which feature should you use to define and enforce these rules?

A. Policies

B. Services

C. ConfigMaps

D. Persistent volumes

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 minutes

Project agenda: To retrieve cluster-specific configurations from a running Kubernetes cluster, ensuring detailed insights into nodes, API versions, and operational statuses for optimal functionality and deployment readiness

Description: Your team lead has given you the task of accessing a Kubernetes cluster to gather specific details about it. You need to report on the available nodes and their IP addresses, determine the API versions supported by the server, check the status of the control plane and CoreDNS, and assess the status of the pods within the kube-system namespace.

LESSON-END PROJECT

Fetching Cluster Specific Configuration

Duration: 25 minutes

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

Expected deliverables: A Kubernetes cluster with high availability enabled

LESSON-END PROJECT

Thank You