Container Orchestration using Kubernetes

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 all the above, and some additional features, you will learn a few concepts in this lesson that will help you in finding a solution for the given scenario.

Learning Objectives

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

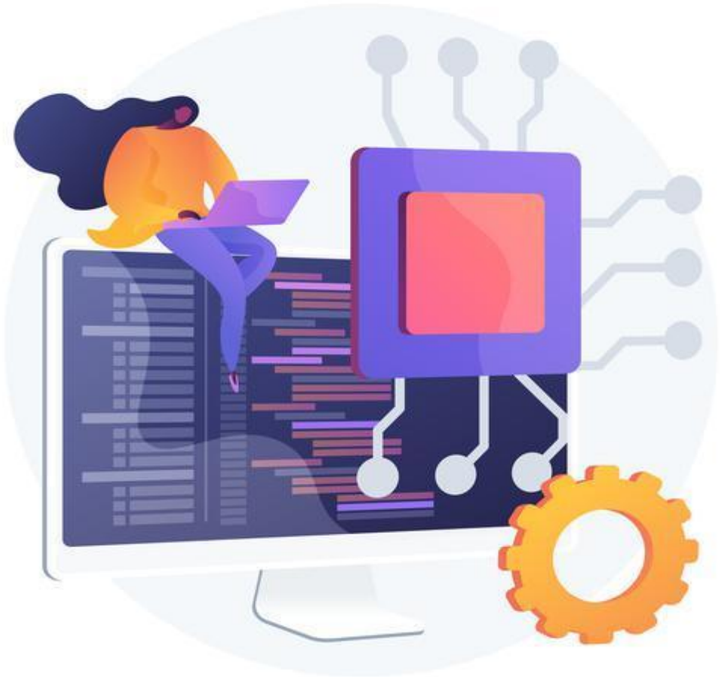
- Explain the working of Container Orchestration in Kubernetes
- Describe the functions of Etcd, Controller, Scheduler, and Kubelet
- Differentiate kube-proxy, Pods, ReplicaSet, and Deployment
- Classify common Deployment use cases
- Summarize an overview of Containers and Policies

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.

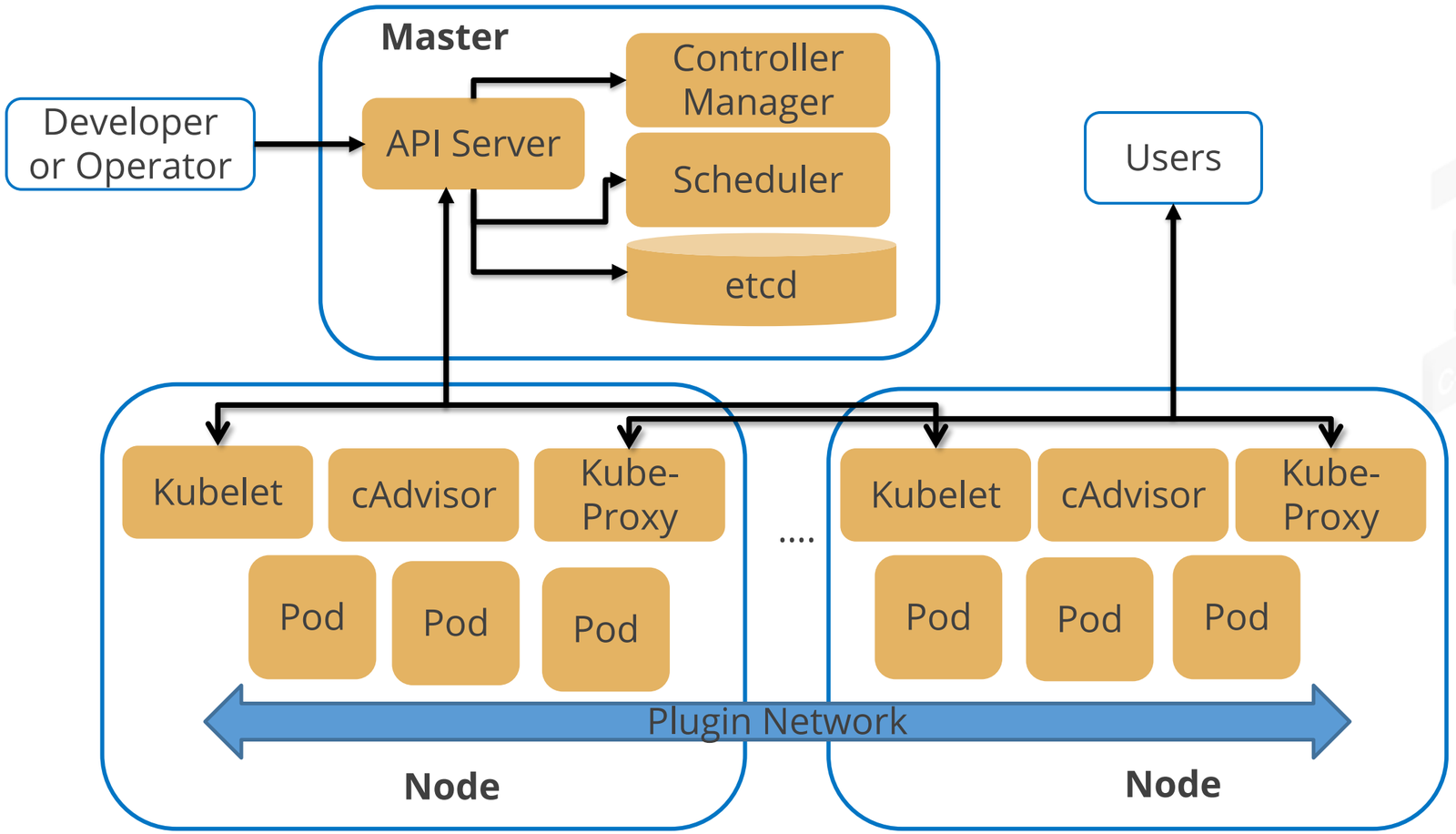


The platform gains control over computing and storage resources by defining resources.

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

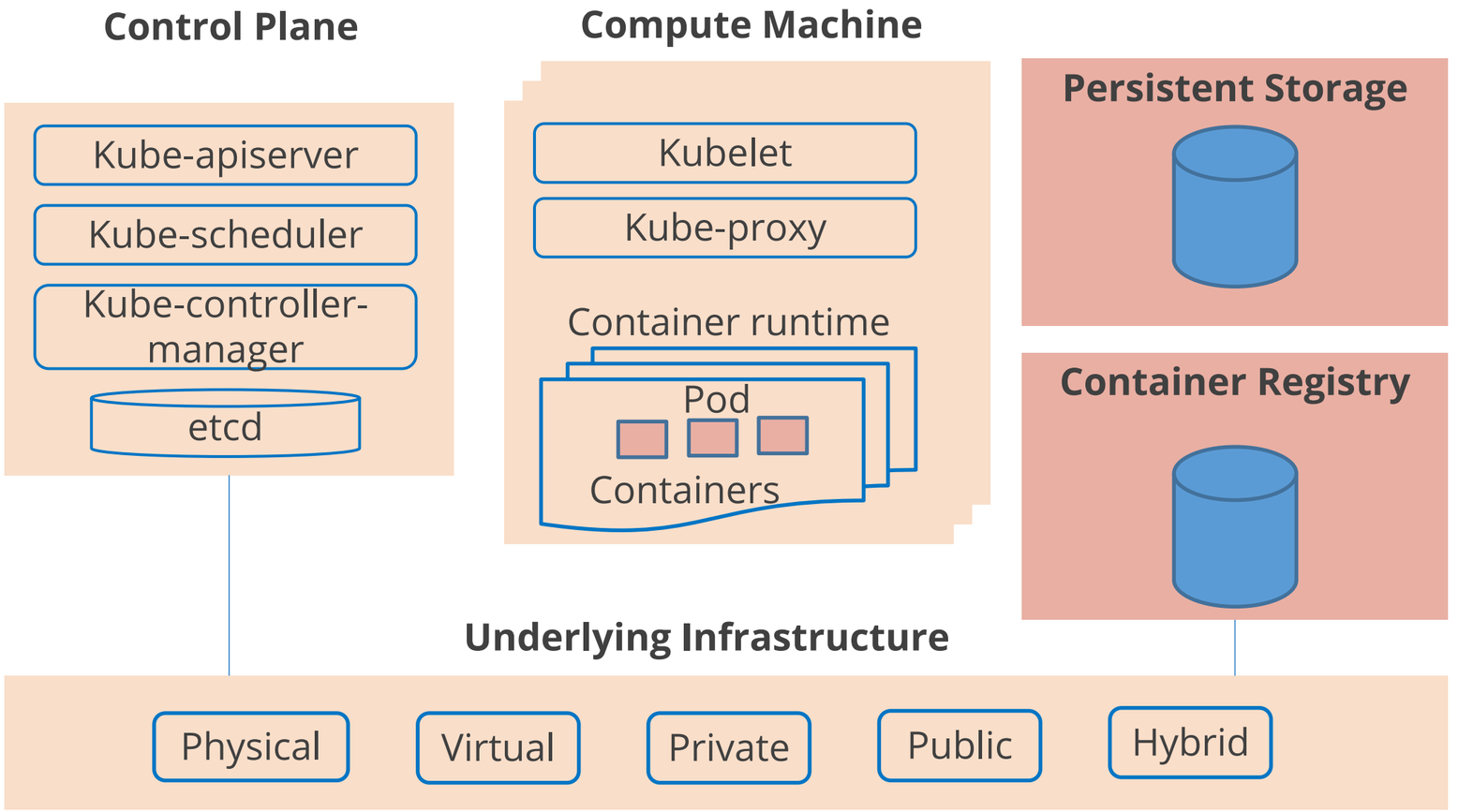
Kubernetes Architecture

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



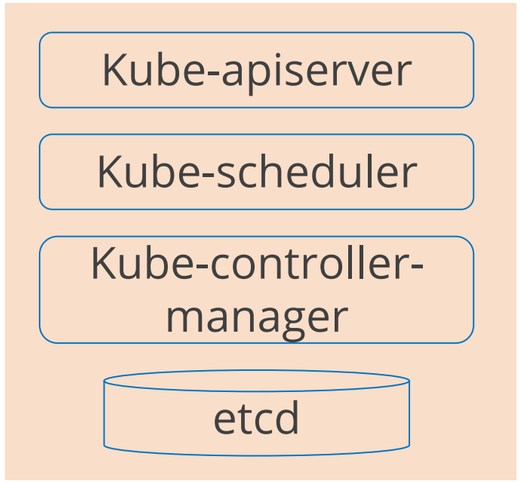
Kubernetes Cluster

A Kubernetes Cluster consists of a master node and a set of worker nodes.



Control Plane

The Control Plane(Master) is in constant contact with the compute machines and ensures that the containers are running on necessary resources.



Kube-apiserver handles internal and external requests.

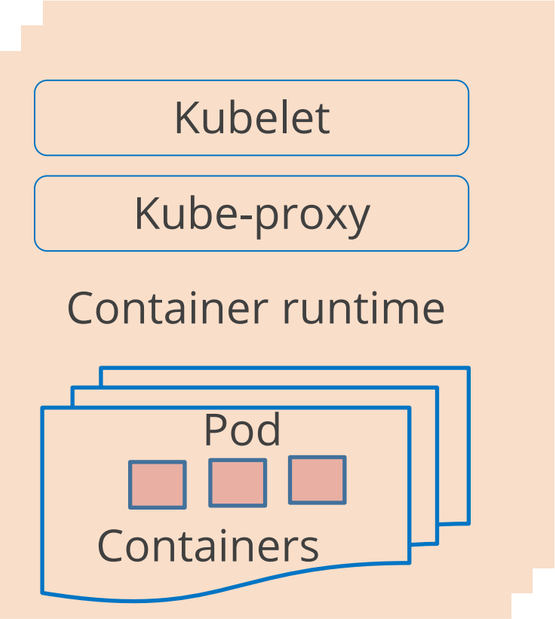
Kube-scheduler schedules the Pod to an appropriate node.

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. As control plane is not schedulable by default because of the default taints value.



Kubelet communicates with the control plane.

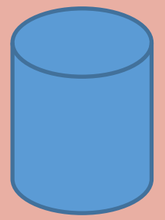
Kube-proxy is a network proxy that facilitates Kubernetes network services.

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

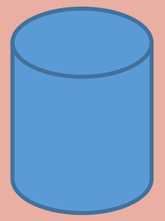
Pod represents a single instance of an application.

Storage and Registry

Persistent Storage



Container Registry



Persistent Storage manages application data attached to a cluster

Container Registry stores the container images that the Kubernetes relies on

Features of Kubernetes

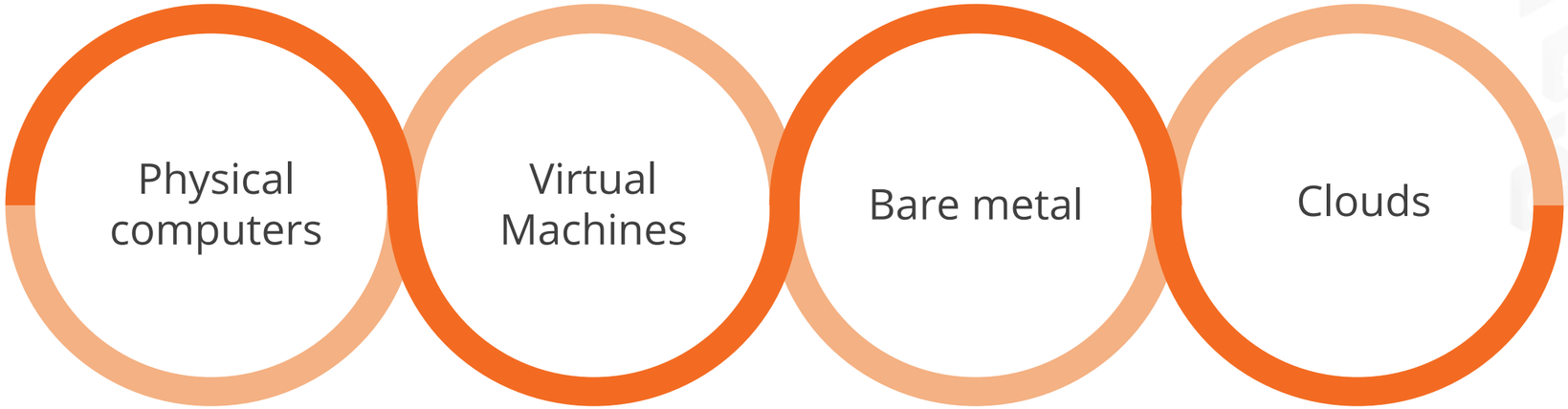
Kubernetes offers the following features to its users:

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

Containers

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

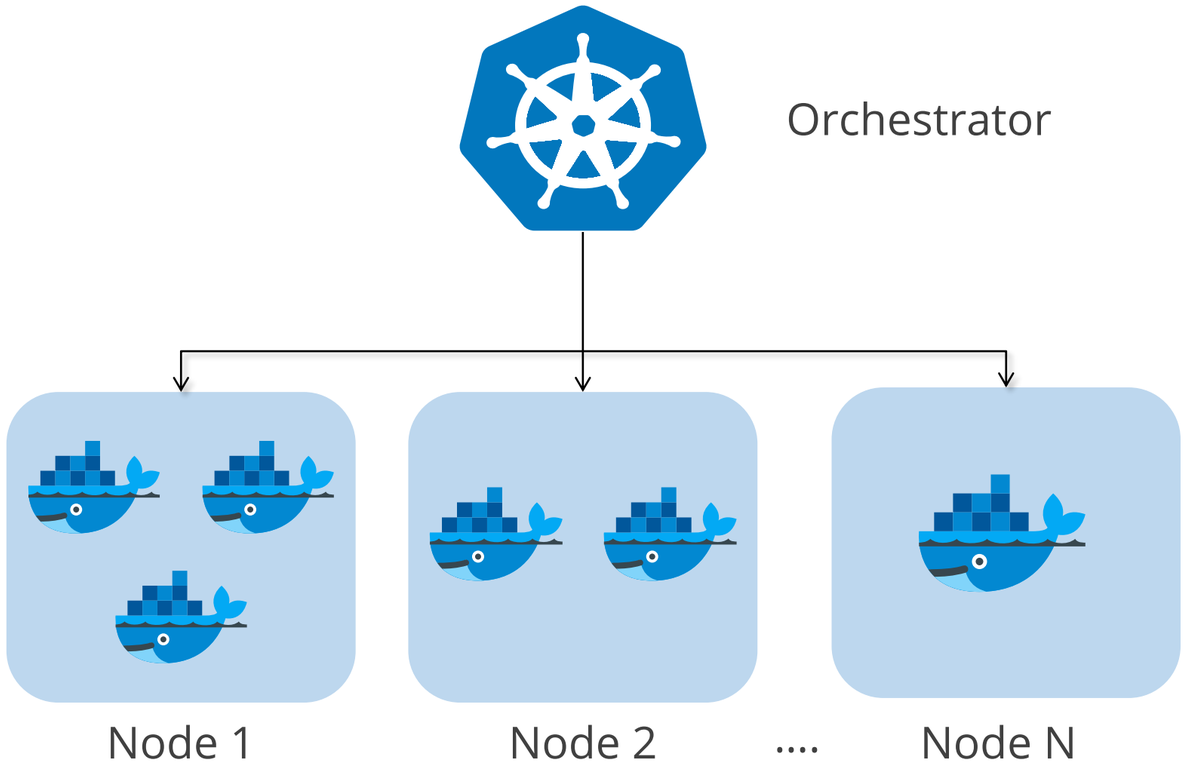
Docker containers run in four modes:



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

Container Orchestration

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



Kubernetes: Overview

The aspects of Kubernetes are as follows:

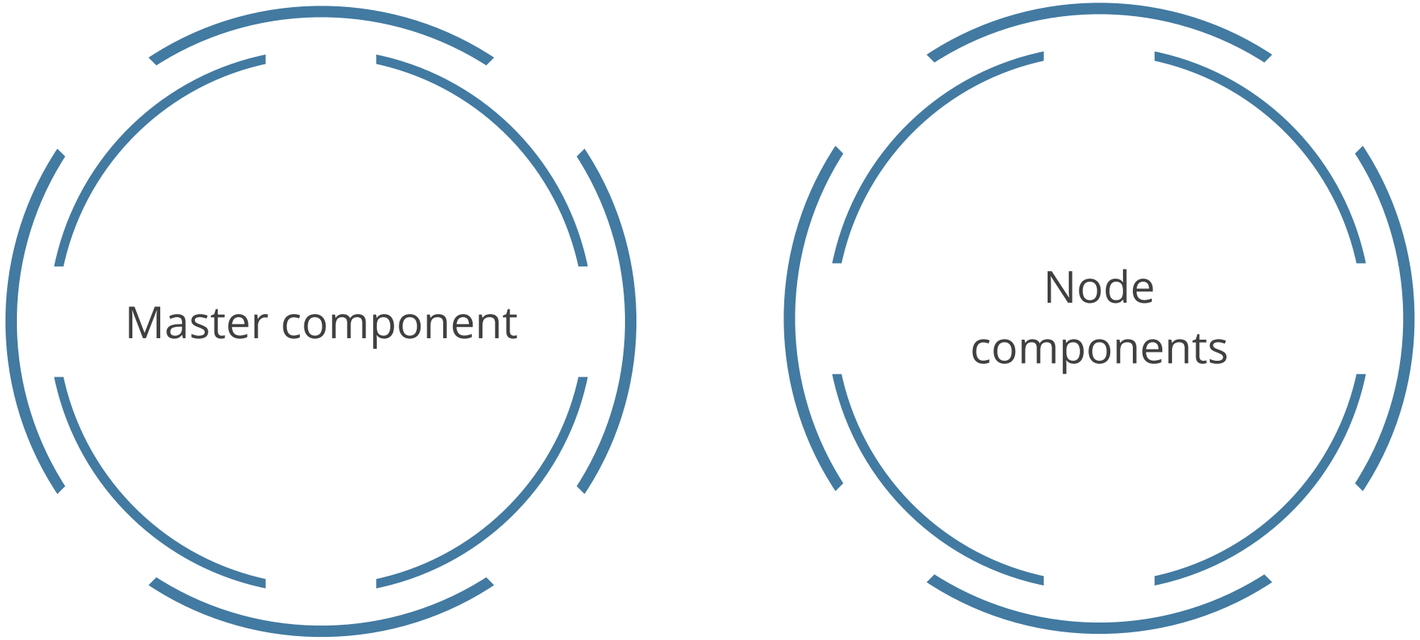
Kubernetes components

Kubernetes API

Kubernetes objects

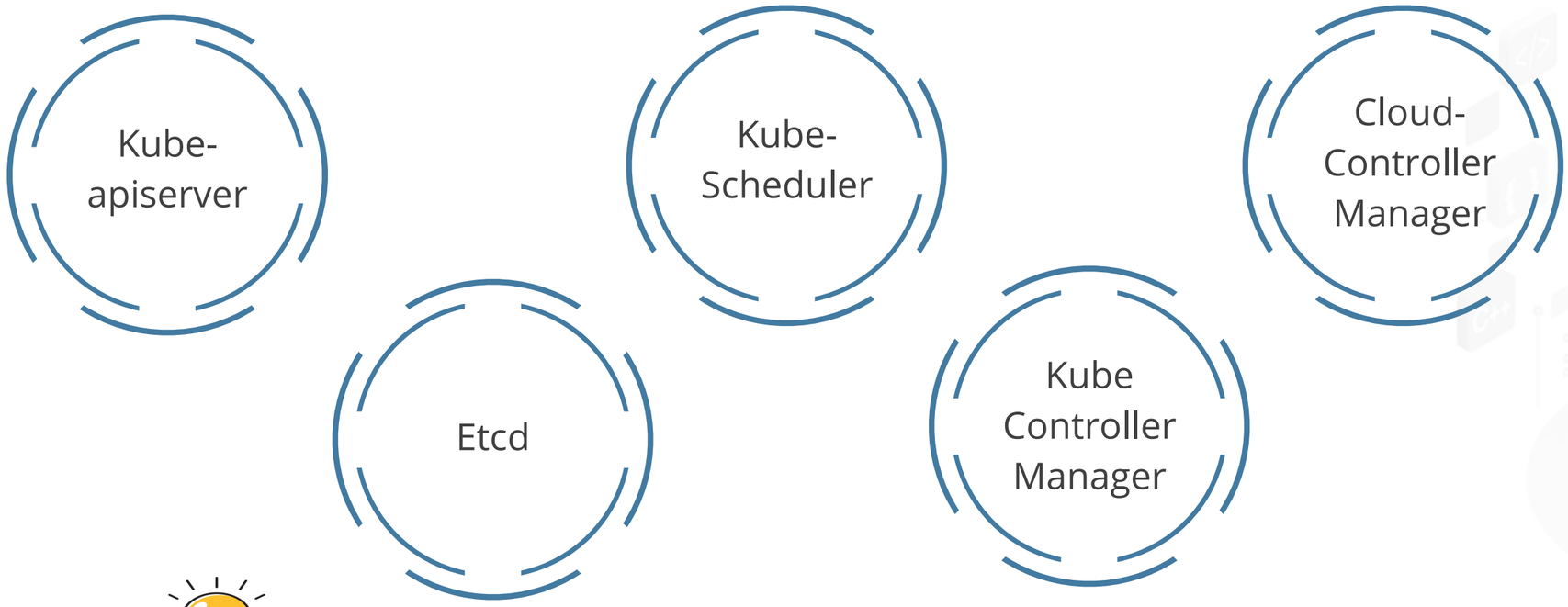
Kubernetes Components

A Kubernetes cluster consists of components that represent the control plane and a set of machines called nodes. The components are divided into:



Master Components

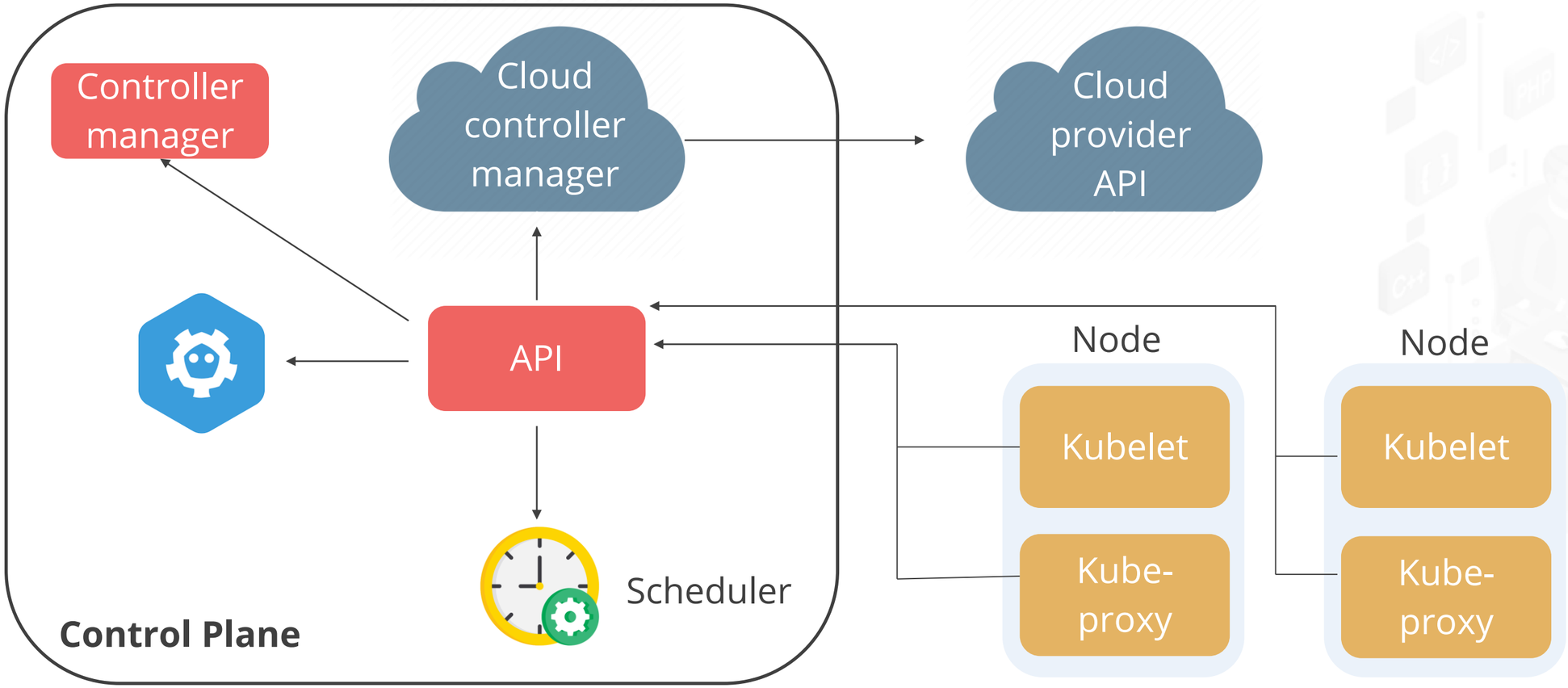
Master components are designed to globally monitor the cluster and respond to cluster events.
These components are running as a pod on master in the kube-system namespace.



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

Cloud-Controller Manager

The Cloud-Controller Manager helps to link clusters into Cloud Provider's API and separates components that interact with the cloud platform.



Kubernetes API

The Kubernetes API 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

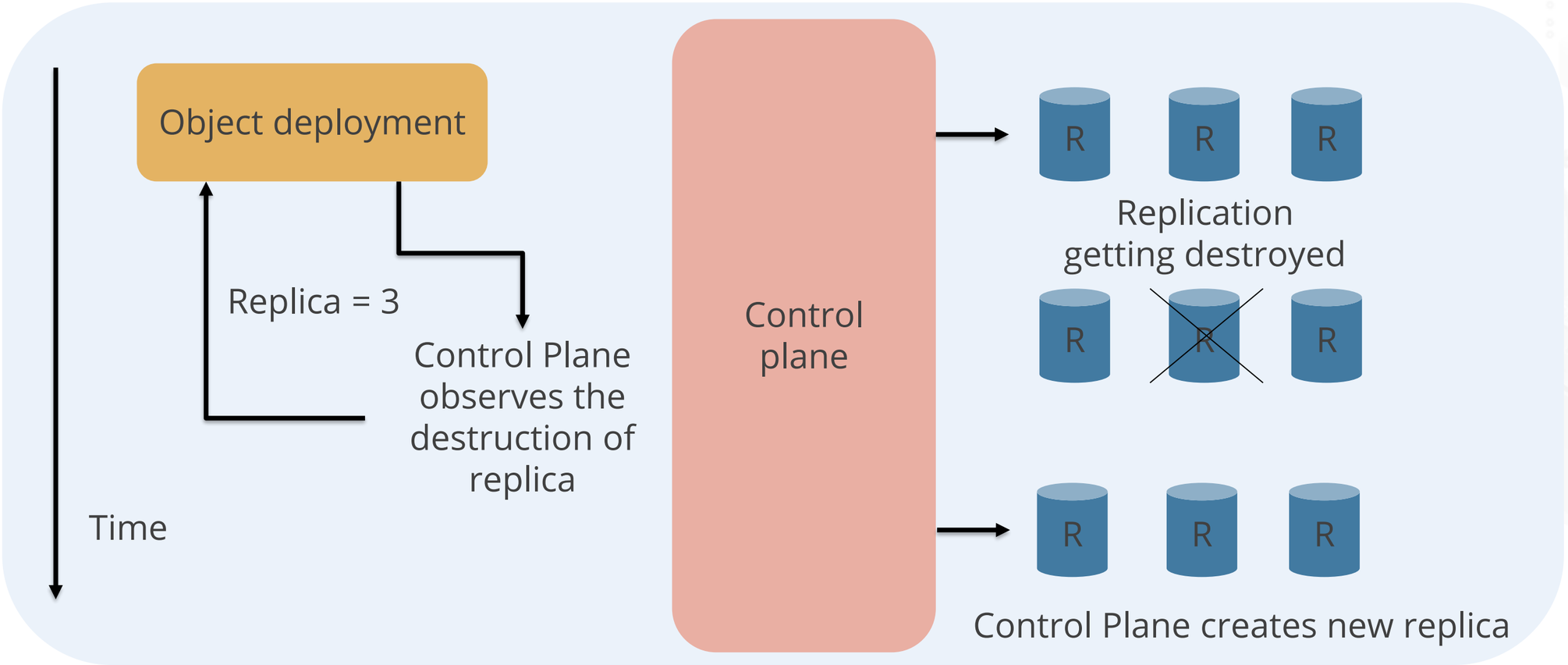
Kubernetes objects are persistent entities in the Kubernetes system.

These objects can describe:

- 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.**



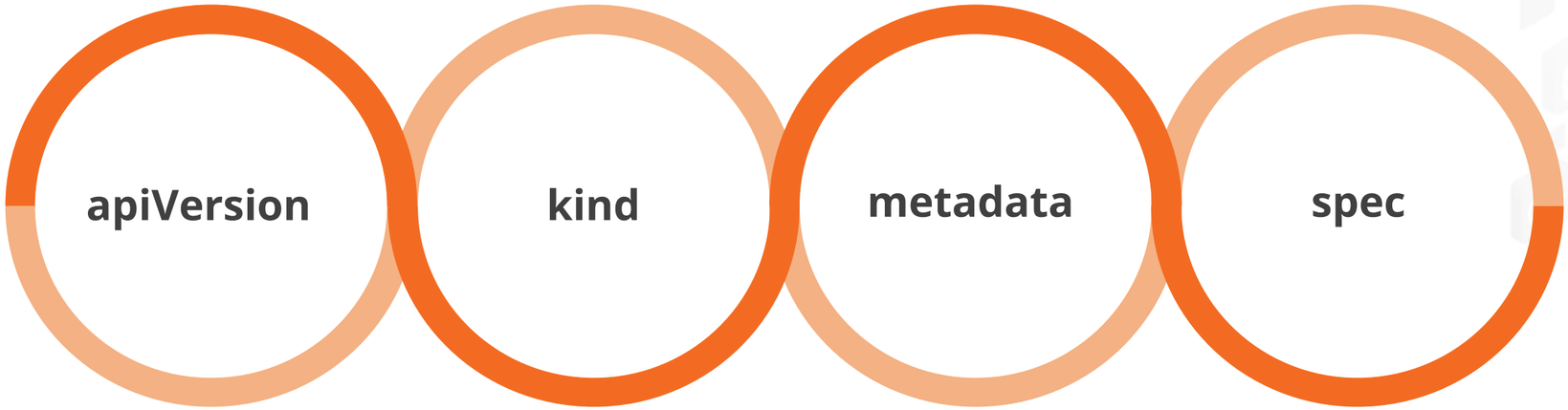
Describing Kubernetes Object

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.

```
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:



Create and Configure Kubernetes Cluster

Duration: 20 mins

Problem Statement:

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

ASSISTED PRACTICE

Assisted Practice: Guidelines

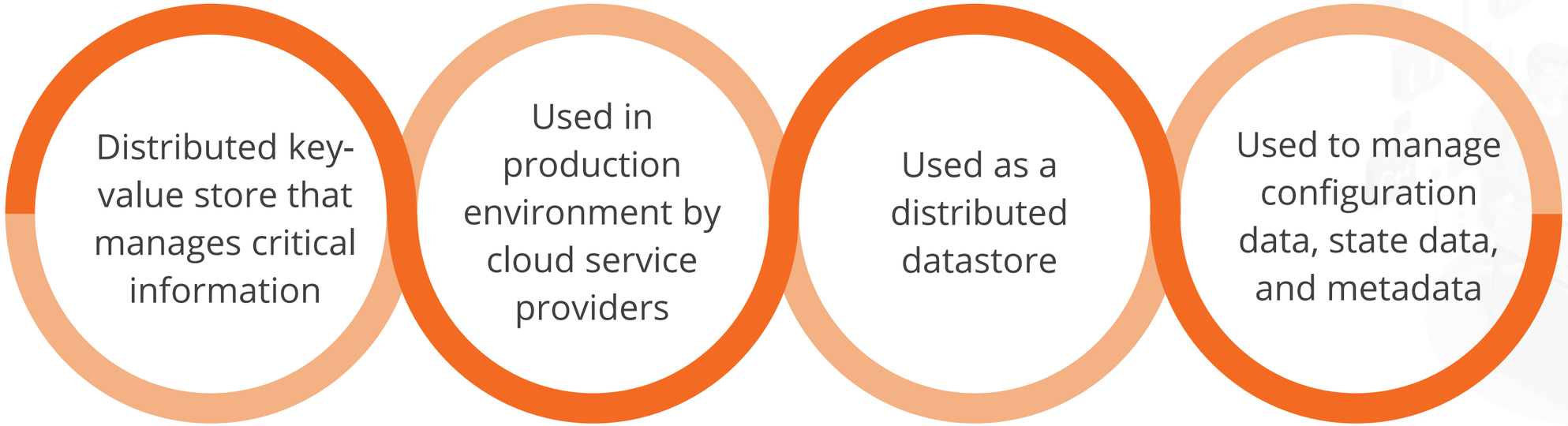
Steps to be followed:

1. Changing the hostnames of all machines
2. Setting up the master node and configuring the cluster
3. Joining the worker nodes to the cluster
4. Verifying the nodes in the cluster

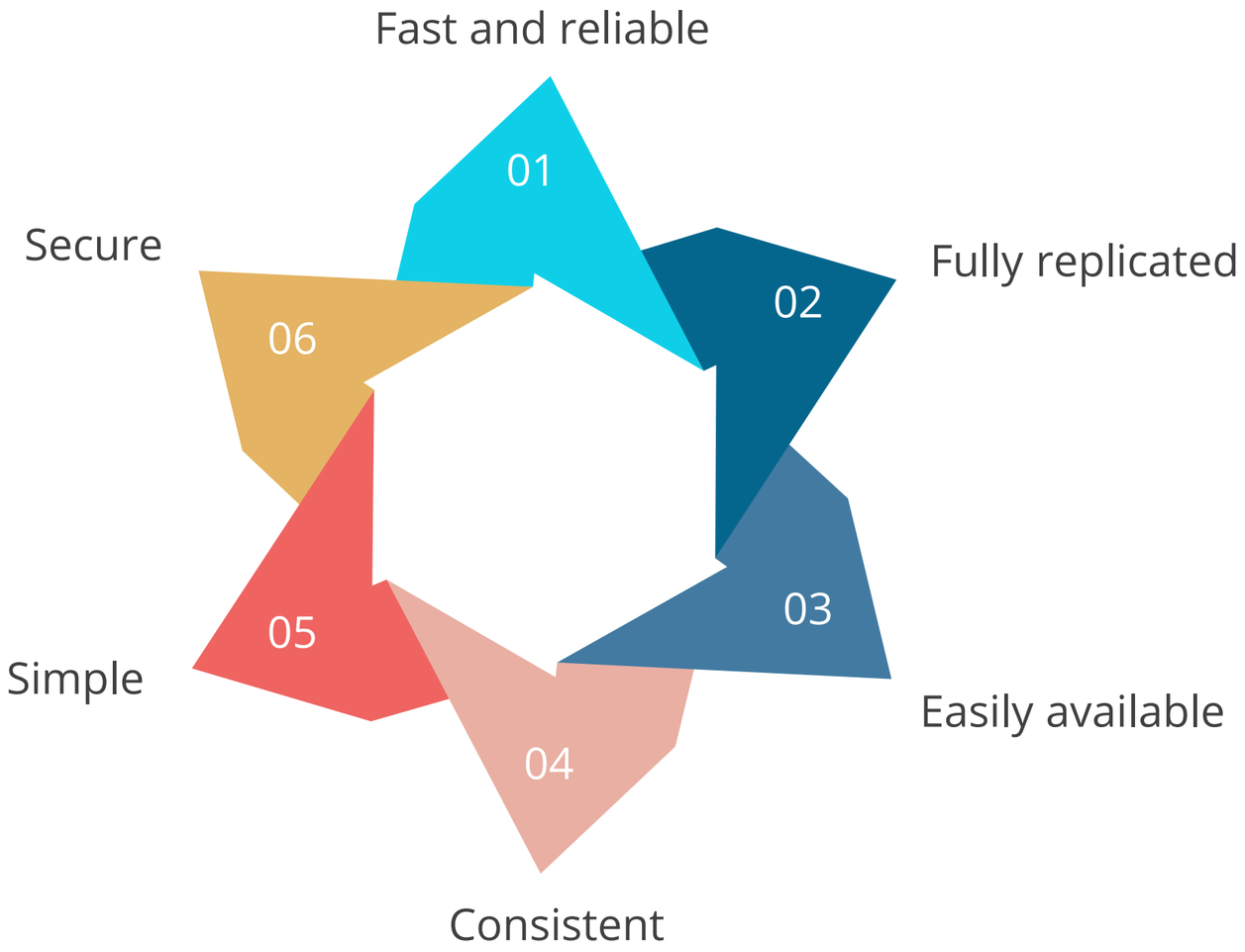
Etcd

Etcd

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



Properties of Etcd



Working of Etcd

Working of Etcd 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

Controller

Controllers in Kubernetes

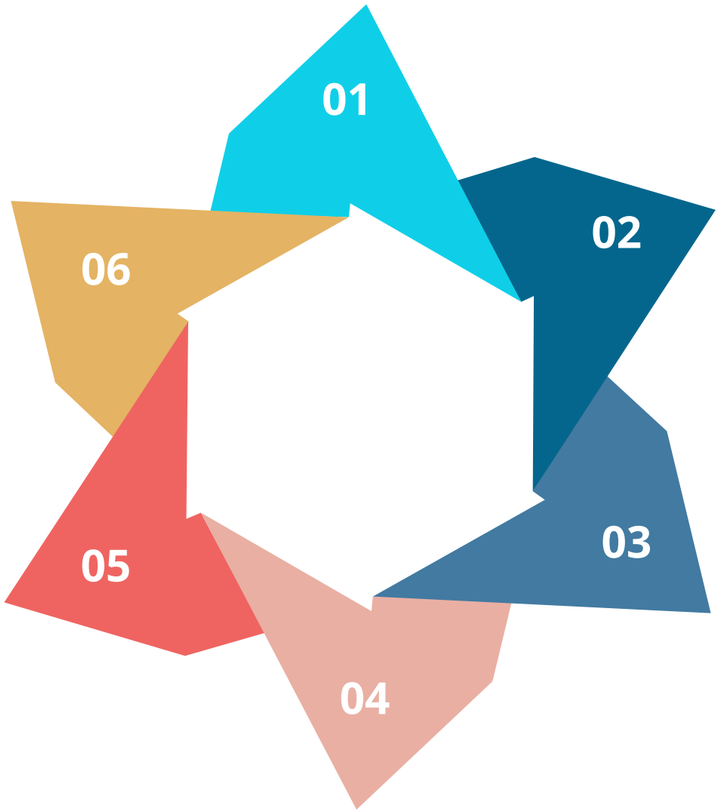
Built-in Controllers manage the state by interacting with the cluster API server.

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

CronJob: Creates a job on schedule



ReplicaSet: Creates a stable set of Pods

Job: Creates one or more short-lived Pods

Deployment: Maintains ReplicaSets with a desired configuration

StatefulSet: Manages stateful applications

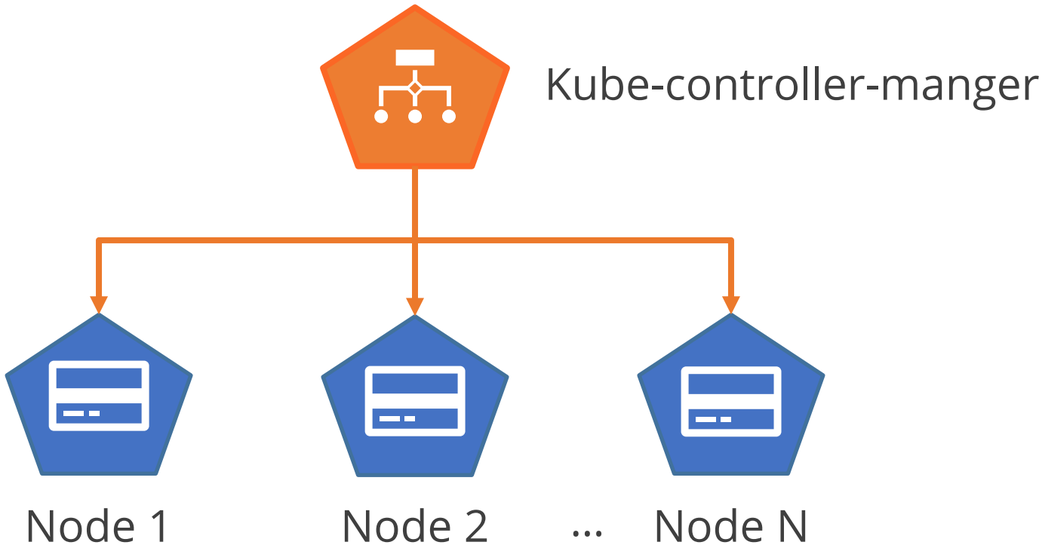
Controller Design

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

A Kube-scheduler 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.

Kubelet

Kubelet

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

The Kubelet 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

Kube-proxy

Kube-proxy

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

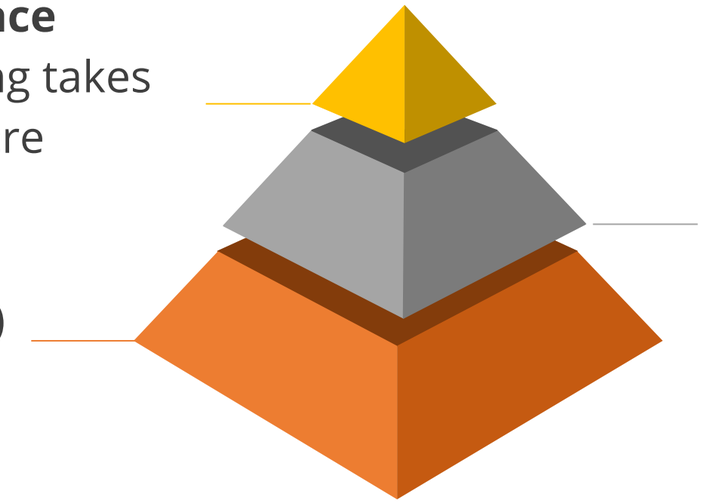
Maintains network rules on nodes

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

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

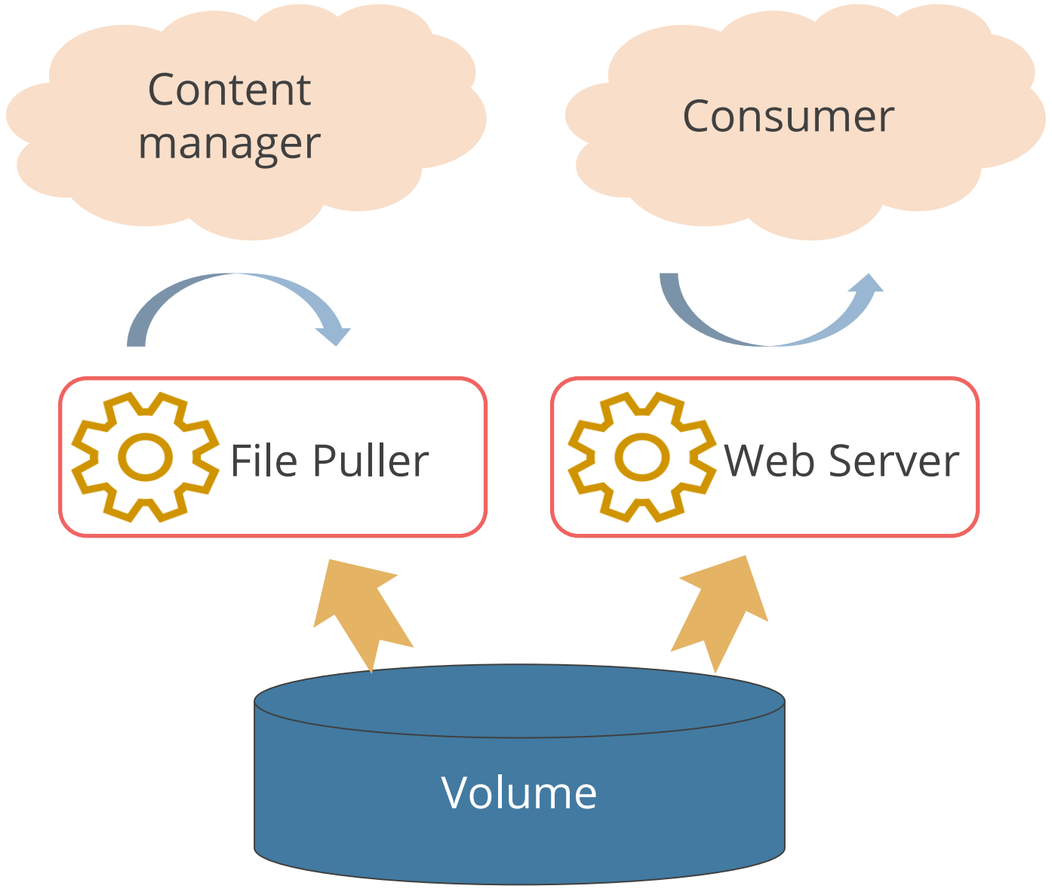
Pods also contain shared networking and storage resources for their containers:

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?

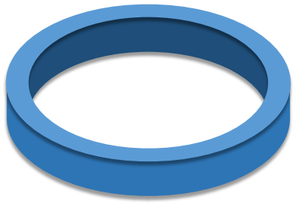
Pods are designed to support multiple cooperating processes (as containers) that form a cohesive unit of service.



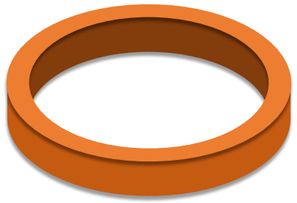
Pods and Controllers

Workload resources create and manage one or more Pods.

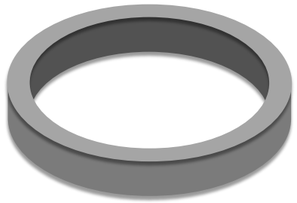
Examples of workload resources:



Deployment



StatefulSet



DaemonSet

Pod Template

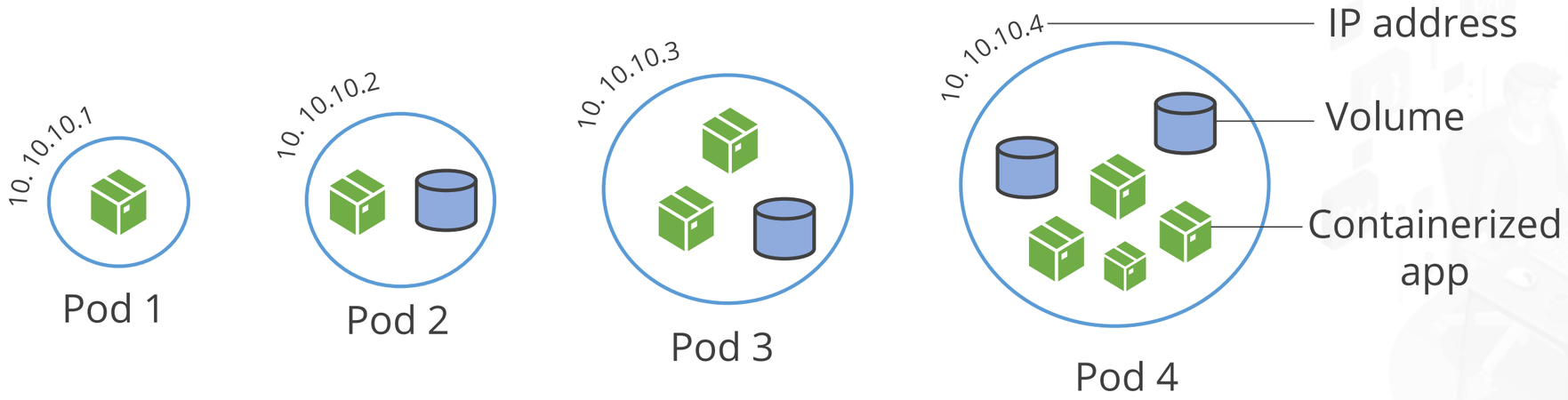
Pod templates are specifications for creating Pods and are included in workload resources, such as deployments, Jobs, and DaemonSets.

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.

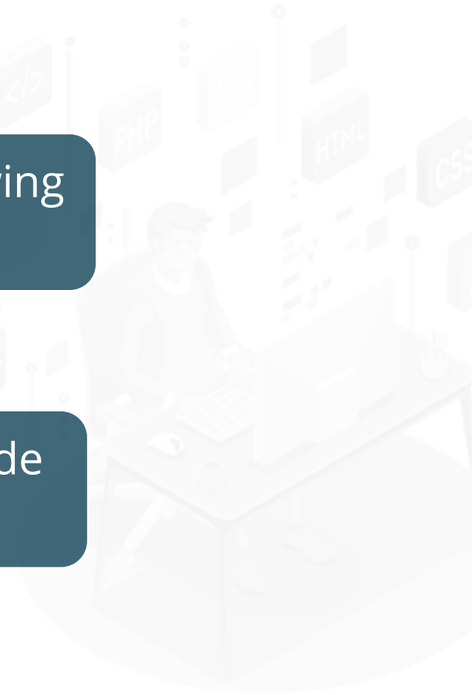
When updating the **spec.activeDeadline** seconds field, two types of updates are allowed.

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.

Configure Pods

Duration: 15 mins

Problem Statement:

You've been assigned a task to create and configure Pods and execute the Apache Services.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Configuring and setting up the Pod files
2. Configuring and setting up the Service file
3. Executing the Apache services

Create OpenShift Pods

Duration: 10 mins

Problem Statement:

You've been assigned a task to create an OpenShift Pod.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Configuring and setting up the Pod files
2. Executing the Pod

ReplicaSets

ReplicaSets

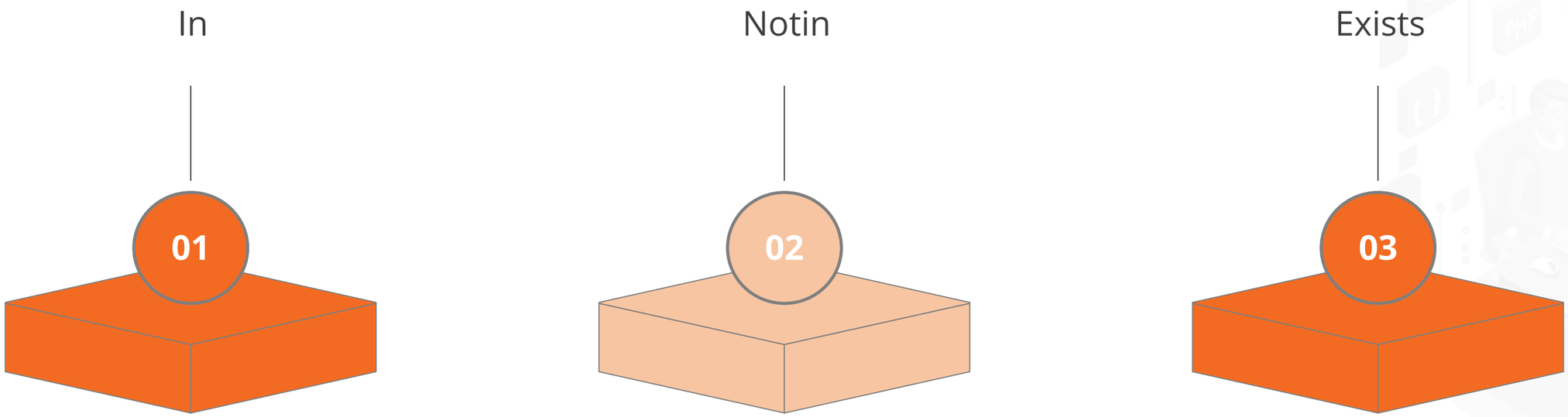
ReplicaSets maintain a stable set of replica Pods running at any given time.

Guarantees the availability of a specified number of identical Pods

Ensures that a specified number of Pod replicas are running

Operators to Use with ReplicaSets

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.

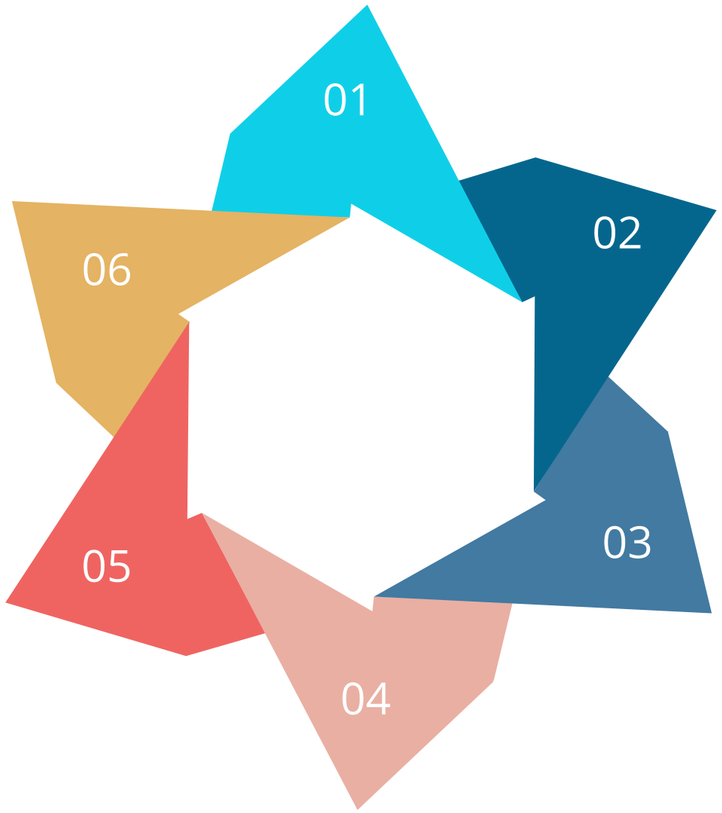
Deployments

Deployment

A Deployment provides declarative updates for Pods and ReplicaSets.

Deployments 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

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

Problem Statement:

You've been assigned a task to create and configure Deployment for an application.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Creating the Deployment
2. Accessing the Pod

TECHNOLOGY

Services, Load Balancing, and Networking

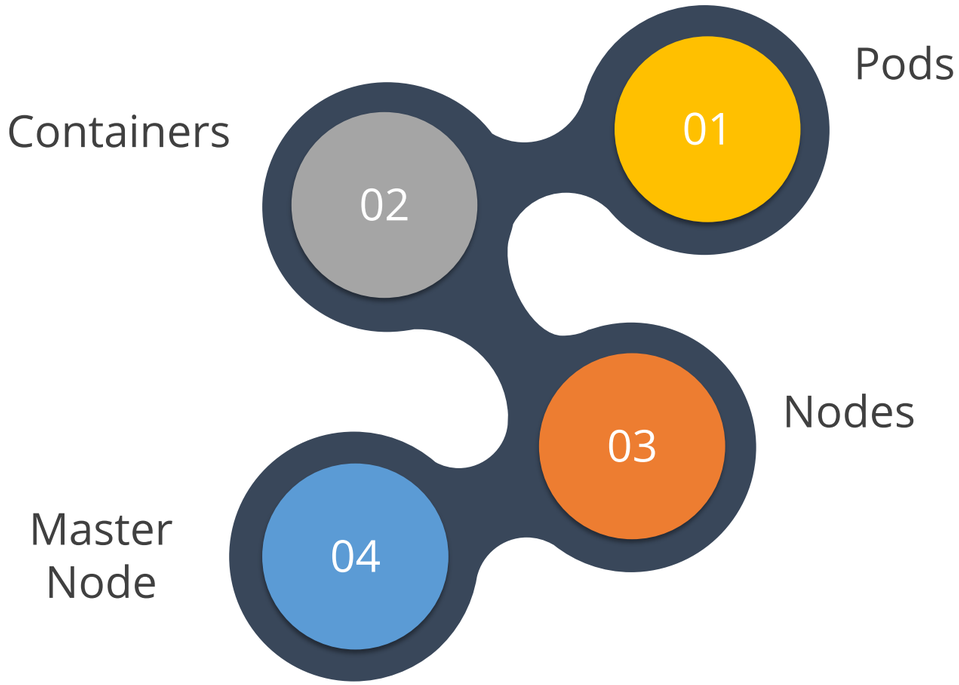
Overview

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

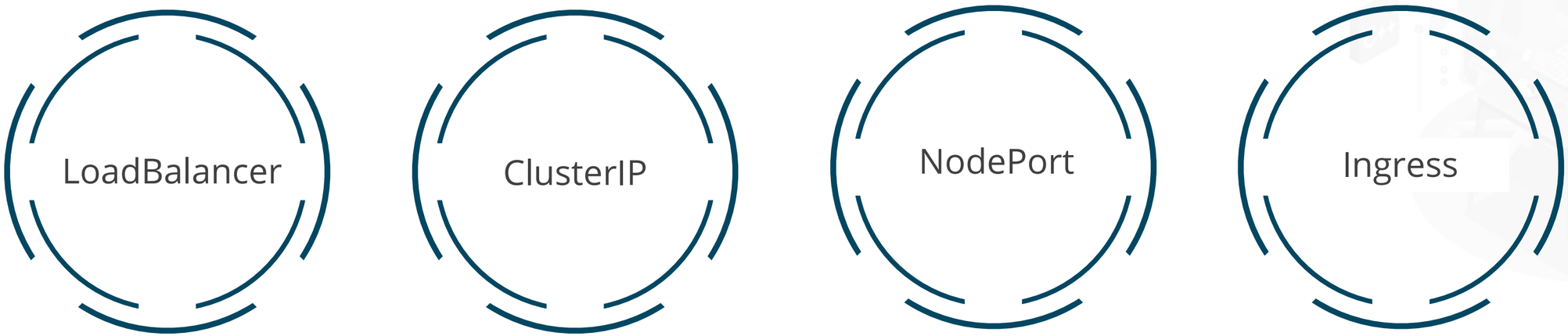
A Kubernetes Pod network connects several interrelated components including:



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:



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

Containerized applications 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.

Union File Systems prevent data duplication.

Control Groups implement resource accounting and resource limitation.

Understanding Basic Commands of Kubernetes

Duration: 15 mins

Problem Statement:

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

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Creating the Pod
2. Creating the Deployment and namespace
3. Deleting all the Services

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

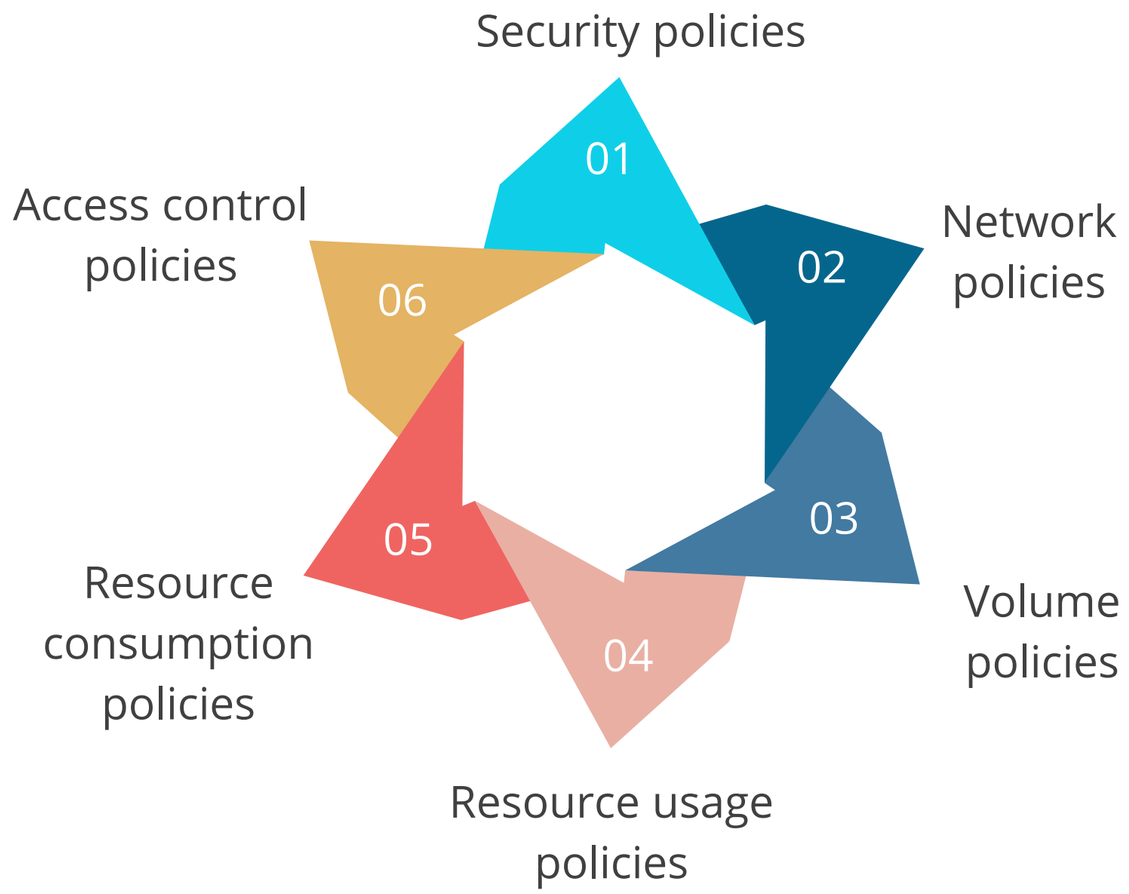
Ease of policy enforcement

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:



Creating Jobs

Duration: 10 mins

Problem Statement:

You've been assigned a task to create and set up Jobs.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Creating the Jobs
2. Setting up the **jobs.yaml** file

Key Takeaways

- Containers are lightweight, standalone, executable software package that includes 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 possible ways to ensure that clusters comply.

Fetch Cluster Specific Configuration

Duration: 25 Min.

Description: Your team lead has requested you to connect to the Kubernetes cluster and provide the following cluster 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. Setting up the cluster
2. Listing available nodes and their IP addresses
3. Identifying API versions that are supported
4. Examining the control plane and CoreDNS status
5. Checking the status of the Pods with kube-system namespace

LESSON-END PROJECT