Container Orchestration using Kubernetes

Kubernetes Cluster

A Day in the Life of an DevOps Engineer

You are working as a DevOps Engineer in an organization that uses single cluster for development, testing and production where in for a particular project, you are required to use multiple Kubernetes namespaces with respective roles.

The goal is to make sure that multiple Kubernetes namespaces are used in the development process to make the work of developers and testers easy by deploying and managing their own namespaces throughout the development.

To achieve the above, along with some additional features, you will be learning 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:

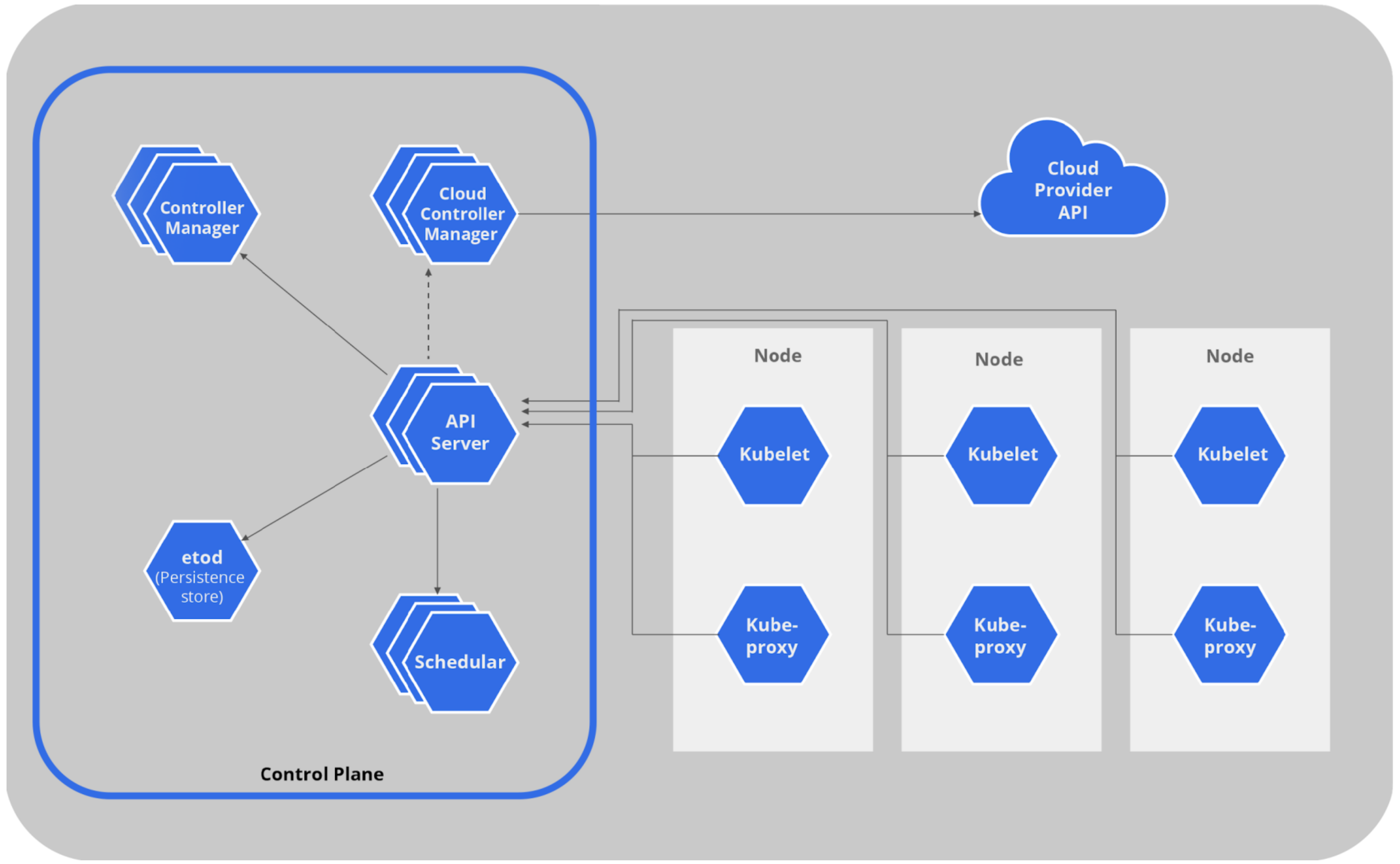
- Explain the Cluster Architecture
- Summarize an overview of Nodes
- Describe Controller in details
- Explain the working of kubeadm
- Classify API Server

TECHNOLOGY

Cluster Architecture Overview

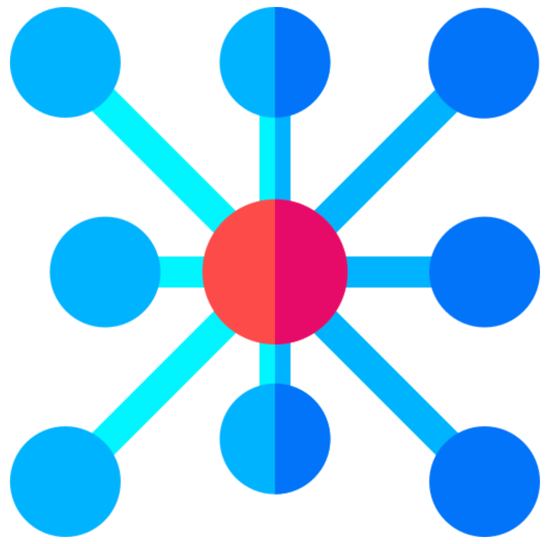
Kubernetes Cluster

Kubernetes Cluster is a set of Nodes for running containerized applications. It contains a Control Plane and one or more Nodes.



Control Plane

A Control Plane is responsible for maintaining the desired state of the cluster. It manages all the applications running in the cluster and the container images associated with the applications.



Nodes are the components that run the applications and workloads associated with them.

Control Plane is the master node in Kubernetes.

Components of Control Plane

The Control Plane has five important components:

1. kube-apiserver
2. etcd
3. kube-scheduler
4. kube-controller-manager
5. Cloud-controller-manager

Nodes

Kubernetes places Containers in Pods to run the workload. These Pods run on Nodes. Nodes are physical or virtual machines that are a part of the cluster.

A Node has two components:

kubelet

kube-proxy

Container Runtime

Container Runtime is the software responsible for running Containers. Kubernetes supports many Container Runtimes including docker, containerd, and CRI-O.

The two most common ways to add Nodes to an API server are:

Kubelet registers itself to the Control Plane.

Users manually add a Node object.

Container Runtime

Following the Node registration, the Control Plane checks the validity of the new Node object. A sample JSON manifest from which a Node is being created is shown below:

Demo

```
{
 “kind”:  “Node”,
 “apiVersion”: “v1”
 “metadata”:         {
   “name”:  “10.240.79.157”
   “label”:            {
     “name”:   “myfirst-k8s-node”
                       }
                       }
 }
```

An unhealthy Node is ignored for any cluster activity.

Control Plane-Node Communication

Kubernetes has a **hub and spoke** API Pattern. All the API usage from Nodes (or Pods) terminates at the API server.

There are two communication paths from Control Plane to Node:

1. Apiserver to kubelet
2. Apiserver to Nodes, Pods, and services

TECHNOLOGY

Configuring a Cluster

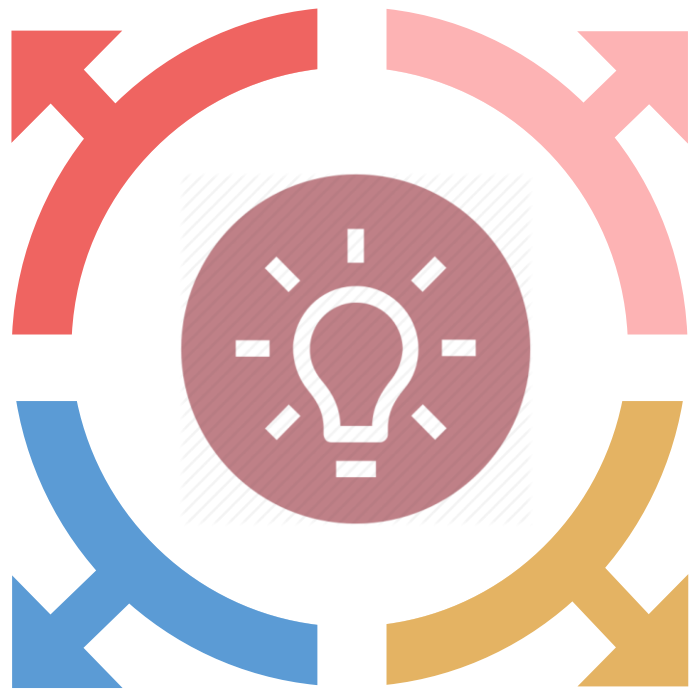
simplilearn

General Configuration Tips

The latest stable API version must be specified while defining the configurations.

Store the configuration files in version control

Use YAML and not JSON to write configuration files



Group related objects in a single file

Call kubectl commands on a directory

Creating Pods

Pods can be created using:

Deployment

Deployment is the most preferred way to create Pods. It creates a ReplicaSet and specifies a strategy to replace Pods.

Jobs

Jobs create Pods that perform a specific job and exit.

Naked Pods

Naked Pods are Pods created directly using a definition file. They are not bound to a ReplicaSet or Deployment.

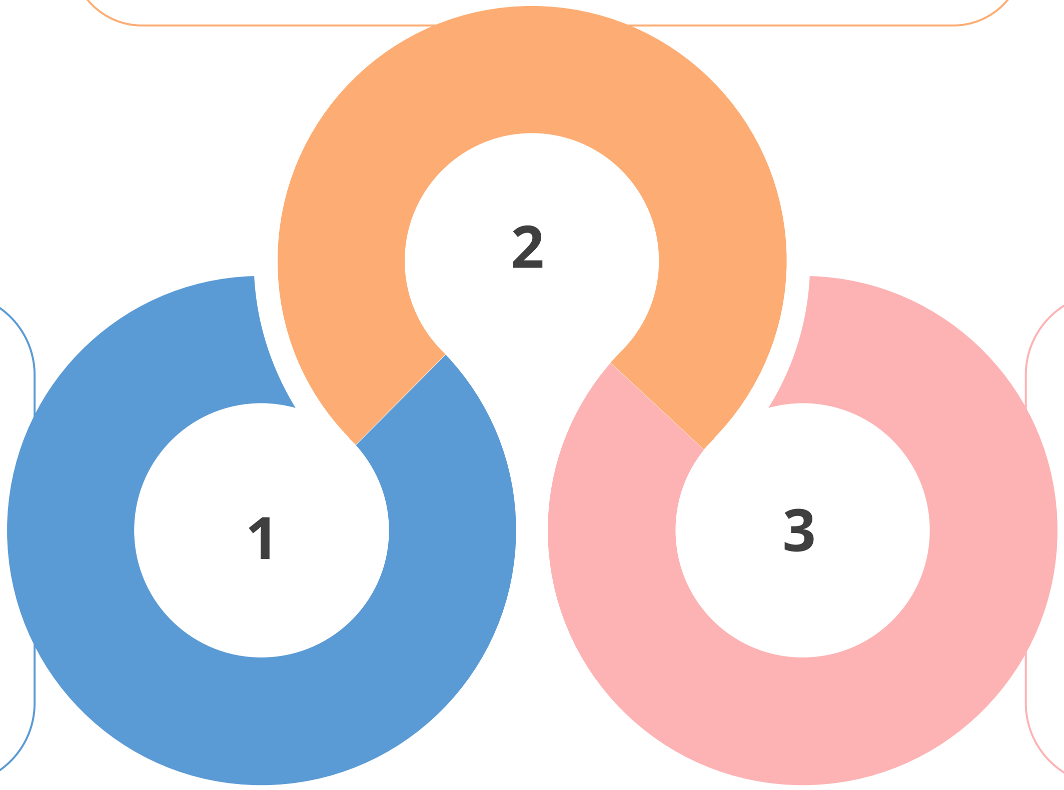
Guidelines for Creating a Service

Follow these guidelines to create a service:

1. Avoid using **hostNetwork** for the same reasons as **hostPort**
2. Use **headless** services for service discovery when you don't need kube-proxy load balancing
3. Create a service before its corresponding backend workloads
4. Add DNS server as a cluster add-on to manage new services and create DNS records
5. Specify a hostPort for a Pod only when it is necessary

Using Labels

Labels are key or value pairs that are attached to objects, such as pods.



2. Omit release-specific labels from their selector to make a service span multiple Deployments

1. Define and use labels that identify semantic attributes { app: myapp, tier: frontend, phase: test, deployment: v3 }

3. Labels can be used to select appropriate Pods for the resources.

Container Images

When the kubelet attempts to pull a specified image, the **imagePullPolicy** and the tag of the image are affected. Here are some examples:

imagePullPolicy: **IfNotPresent** - The image gets pulled if it is not already present locally.

imagePullPolicy: **Always** - kubelet queries the container image registry to resolve the name to an image digest.

imagePullPolicy: **Never** - The image is never assumed to exist locally.

Using kubectl

The **kubectl apply** maintains the changes applied to a live object. kubectl apply follows declarative management.

1 To create single-container deployments and services, use **kubectl create deployment** and **kubectl expose.**

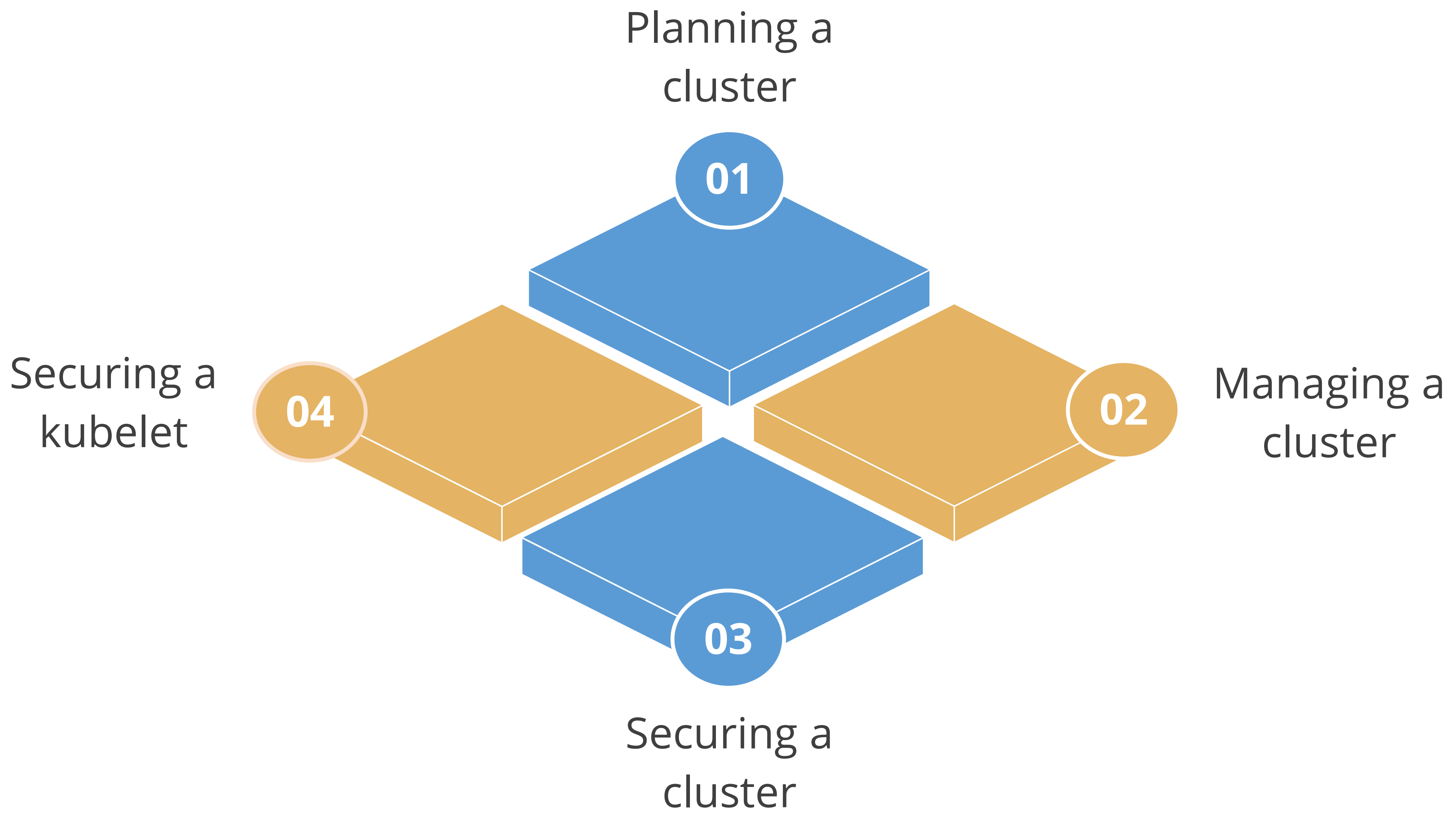
2 For **get** and **delete** operations, use label selectors and not specific object names.

TECHNOLOGY

Managing and Administering Clusters

Overview

Cluster administration and management involves four steps:



Optional Cluster Services

The following are optional for Kubernetes Clusters:

DNS Integration

1

Logging and Monitoring Cluster Activity

2

Managing and Administering a Kubernetes Cluster

Duration: 5 Mins

Problem Statement:

You’ve been asked to manage and administer a Kubernetes Cluster.

Assisted Practice: Guidelines

Steps to be followed:

1. Verify the certificates of the cluster
2. View the cluster information
3. Create a namespace
4. Access clusters using Kubernetes API

TECHNOLOGY

Node

Node

To run the workload, Kubernetes places containers in Pods and runs them on Nodes. A Node can be a physical or a virtual machine.

A cluster can have several Nodes.

The Control Plane manages every Node.

A Node is made up of the kubelet, a container runtime, and the kube-proxy.

Management

There are two main ways to add nodes to the API server:

1. Self-register
2. Manually add the nodes

Management

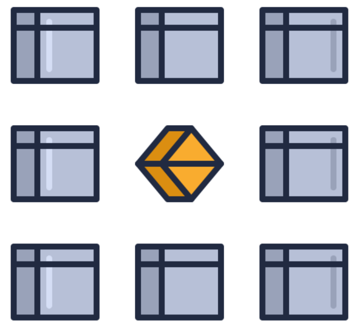
To create a Node from a JSON manifest, use:

Example

```
{
  "kind":  "Node",
  "apiVersion": "v1"
  "metadata":         {
    "name":  "10.240.79.157"
    "label":             {
      "name":   "myfirst-k8s-node"
                         }
                         }
  }
```

Node Name Uniqueness

The Node name identifies a Node and must, therefore, be unique. Two Nodes cannot have the same name at the same time.



1. Kubernetes assumes that a resource with the same name is the same object.
2. An instance using the same name is assumed to have the same state.

Self-Registration of Nodes

When the kubelet flag **--register-node** is true (the default), the kubelet will attempt to register itself with the API server. To accommodate self-registration, use the following options to start the kubelet:

--kubeconfig

Set a path to credentials to authenticate itself to the API server

--register-with-taints

Register the Node with the given list of taints

--cloud-provider

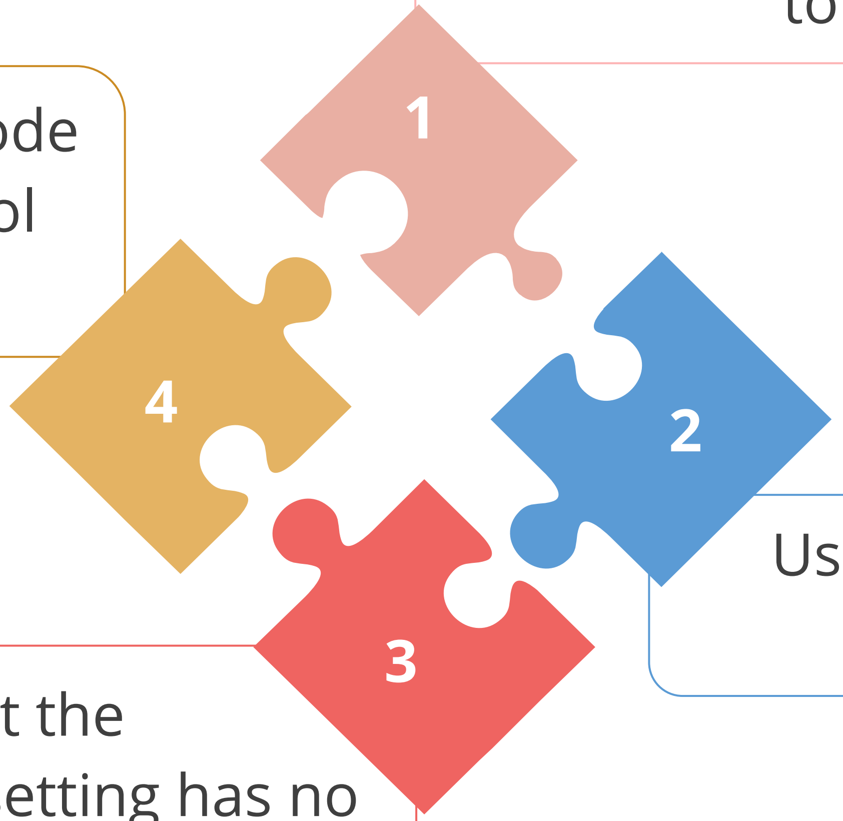
Talk to a cloud provider to read metadata about itself

--register-node

Register the Node automatically with the API server

Manual Node Administration

Node objects can be created and modified manually.



1. Set kubelet **flag --register-node** to **false**
2. Use Kubectl to create and modify Node objects
3. Note that the **--register-node** setting has no effect on modification of Node objects
4. Use labels on Nodes with Node Selectors on Pods to control scheduling

Node Status

A Node's status contains the following information:

- Addresses
- Conditions
- Capacity and allocable
- Info

To view the status of a Node, use:

```
kubectl describe node <insert-node-name-here>
```

Addresses

The output of the **kubectl describe node** can have three fields. Based on the cloud provider or the bare metal configuration, the usage of these fields will differ:

HostName

The hostname, as reported by the node's kernel, that can be overridden via the kubelet --hostname-override parameter

External IP

The IP address of the node that is externally routable

Internal IP

The IP address of the node that is routable only within the cluster

Conditions

The conditions field describes the status of all running nodes. Some examples of conditions are given below:

Node Condition	Description
Ready	True if the node is healthy, ready to accept Pods; False if the node is not healthy, not accepting pods; Unknown if there is no response during the last node-monitor-grace-period
DiskPressure	True if pressure exists on the disk size, that is if the disk capacity is low; otherwise, false
MemoryPressure	True if pressure exists on the node memory, that is, if the node memory is low; otherwise, false
PIDPressure	True if pressure exists on the processes, that is, if there are too many processes on the node; otherwise, false
NetworkUnavailable	True if the network for the node is not correctly configured; otherwise, False

Conditions

Node condition is represented as a JSON object. The structure below describes a healthy node:

Example

```
"conditions": [
{
type": "Ready",
"status":  "True",
"reason": "KubeletReady",
"message": "kubelet is posting ready status",
"LastHeartbeatTime": "2019-06-05T18:38:352",
"LastTransitionTime" "2019-06-05T11:41:272"
}

]
```

Capacity and Allocatable Blocks

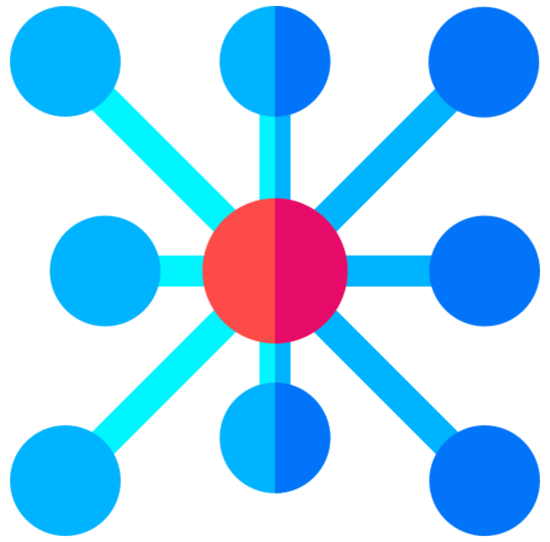
Capacity and allocatable blocks describe the resources available on the node. The resources include CPU, memory, and the maximum number of Pods that can be scheduled on the node.

Capacity block specifies the total number of resources present in the node.

Allocatable block specifies the number of available resources on a node.

Info

Info provides general information about the node, such as kernel version, Kubernetes version, Docker version, and OS name.



kubelet is used to gather information about a node.

Heartbeats

Heartbeats are used to determine the availability of a node. Heartbeats can be the updates of nodestatus or the Lease object.

Kubelet updates the NodeStatus and the Lease object.

1. The **NodeStatus** gets updated when there is a change in status or if there has been no update for a configured interval.

2. **Lease** objects are created and updated every 10 seconds.

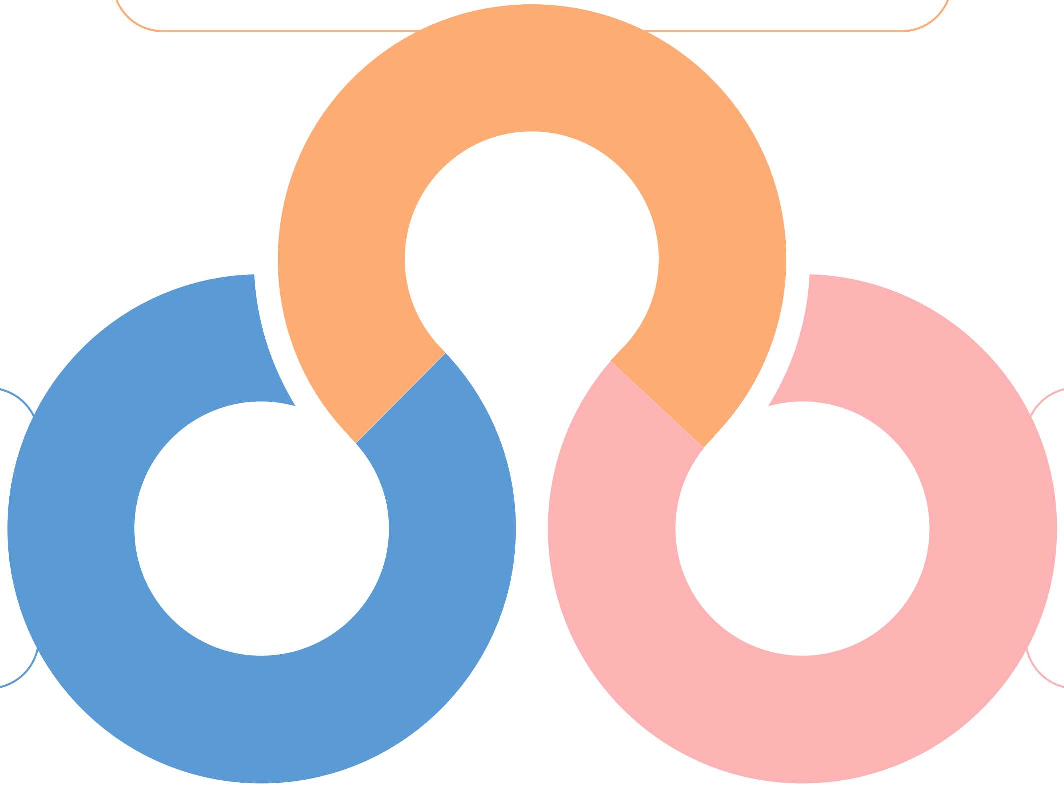
Node Controller

Node controller is a Kubernetes control plane component that manages nodes.

Keeps the node controller's internal list of nodes up-to-date

Assigns a CIDR block to the node when it is registered

Monitors the node’s health



Rate limits on Eviction

The node eviction behavior changes when a node becomes unhealthy. Following are a few scenarios that trigger a change in eviction behavior:

01 Eviction rate is reduced if the fraction of unhealthy nodes is at the least **--unhealthy-zone-threshold.**

02 Eviction rate is stopped if the cluster is small (≤ **--large-cluster-size-threshold** nodes). Otherwise, the eviction rate is reduced to **--secondary-node-eviction-rate.**

Node’s Resource Capacity

The node objects track information about the resource capacity of a node.

1. **Self-registering nodes:** The node capacity is reported when the node is registered.

2. **Manually added nodes:** The node capacity must be set when the node is added.

Graceful Node Shutdown

Kubernetes supports the Graceful Node feature. This feature detects a node system shutdown and terminates Pods running on the node.

The Graceful Node shutdown feature depends on the system.

Graceful Node shutdown is controlled using the **GracefulNodeShutdown** feature gate.

Graceful Node Shutdown

Kubelet carries out the termination process in the two phases:

1. Terminates regular Pods running on the node

2. Terminates critical Pods running on the node

Understanding the Working of Nodes

Duration: 5 Mins

Problem Statement:

You’ve been assigned a task to understand the working of nodes.

Assisted Practice: Guidelines

Steps to be followed:

1. Verify the status of a node
2. Delete a worker node
3. Register a worker node using a config file

Understanding and Deploying the First Pod

Duration: 15 mins.

Problem Statement:

You’ve been tasked with understanding and deploying the first Pod.

Assisted Practice: Guidelines

Steps to understand and deploy the first Pod:

1. Add the given code in Firstpod.yaml
2. Create the Deployment using a command
3. List the Pods available by using a command
4. Access the Apache Pod by using the given commands

Guidelines for Creating a Pod with Service

Duration: 10 mins.

Problem Statement:

You've been asked to deploy a Pod and assign Service to it using the label selector.

Assisted Practice: Guidelines

Steps to be followed:

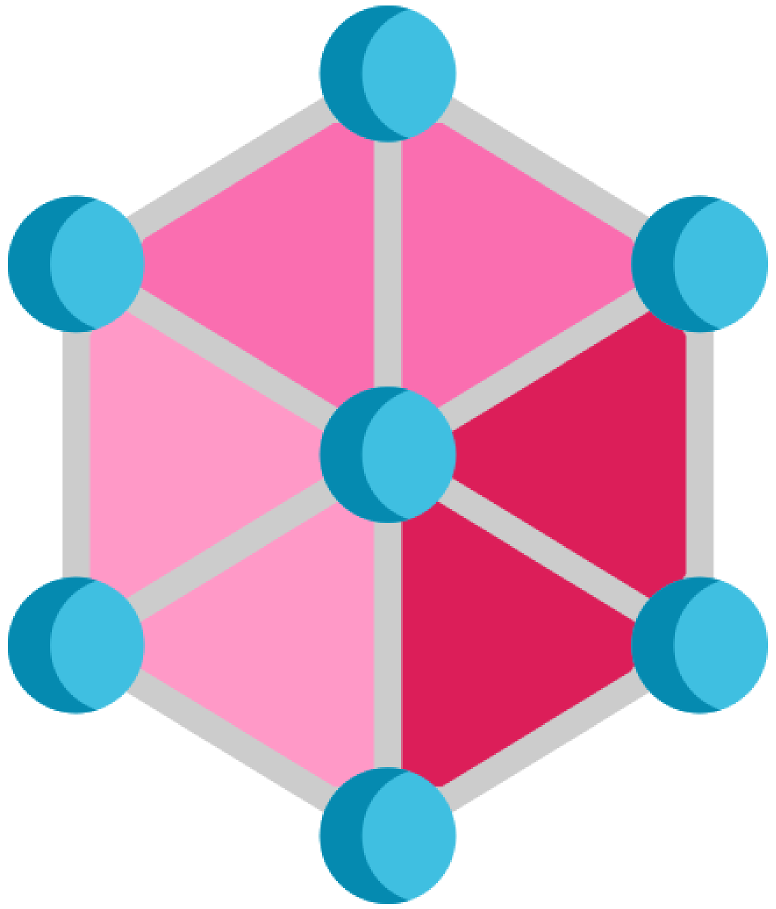
1. Creating a Deployment object
2. Creating a Service for Deployment using label selector

TECHNOLOGY

Control Plane Node Communication

Overview

The control plane catalogs the communication paths between the control plane and the Kubernetes cluster. This helps users to customize the installation and harden network configuration.



Communication from Node to Control Plane

Kubernetes has a hub and spoke pattern, which ensures that all API usage from nodes terminates at the API server.

Step 01

The API server is configured to listen for remote connections on a secure HTTPS port with one or more forms of client authentication enabled.

Step 02

Nodes should be provisioned with a public root certificate for the cluster in such a way that they can connect securely to the API server with valid credentials.

Step 03

The components of the control plane communicate with the cluster API server over the secure port.

Control Plane to Node

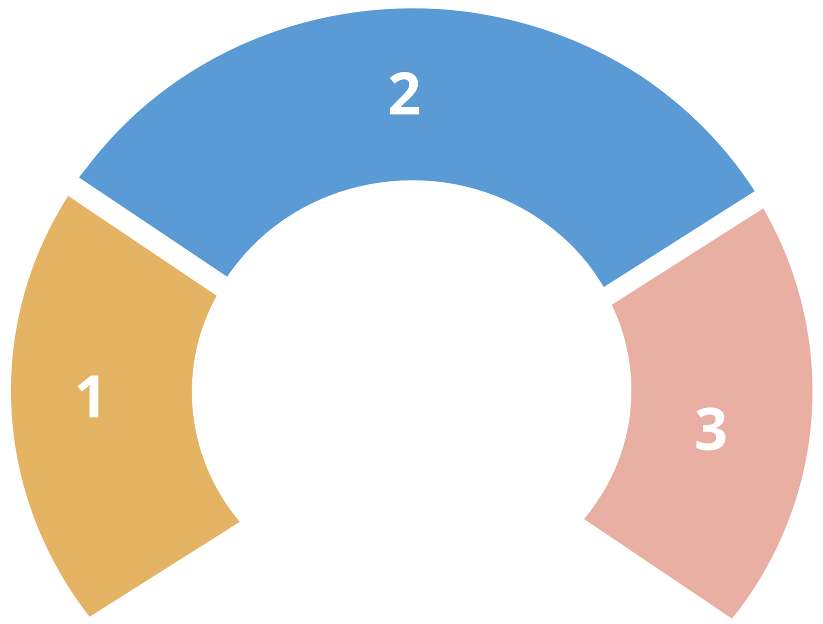
The communication from control plane to nodes can follow two paths:

1. From the API server to the kubelet process

2. From the API server to any node, Pod, or service

Connection from API Server to kubelet

The connections from the API server to the kubelet serve three purposes:



Attaching (through kubectl) to running Pods

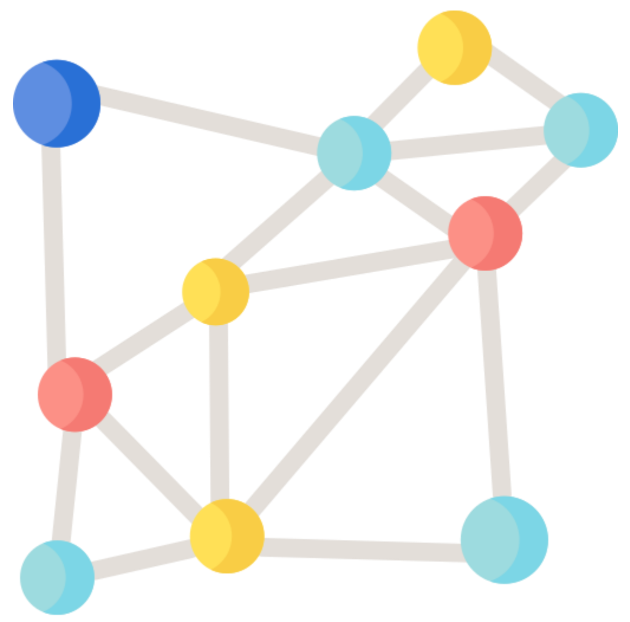
Fetching logs for Pods

Providing kubelet's
port-forwarding functionality

Connection from API Server to Nodes, Pods, and Services

The API server can connect to a node, Pod, or service. These connections that default to a plain HTTP connection are neither encrypted nor authenticated.

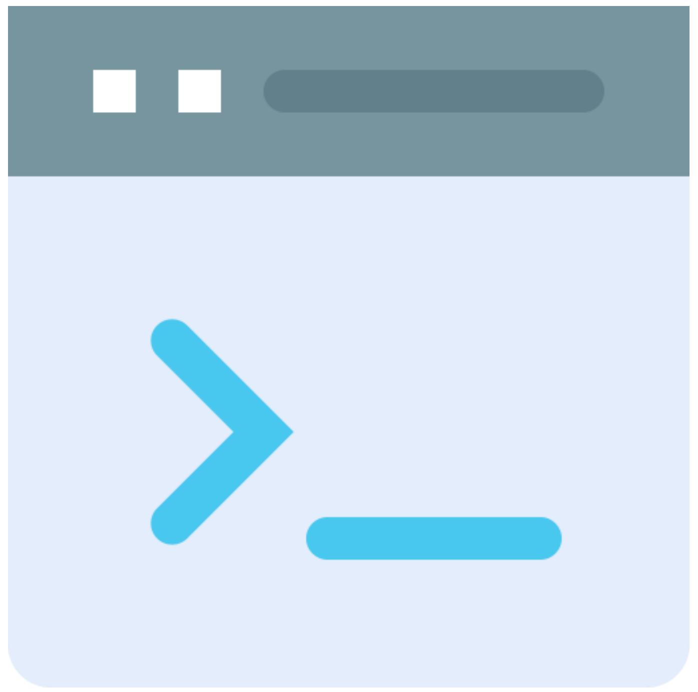
The connections can run over a secure HTTPS. This is done by prefixing HTTPS: to the node, Pod, or service name in the API URL.



Connections do not validate the certificate provided by the HTTPS endpoint. They do not provide client credentials.

SSH Tunnels

Kubernetes uses SSH tunnels to protect the control plane to help the communication paths between the nodes. They are currently deprecated and, hence, the usage should be minimal.



Konnectivity service is used in place of SSH tunnels.

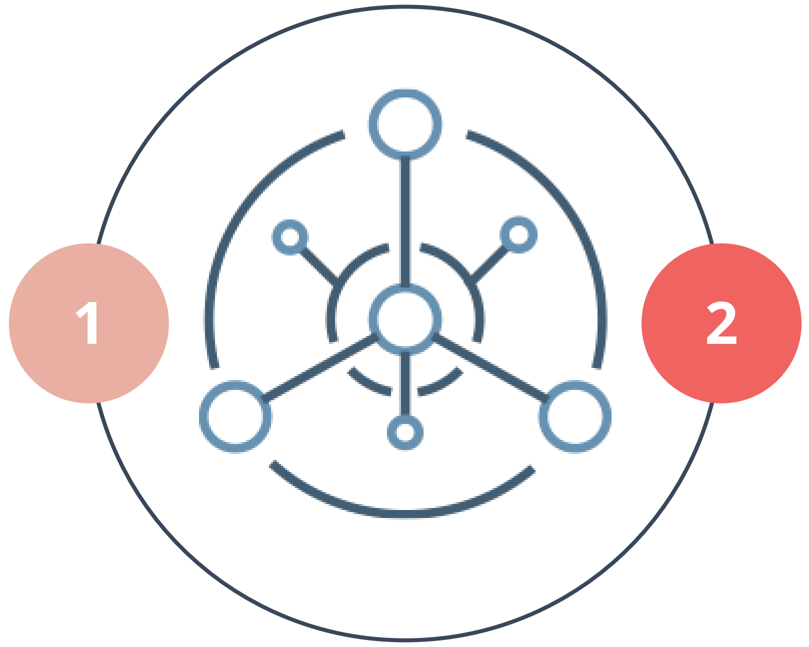
Konnectivity Service

Konnectivity service provides a TCP level proxy for the control plane to cluster communication.

It consists of two parts:

1 Konnectivity server

2 Konnectivity agents



TECHNOLOGY

Understanding Controllers

Controller Pattern

Controllers are control loop that watches the state of the cluster, then make or requests changes where needed.

Tracks at least one Kubernetes resource type

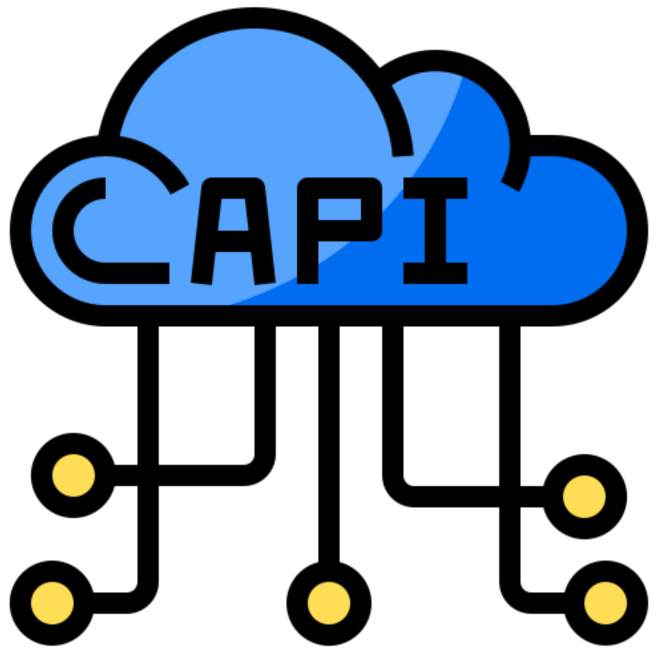
Represents the desired state of the objects

Moves the resource’s current state come closer to that desired state

Control via API Server

Built-in Controllers interact with the cluster API server and manage the system state.

Job Controller is an example of a Kubernetes built-in Controller.



Controllers can update the objects that are used to configure them.

Direct Control

Direct control is employed when Controllers need to make changes to things outside of the cluster.

Controllers find the desired state from the API server.

Controllers communicate directly to bring the current state closer in line.

Running Controllers

There are two ways of running Controllers:

Inside the kube-controller-manager

1

Outside the kube-controller-manager

2

TECHNOLOGY

Importance of Cloud Controller Manager

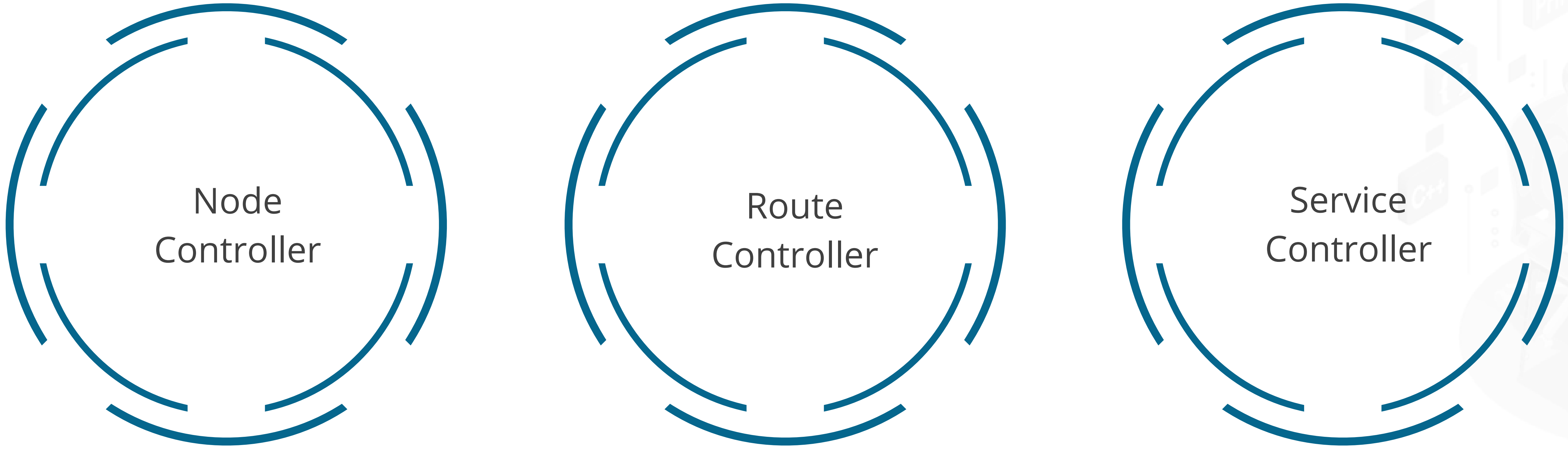
Cloud Infrastructure

Cloud infrastructure technologies:

- ✓ Help run Kubernetes on public, private, and hybrid clouds
- ✓ Embed cloud-specific control logic to run on different cloud platforms
- ✓ Are structured using a plugin mechanism that allows different cloud providers to integrate their platforms with Kubernetes

Cloud Controller Manager Types

There are many types of controllers inside the Cloud Controller Manager, each of which has a role to play. The three important controllers are as follows:



TECHNOLOGY

Working with Kubeadm

Kubeadm

The Kubeadm toolkit is used to bootstrap a best-practice Kubernetes Cluster. Kubeadm provides **kubeadm init** and **kubeadm join** to achieve this.

kubeadm

Kubeadm acts as an installer and a building block that helps to get a minimum viable cluster up and running.

Kubeadm Init Command Options

The **kubeadm init [flags]** command initializes and sets up a Kubernetes Control Plane Node. The various options available for this command are:

1. --apiserver-advertise-address
2. --apiserver-bind-port
3. --apiserver-cert-extra-sans
4. --cert-dir
5. --certificate-key
6. --config
7. --control-plane-endpoint
8. --cri-socket

Kubeadm Init Command Options

The **kubeadm init [flags]** command initializes and sets up a Kubernetes Control Plane Node. The various options available for this command are:

9 --dry-run

10 --experimental-patches

11 --feature-gates

12 --h, --help

13 --ignore-preflight-errors

14 --image-repository

15 --node-name

16 --pod-network-cidr

Kubeadm Init Command Options

The **kubeadm init [flags]** command initializes and sets up a Kubernetes Control Plane Node. The various options available for this command are:

17 --service-cidr

18 --service-dns-domain

19 --skip-certificate-key-print

20 --skip-phases

21 --skip-token-print

22 --token

23 --token-ttl duration

24 --upload-certs

Init Workflow

To bootstrap a Kubernetes control plane, kubeadm init follows these steps:

Step 1

Run a series of preflight checks to validate the system state before making changes

Step 2

Set up identities for each cluster component by generating a self-signed CA

Step 3

Write kubeconfig files in /etc/kubernetes/ for the kubelet and controller-manager

Step 4

Generate static Pod manifests for the API server, controller-manager, and scheduler

Init Workflow

To bootstrap a Kubernetes control plane, kubeadm init follows these steps:

Step 5

Apply labels and taints to the Control Plane Node

Step 6

Generate the token that additional Nodes can use to register themselves with a Control Plane

Step 7

Make necessary configurations for allowing Node joining with the Bootstrap Tokens and TLS

Step 8

Install a DNS server and the Kube-proxy addon components through the API server

Using Init Phases with kubeadm

To create a control plane in phases, use **kubeadm init phase** command:

```
sudo kubeadm init phase control-plane controller-manager --help
```

Certain control plane phases have unique flags. To get a list of available options, add **–help.** Here's an example:

```
sudo kubeadm init phase control-plane --help
```

Using Custom Images

By default, **kubeadm** pulls images from **k8s.gcr.io**. If the requested Kubernetes version is a CI label, **gcr.io/k8s-staging-ci-images** is used.

A configuration file helps override this behavior. Kubernetes allows the following customization:

To provide **kubernetesVersion** which affects the version of the images.

To provide an alternative **imageRepository** to be used instead of **k8s.gcr.io**.

To provide a specific **imageRepository** and **imageTag** for etcd or CoreDNS.

Setting the Node Name

Based on the machine’s host address, **kubeadm** assigns a node name. **--node-name** flag may be used to override this setting.

--node-name flag passes the appropriate **--hostname-override** value to the kubelet.

Running kubeadm Without Internet Connection

To run kubeadm without an internet connection, the required control plane images must be pre-pulled.

The images can be listed and pulled using the **kubeadm config images** sub-command:

Demo

```
kubeadm config images list
kubeadm config images pull
```

Kubeadm Join

The **kubeadm join** command initializes and joins a Kubernetes worker node to the cluster. The **kubeadm join [api-server-endpoint] [flags]** command supports the following options:

1. --apiserver-advertise-address
2. --apiserver-bind-port
3. --certificate-key
4. --config
5. --control-plane
6. --cri-socket
7. --discovery-file
8. --discovery-token

Kubeadm Join

The **kubeadm join** command initializes and joins a Kubernetes worker node to the cluster. The **kubeadm join [api-server-endpoint] [flags]** command supports the following options:

9 --discovery-token-ca-cert-hash

10 --experimental-patches

11 --discovery-token-unsafe-skip-ca-verification

12 --h, --help

13 --ignore-preflight-errors

14 --node-name

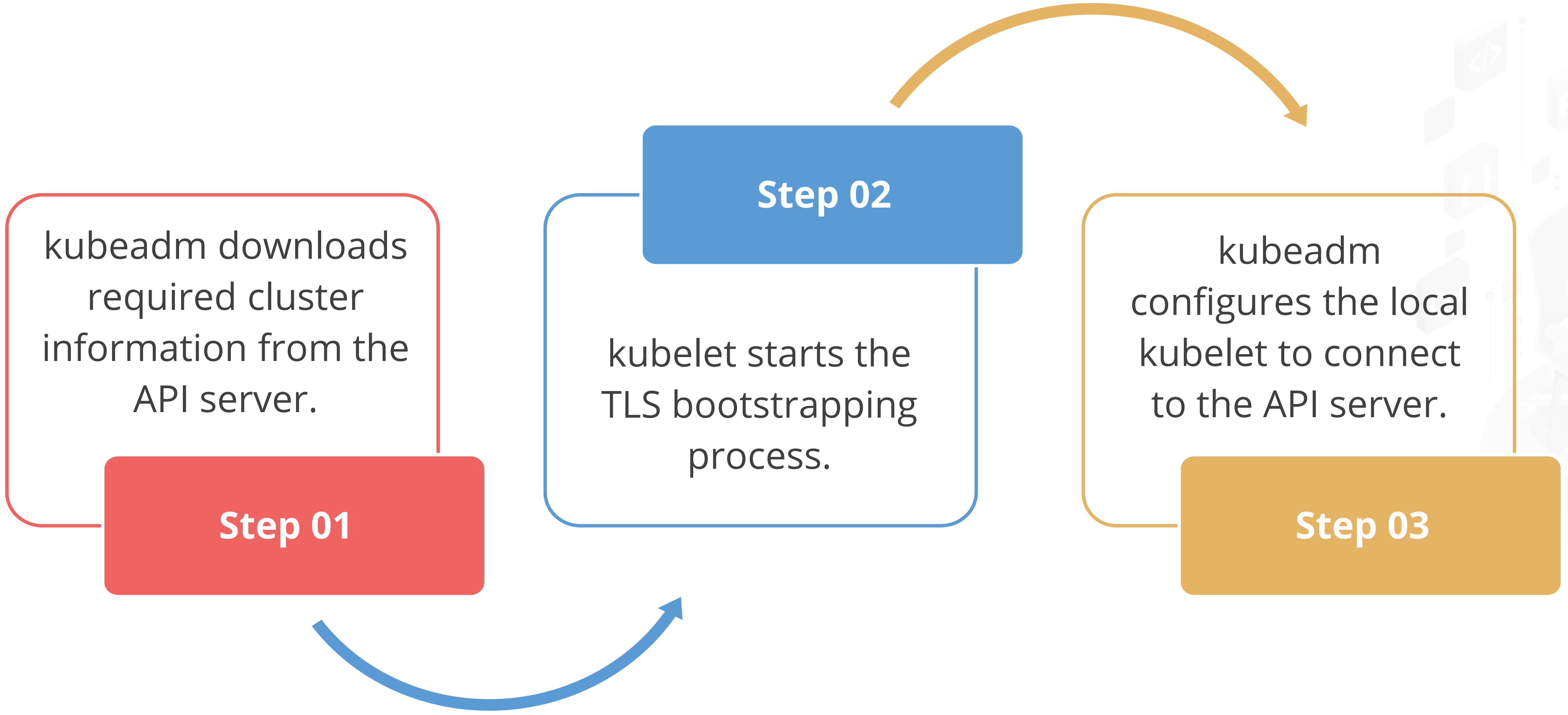
15 --skip-phases

16 --tls-bootsrap-token

17 --token

The Join Workflow

kubeadm join bootstraps a Kubernetes worker node or a control plane node and adds it to the cluster. This action is completed in three steps for worker nodes which are as follows:



Step 01

kubeadm downloads required cluster information from the API server.

Step 02

kubelet starts the TLS bootstrapping process.

Step 03

kubeadm configures the local kubelet to connect to the API server.

Using Join Phases with kubeadm

Kubeadm allows users to join a node to the cluster in phases using **kubeadm join phase**.

Certain phases have unique flags. To get a list of available options, add **--help.**

An example is given below:

Example

```
kubeadm join phase kubelet-start --help
```

Working with Kubeadm

Duration: 5 mins

Problem Statement:

You’ve been asked to generate tokens and manage certificates using kubeadm.

Assisted Practice: Guidelines

Steps to be followed:

1. Generating tokens for kubeadm
2. Managing kubernetes certificates
3. Viewing the configuration details

Kubernetes Dashboard

Duration: 10 mins.

Problem Statement:

You’ve been assigned a task to deploy and access Kubernetes dashboard.

Assisted Practice: Guidelines

Steps to be followed:

1. Deploying the dashboard
2. Verifying the pods, services, and deployments
3. Editing the service type of the dashboard
4. Verifying the service type of the dashboard
5. Checking where the pod is running
6. Copying the IP and NodePort
7. Selecting desktop form taskbar from the master node
8. Opening the link on Firefox browser
9. Accessing the dashboard

TECHNOLOGY

Managing a Cluster Using Kubelet

simplilearn

Managing Clusters

The kubelet is the primary **node agent** that runs on each node. It can register the node with the API server using a hostname, a flag to override the hostname or specific logic for a cloud provider.

A container manifest may be provided to the Kubelet using the following:

- PodSpec
- File
- HTTP endpoint
- HTTP server

Options

There are non-deprecated and non-alpha version options available with the current version of Kubernetes:

--add-dir-header	--cert-dir string
--alsologtostderr	--config
--azure-container-registry-config	--container-runtime
--bootstrap-kubeconfig	--container-runtime-endpoint
--docker-endpoint	--dynamic-config-dir

Options

There are non-deprecated and non-alpha version options available with the current version of Kubernetes:

- --enable-controller-attach-detach
- --exit-on-lock-contention
- -h, --help
- --hostname-override
- --housekeeping-interval
- --image-credential-provider-bin-dir
- --image-pull-progress-deadline
- -- kubeconfig
- -- log-backtrace-at traceLocation
- --log-dir
- --log-file
- --log-file-max-size
- --log-flush-frequency
- -- logtostderr

Options

There are non-deprecated and non-alpha version options available with the current version of Kubernetes:

- --image-service-endpoint
- -- one-output
- --pod-infra-container-image
- -register-node
- --register-with-taints
- -- root-dir
- -- node-ip
- -- runtime-cgroups
- -- skip-headers
- --skip-log-headers
- -- version
- -- vmodule

Kubelet Authentication

Requests to the kubelet's HTTPS endpoint that are not rejected by other configured authentication methods are treated as anonymous requests.

They will be given a username **system:anonymous** and a group **system:unauthenticated**.

To disable anonymous access and send **401 Unauthorized** responses to unauthenticated requests:

Start the kubelet with the **--anonymous-auth=false** flag

Kubelet Authentication

To **enable X509 client certificate authentication** to the kubelet's HTTPS endpoint:

- Start the kubelet with the **--client-ca-file** flag, providing a CA bundle to verify client certificates with
- Start the apiserver with **--kubelet-client-certificate** and **--kubelet-client-key** flags

Kubelet Authentication

Here are a few steps to **enable API bearer tokens** to be used to authenticate to the kubelet's HTTPS Endpoint:

1. Ensure the **authentication.k8s.io/v1beta1** API group is enabled in the API server

2. Start the kubelet with the **--authentication-token-webhook** and **–kubeconfig** flags

3. The kubelet calls the **TokenReview** API on the configured API server to determine user information from bearer tokens

Kubelet Authorization

Any request that is successfully authenticated is then authorized. The default authorization mode is **AlwaysAllow**, which allows all requests.

Access to the kubelet API can be subdivided. Subdivisions occur under the following conditions:

- Anonymous auth is enabled, but limits the ability of anonymous users to call the kubelet API
- Bearer token auth is enabled, but limits the ability of arbitrary API users to call the kubelet API
- Client certificate auth is enabled, but only some of the client certificates must be allowed to use the kubelet API

Kubelet Authorization

To subdivide access to the kubelet API, delegate authorization to the API server:

Start the kubelet with the **--authorization-mode=Webhook** and the **--kubeconfig** flags

Ensure the **authorization.k8s.io/v1beta1** API group is enabled in the API server

The kubelet calls the **SubjectAccessReview** API on the configured API server to determine whether each request is authorized.

Kubelet Authorization

To authorize API requests using request attributes, kubelet adopts an approach similar to the API server. The verb is determined from the incoming request’s HTTP web.

HTTP verb	Request verb
POST	create
GET.HEAD	get
PUT	update
PATCH	patch
DELETE	delete

TECHNOLOGY

Role-Based Access Controller

RBAC

Role-based access controller (RBAC) is used to configure fine-grained and specific sets of permissions within a Kubernetes cluster.

Extensions or declarations can define roles and permissions.

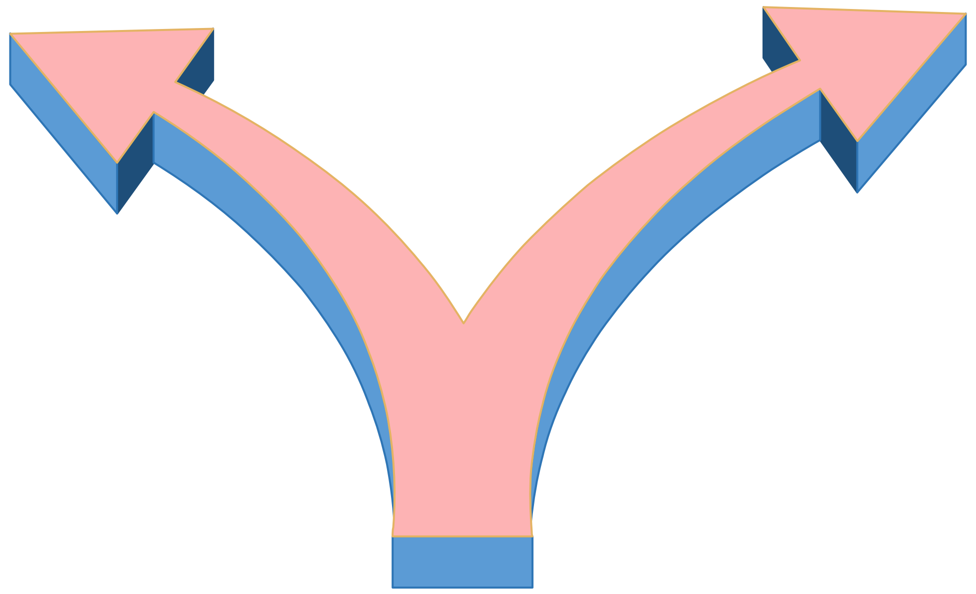
Subject	Operations	Resources
User	List, get, create, update, delete, watch, patch, post, put	Pods, nodes, configmaps, secrets, deployment
Group		
Service account		

Permissions are purely additive (there are no ”deny” rules).

Operations and Subjects

Access control connects operations and subjects.

HTTP verbs sent to the API represent the operations on resources.



Subjects are the actors in the Kubernetes API and RBAC.

Resource vs. Non-Resource Requests

The Kubernetes API server can add additional attributes as a resource or a non-resource request. Some of the attributes that can be added are:

Request Type	Attribute	Description
Authentication	User	The user string
	Group	The list of group names
	Extra	A map of arbitrary keys
API	Non-resources or API resources flag	

Resource vs. Non-Resource Requests

Request Type	Attribute	Description
API resource request	API request verb	Lowercase resource verb
	Namespace	The namespace
	API group	The API group being accessed
	Resource	The resources ID
	Subresource	The subresources
Non-resource request	HTTP request verb	Lowercased HTTP method
	Request path	Non-resources request path
Verbs	Common API resource request	Get, list, watch, create, update, patch, delete, delete collection
	Special API resource request	Use, bind, escalate, impersonate, userextras

Authentication Methods for Kubernetes

There are several authentication mechanisms available in Kubernetes. Some of these include:

- X509 Client Certs
- Bearer Token
- Static Token File
- Bootstrap Tokens
- Service Account

Kubernetes Dashboard Role-Based Access Control (RBAC)

Duration: 05 mins

Problem Statement:

You've been asked to implement Role-based access control (RBAC) authorization on the Kubernetes dashboard.

Assisted Practice: Guidelines

Steps to be followed:

1. Adding, Deleting, and Verifying Cluster Roles

RBAC Using Namespace

Duration: 15 mins

Problem Statement:

You've been asked to implement Role-based access control (RBAC) using Namespace.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

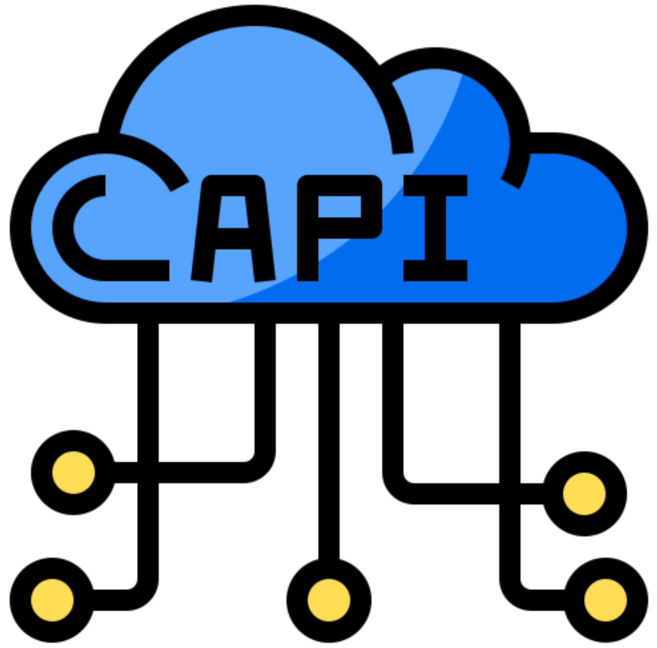
1. Creating a Namespace
2. Generating an RSA private key and certificate requests
3. Creating Role
4. Creating Rolebinding
5. Setting credentials to user
6. Copying the config file to client machine
7. Verifying roles

TECHNOLOGY

API Server

Overview

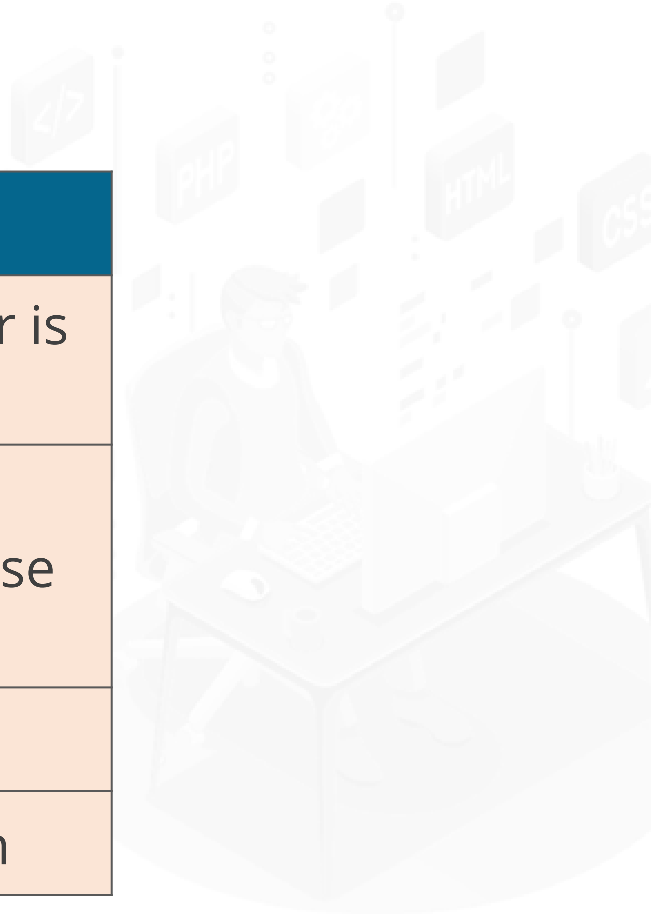
The API server is the core of the Kubernetes control plane. It exposes an HTTP API that lets end users, clusters, and external components communicate with one another.



OpenAPI Specification

The Kubernetes API server uses **/openapi/v2** endpoint to provide OpenAPI v2 spec. The response format can be requested using request headers as shown below:

Header	Possible values	Notes
Accept-Encoding	gzip	Not supplying this header is also acceptable
Accept	application/com.github.proto-openapi.spec.v2@v1.0+protobuf	Mainly for intra-cluster use
	application/json	Default
	*	Serves application/json



API Groups and Versioning

Kubernetes supports multiple API versions that help to restructure resource representations and eliminate fields.

API group, resource type, namespace, and name distinguish the API resources.

API Changes

Kubernetes is designed to adapt to changes.

1

Supports the addition of new API resources and resource fields

2

Provides compatibility for official Kubernetes APIs

API Extension

The Kubernetes API can be extended in one of two ways:

1. Using custom resources
2. Implementing an aggregation layer

Achieving High Availability

Setup

A highly available Kubernetes cluster can be set up with kubeadm by applying two different approaches, namely, using **stacked control plane nodes** and an **external etcd cluster**.

Steps

1. Create a kube-apiserver load balancer, the name of the load balancer must resolve to DNS
2. Test the connection after adding the first control plane node to the load balancer
3. Add the remaining control plane nodes to the load balancer target group

Stacked Control Plane and etcd Nodes

Steps for the first control plane node:

1 Initializing control plane

2 Applying CNI plugin

3 Running a command to start the Pods of the control plane components

Stacked Control Plane and etcd Nodes

To set the Kubernetes version, use **--kubernetes-version** flag. It is recommended that **kubeadm**, **kubelet**, **kubectl,** and **Kubernetes** versions should match.

The **--control-plane-endpoint** flag should be set to the address or DNS and port of the load balancer.

Command to initialize the control plane is given below:

```
sudo kubeadm init --control-plane-endpoint "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT" --upload-certs
```

Stacked Control Plane and etcd Nodes

To upload the certificates to be shared across all the control plane instances, **--upload-certs** flag is used.

Use the following command to re-upload the certificates and generate a new decryption key on a control plane node that is joined to the cluster:

```
sudo kubeadm init phase upload-certs --upload-certs
```

Stacked Control Plane and etcd Nodes

To apply the CNI, the CNI provider configuration must correspond to the Pod CIDR specified in the kubeadm configuration file. For instance, to apply Weave Net CNI, use the following command:

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n’)”
```

To watch the Pods of the control plane components, use the following command:

```
kubectl get pod -n kube-system -w
```

Setting Up Additional Control Plane Nodes

Execute the join command that is generated by kubeadm when the init command on the first node is run. The command is shown below:

```
sudo kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26aec866 --control-plane --certificate-
key f8902e114ef118304e561c3ecd4d0b543adc226b7a07f675f56564185ffe0c07
```

External etcd Nodes

Setting up a cluster with external etcd nodes differs from setting up with stacked etcd in one aspect.

Steps

1. Set up etcd; pass the etcd information in the kubeadm config file
2. Set up the first control plane node
3. Set up additional control plane nodes as required

Setting Up etcd Cluster

To set up the etcd cluster, set up SSH and copy the files given below from any etcd node in the cluster to the first control plane node:

Example

```
export CONTROL_PLANE="ubuntu@10.0.0.7"

scp /etc/kubernetes/pki/etcd/ca.crt "${CONTROL_PLANE}";

scp /etc/kubernetes/pki/apiserver-etcd.client.crt "${CONTROL_PLANE}";

scp /etc/kubernetes/pki/apiserver-etcd.client.key "${CONTROL_PLANE}";
```

Then, the value of **CONTROL_PLANE** must be replaced with **user@host** of the first control plane node.

Setting Up First Control Plane Node

kubeadm-config.yaml with the content shown below must be created:

Example

```
apiversion: kubeadm.k8s.io/v1beta2
Kind: clusterConfiguration
kuberenetesVersion: stable
controlPlaneEndpoint: “LOAD_BALANCER_DNS:LOAD_BALANCER_PORT”
etcd:
        external:
        endpoint:
        https://etcd_0_IP:2379
         https://etcd_1_IP:2379
         https://etcd_2_IP:2379
        caFile: /etc/kubernetes/pki/etcd/ca.crt
        certfilr: /etc/kubernetes/pki/apiserver-etcd-client.crt
        keyfile: /etc/kubernetes/pki/apiserver-etcd-client.key
```

Setting Up First Control Plane Node

The following variables in the config template must be replaced with the appropriate values for the cluster:

Example

```
LOAD_BALANCER_DNS
LOAD_BALANCER_PORT
etcd_0_IP
etcd_1_IP
etcd_2_IP
```

Setting Up First Control Plane Node

Once the **.yaml** file is created, there are three steps to be followed:

1. On the node, run **sudo kubeadm init --config kubeadm-config.yaml --upload-certs**
2. Write the output **join** commands that are returned to a text file
3. Apply the CNI plugin

Common Tasks After Bootstrapping Control Plane

Install workers

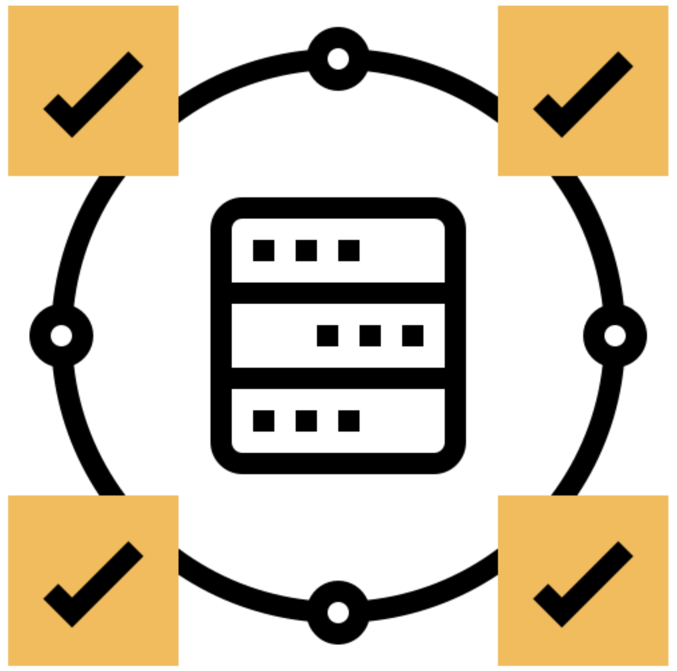
The command generated as the output of **kubeadm init** command must be used to install worker nodes to the cluster. This can be done as shown below:

```
sudo kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash
sha256:7c2e69131a36ae2a042a339b33381c6d0d43887e2de83720eff5359e26aec866
```

Common Tasks After Bootstrapping Control Plane

Manual Certificate Distribution

Certificates must be manually copied from the primary control plane node to the joining control plane nodes if **–upload-certs** flag is not used during initialization.



Manual Certificate Distribution

To distribute certificates using **ssh** and **scp:**

1. Enable ssh-agent on the main device that has access to all other nodes in the system
2. Add the SSH identity to the session
3. SSH between nodes to check that the connection is working correctly
4. Run the script to copy the certificates from the first control plane node to the others
5. Run the script on each control plane node to move the copied certificates from home directory to **/etc/kubernetes/pki** directory

TECHNOLOGY

Backup and Restoration of etcd Cluster Data

Operating etcd Clusters for Kubernetes

etcd is a highly-available key value store that is used as a backing store for all cluster data in Kubernetes.

To have a Kubernetes cluster that uses etcd as its backing store, a backup plan for the data is mandatory.

Note

Kubectl command-line tool must be configured to communicate with the cluster.

Prerequisites for etcd

1. Etcd needs to be run as a cluster of odd members
2. A distributed etcd system keeps the cluster stable
3. In an unstable etcd system, a cluster cannot make any changes to its current state
4. Keeping etcd clusters stable is critical to the stability of Kubernetes clusters
5. To run an etcd in production, the minimum recommended version is **3.2.10+**

Starting a Multi-Node etcd Cluster

The durability and availability of etcd significantly increase when it is run as a multi-node cluster.

To start a multi-node etcd cluster, run the following command:

```
etcd --listen-client-urls=http://$IP1:2379,http://$IP2:2379,http://$IP3:2379,http://$IP4:2379,http://$IP5:2379
--advertise-client-urls=http://$IP1:2379,http://$IP2:2379,http://$IP3:2379,http://$IP4:2379,http://$IP5:2379
```

Then, start the Kubernetes API server with the flag **--etcd-servers=$IP1:2379,$IP2:2379,$IP3:2379,$IP4:2379,$IP5:2379**.

Note

IP<n> variables must be set to the client IP addresses.

Multi-Node etcd Cluster with Load Balancer

To run a load balancing etcd cluster, follow the below three steps:

An etcd cluster must be set up.

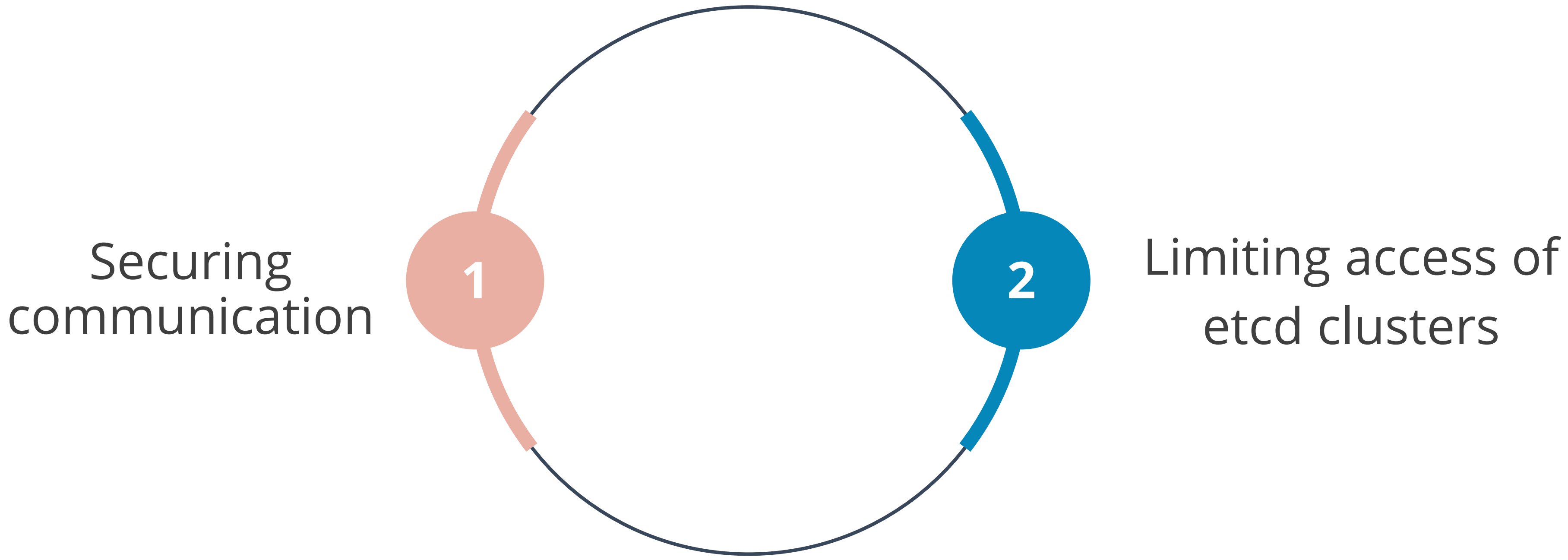
A load balancer must be configured.

The Kubernetes API servers must be started with the flag **--etcd-servers** set to **$LB:2379**.

Securing etcd Clusters

To secure etcd clusters, a firewall may be set up or etcd-provided security features may be used.

Securing etcd clusters involves:



Securing Communication

Configuring etcd with secure peer communication

Set flags **--peer-key-file=peer.key** and **--peer-cert-file=peer.cert** respectively and use **HTTPS** as the **URL** schema.

Configuring etcd with secure client communication

Set flags **--key-file= k8sclient.key** and **--cert-file= k8sclient.cert** respectively and use **HTTPS** as the **URL** schema.

Limiting Access of etcd Clusters

The access to etcd cluster must be restricted to the Kubernetes API servers through TLS authentication.

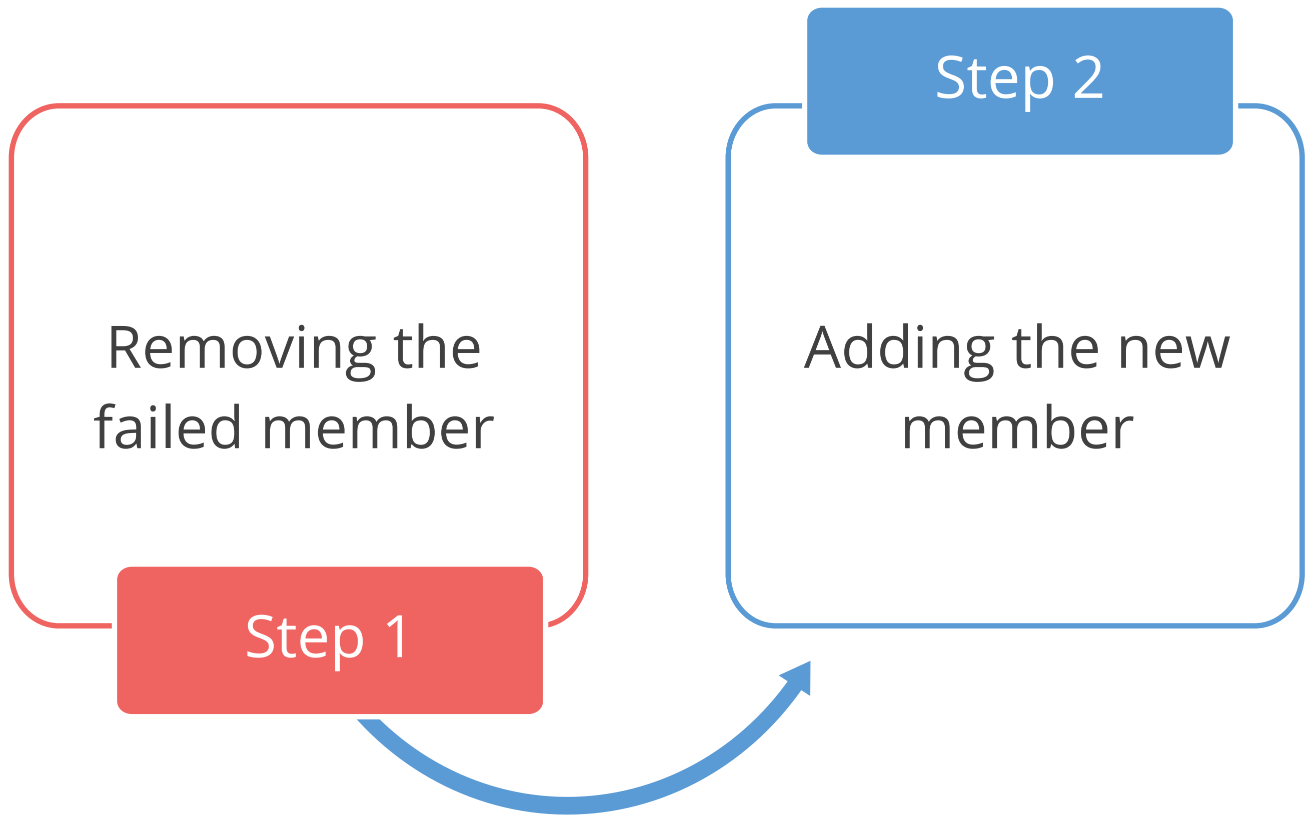
Example

- Consider a key pair **k8sclient.key** and **k8sclient.cert** trusted by the CA **etcd.ca**
- Etcd configured with **--client-cert-auth** and TLS, uses system CAs or the CA passed by **--trusted-ca-file** flag to verify the client certificates
- Setting **--client-cert-auth** and **--trusted-ca-file** to **true** and **etcd.ca** respectively restricts the access to clients **k8sclient.cert**

To provide access to Kubernetes API servers, set flags **--etcd-certfile,--etcd-keyfile**, and **--etcd-cafile** to **k8sclient.cert, k8sclient.key,** and **ca.cert** respectively.

Replacement of a Failed etcd Member

Failed members of a cluster must be replaced to improve the overall health of the cluster. Replacement of a failed member comprises two steps:



Replacing a Failed etcd Member

Example: Replacing a failed member in a three-member etcd cluster
member1=http://10.0.0.1,member2=http://10.0.0.2, and member3=http://10.0.0.3

Scenario: member1 fails and member4=http://10.0.0.4 gets added as a replacement

1. Get the member ID of the failed member
2. Remove the failed member
3. Add the new member
4. Start the newly added member on a machine with the IP 10.0.0.4
5. Update **--etcd-servers** flag and load balancer configuration

Backing Up an etcd Cluster

An etcd cluster can be backed up either by using a built-in snapshot or a volume snapshot.

etcd built-in snapshot

Snapshot supported by etcd can be taken from a live member or by copying the **member/snap/db** file from the etcd data directory

Volume snapshot

Snapshots provided by storage volumes on which etcd is running

Scaling Up etcd Clusters

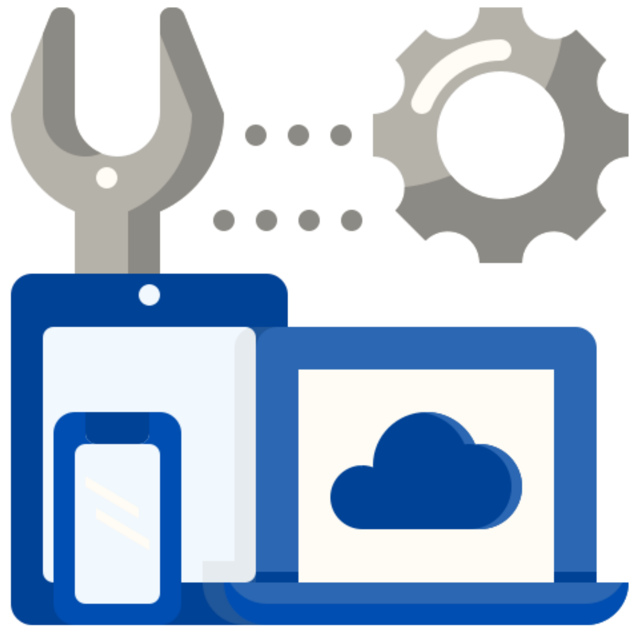
Scaling up etcd clusters increases cluster availability.

Scaling does not increase cluster performance or capability.

Restoring an etcd Cluster

Snapshots taken from an etcd process (major.minor version) restore an etcd cluster. Kubernetes supports restoring a version from a different patch version of etcd.

To recover data from a failed cluster, a restore operation must be employed.



Backing Up and Restoring Etcd Cluster Data

Duration: 10 mins

Problem Statement:

You’ve been assigned a task to back up and restore etcd cluster.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Backing up etcd cluster data
2. Retrieving etcd cluster data

TECHNOLOGY

Version Upgrade on Kubernetes Cluster

simplilearn

Overview

At a high level, the upgrade workflow of a Kubernetes cluster comprises the following:

Primary control plane node upgrade

Additional control plane nodes upgrade

Worker nodes upgrade

Determine Version to Upgrade

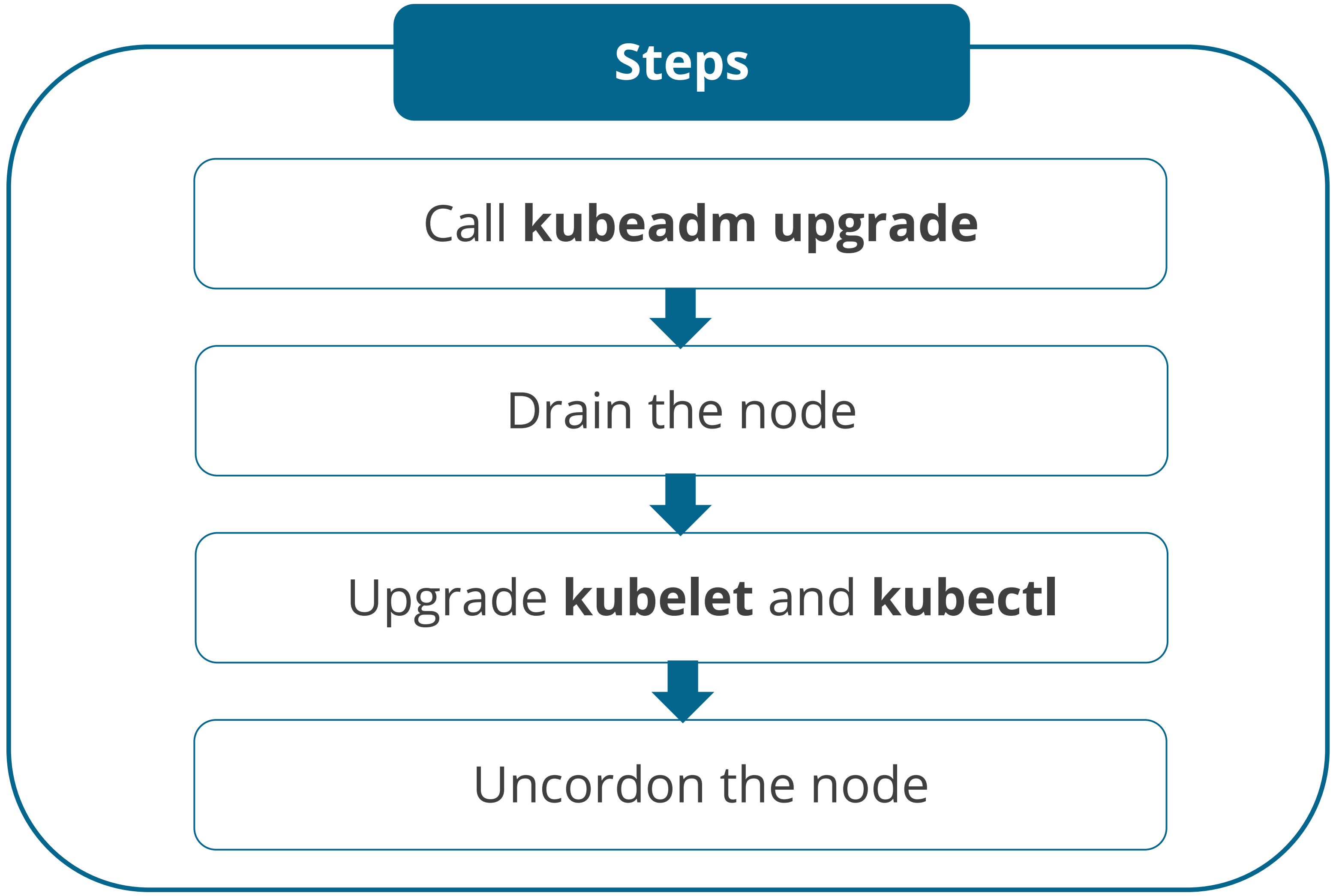
Enter the commands shown below to find the latest stable 1.23 version using the OS package manager:

Example

```
apt update
apt-cache madison kubeadm
# find the latest 1.23 version in the list
# it should look like 1.23.x-00, where x is the latest patch
```

Upgrading Control Plane Nodes

The control plane nodes must be updated one at a time. The node selected for upgrade must have the **/etc/kubernetes/admin.conf** file.



Upgrading Control Plane Nodes

Upgrade kubeadm for the first control plane node as shown below:

Example

```
# replace x in 1.23.x-00 with the latest patch version
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.23.x-00 && \
apt-mark hold kubeadm
-
# the following method can also be used.
apt-get update && \
apt-get install -y --allow-change-held-packages kubeadm=1.23.x-00
```

Upgrading Control Plane Nodes

Verify the working of kubeadm and check its version using the commands shown below:

Example

```
#Verify that the download works and has the expected version:
kubeadm version

#Verify the upgrade plan:
kubeadm upgrade plan
```

Upgrading Control Plane Nodes

Use this command for other control plane nodes:

Example

```
#For other control plane nodes, use:
sudo kubeadm upgrade node
#instead of:
sudo kubeadm upgrade apply
```

Upgrading Control Plane Nodes

To prepare the node for maintenance, mark the unschedulable and evict workloads:

Example

```
# replace <node-to-drain> with the name of your node you are draining
kubectl drain <node-to-drain> --ignore-daemonsets
```

To upgrade kubelet and kubectl:

Example

```
# replace x in 1.23.x-00 with the latest patch version
 apt-mark unhold kubelet kubectl && \
 apt-get update && apt-get install -y kubelet=1.23.x-00 kubectl=1.23.x-00 && \
 apt-mark hold kubelet kubectl
 -
 # the following method can also be used
 apt-get update && \
 apt-get install -y --allow-change-held-packages kubelet=1.23.x-00 kubectl=1.23.x-00
```

Upgrading Control Plane Nodes

To restart the kubelet, use the command:

Example

```
Restart the kubelet:

sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

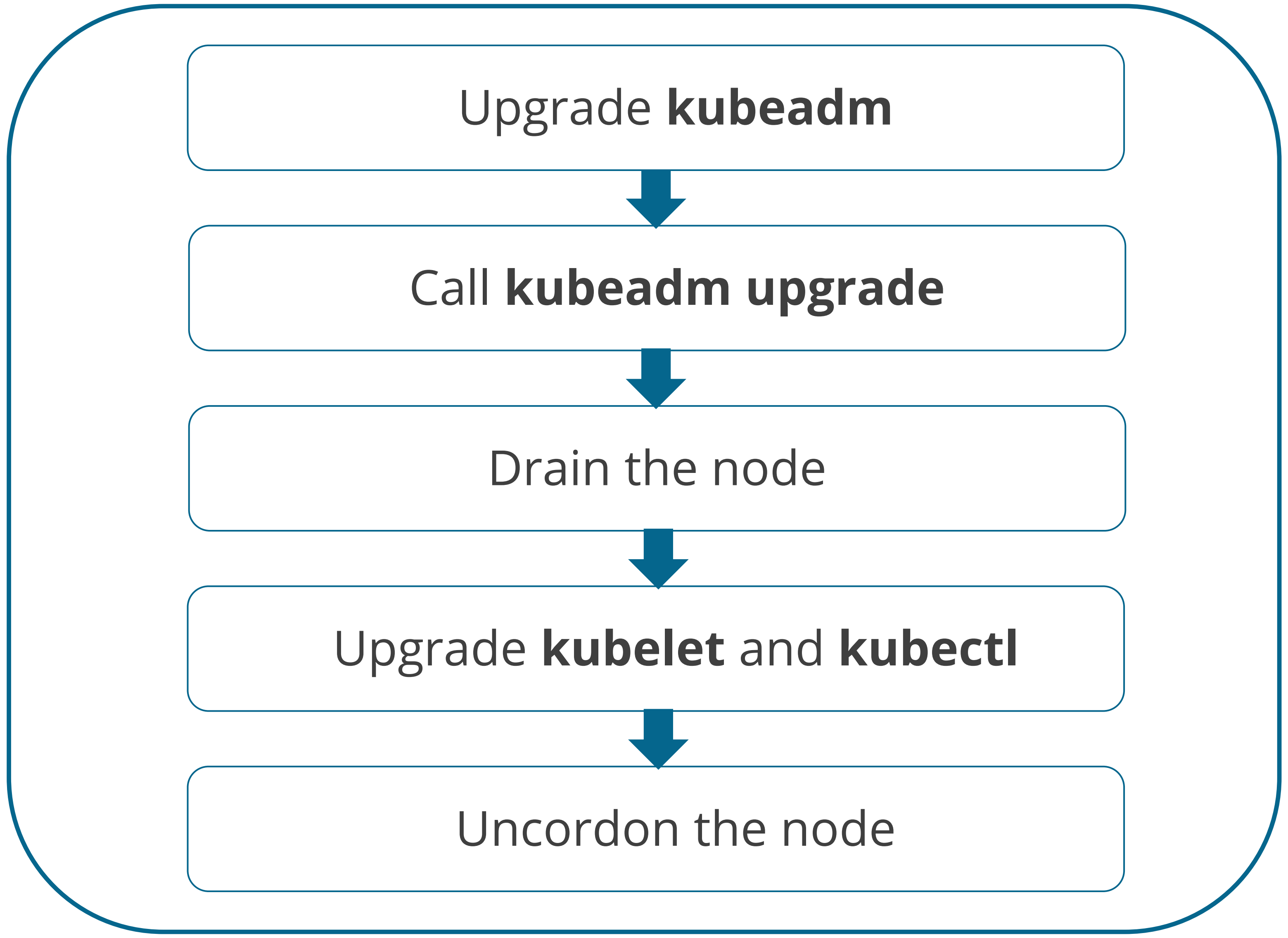
To uncordon the node, enter the command:

Example

```
# replace <node-to-drain> with the name of your node
kubectl uncordon <node-to-drain>
```

Upgrade Worker Nodes

The upgrade procedure on worker nodes involves the following steps:



Upgrade Worker Nodes

To upgrade kubeadm the below mentioned commands can be used:

Example

```
# replace x in 1.23.x-00 with the latest patch version
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.23.x-00 && \
apt-mark hold kubeadm
-
# you can also use the following method
apt-get update && \
apt-get install -y --allow-change-held-packages kubeadm=1.23.x-00
```

Upgrade Worker Nodes

Worker nodes can be upgraded and drained using the following commands:

Example

```
#For worker nodes this upgrades the local kubelet configuration:
sudo kubeadm upgrade node


#Drain the node
# replace <node-to-drain> with the name of your node you are draining
kubectl drain <node-to-drain> --ignore-daemonsets
```

Upgrade Worker Nodes

The upgrade procedure for kubelet and kubectl can be done as follows:

Example

```
# replace x in 1.23.x-00 with the latest patch version
apt-mark unhold kubelet kubectl && \
apt-get update && apt-get install -y kubelet=1.23.x-00 kubectl=1.23.x-00 &&
\
apt-mark hold kubelet kubectl
-
# you can also use the following method
apt-get update && \
apt-get install -y --allow-change-held-packages kubelet=1.23.x-00
kubectl=1.23.x-00
```

Upgrade Worker Nodes

Here is how to restart the kubelet and uncordon the node:

Example

```
#Restart the kubelet:
sudo systemctl daemon-reload
sudo systemctl restart kubelet

#Uncordon the node
#Bring the node back online by marking it schedulable:

# replace <node-to-drain> with the name of your node
kubectl uncordon <node-to-drain>
```

Verify the Status of the Cluster

After upgrading the kubelet on all the nodes, the status of the cluster must be checked. This is done to verify the availability of all the nodes.

It can be done by running the following command :

Example

```
kubectl get nodes
```

Note

The Status column should display **ready** for all the nodes. The version number should also be updated.

Recovering from a Failure State

If an operation fails, for example, if **kubeadm upgrade** fails, kubeadm can be recovered from a bad state.

Rerun **kubeadm upgrade**

Run **kubeadm upgrade apply --force** without changing the version that the cluster is running

Upgrading Control Plane's Kubernetes Versions

Duration: 10 mins

Problem Statement:

You have been asked to upgrade the kubeadm, kubectl, and kubelet versions of the control plane.

Assisted Practice: Guidelines

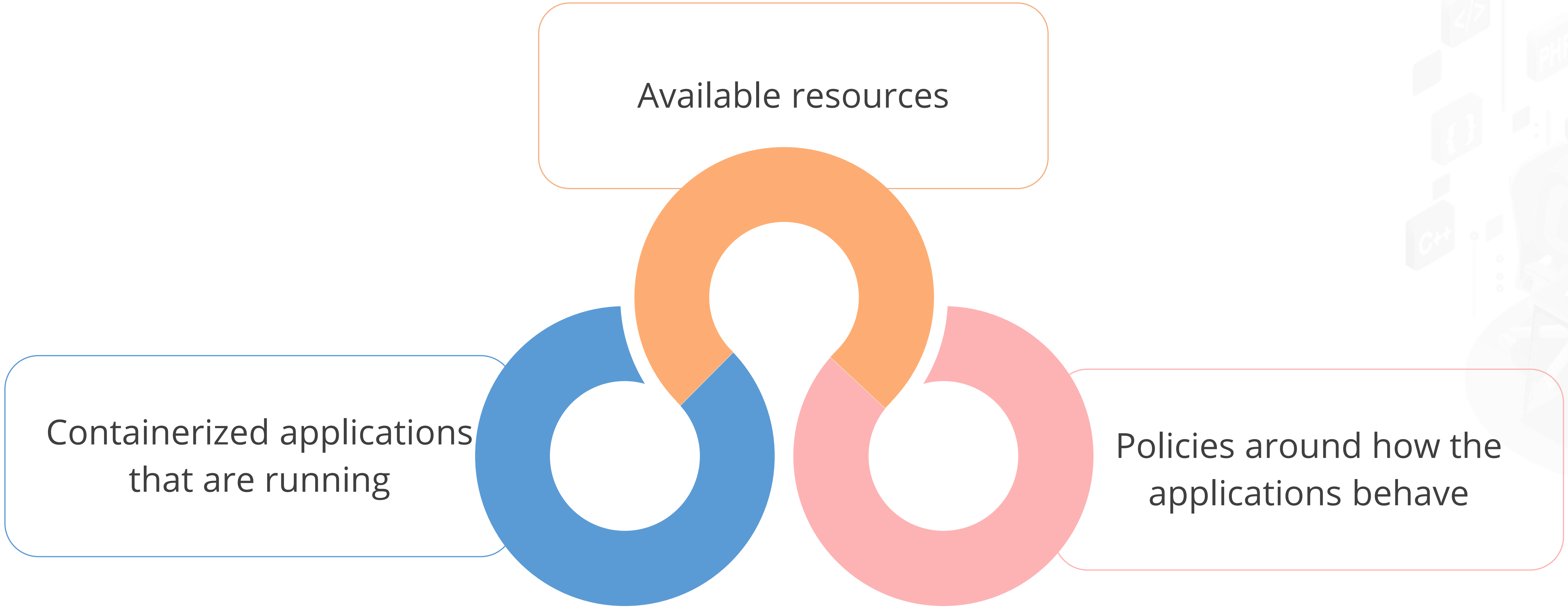
Steps to be followed:

1. Determining which version to upgrade
2. Finding the latest patch release of Kubernetes
3. Verifying the current version of Kubernetes
4. Upgrading the repositories
5. Holding the Kubernetes versions
6. Upgrading the control plane
7. Verifying the updated version of Kubernetes

Managing Kubernetes Objects

Kubernetes Objects

Kubernetes objects are persistent entities that represents the state of the cluster. They describe:


Available resources
Containerized applications that are running
Policies around how the applications behave

Object Spec and Status

A Kubernetes object usually includes two nested object fields, namely, object **spec** and object **status**. They govern the object's configuration.

spec

Describes the **desired state** of the object

status

Describes the **current state** of the object

Managing Kubernetes Objects

kubectl provides different ways to create and manage Kubernetes objects. Techniques that may be employed for managing Kubernetes objects are mentioned below:

Management technique	Operates on	Recommended environment	Supported writers	Learning curve
Imperative commands	Live objects	Development projects	1+	Lowest
Imperative object configuration	Individual files	Production projects	1	Moderate
Declarative object configuration	Directories of files	Production projects	1+	Highest

Managing the Kubernetes Objects

Duration: 5 mins

Problem Statement:

You have been assigned a task to understand the management of Kubernetes objects.

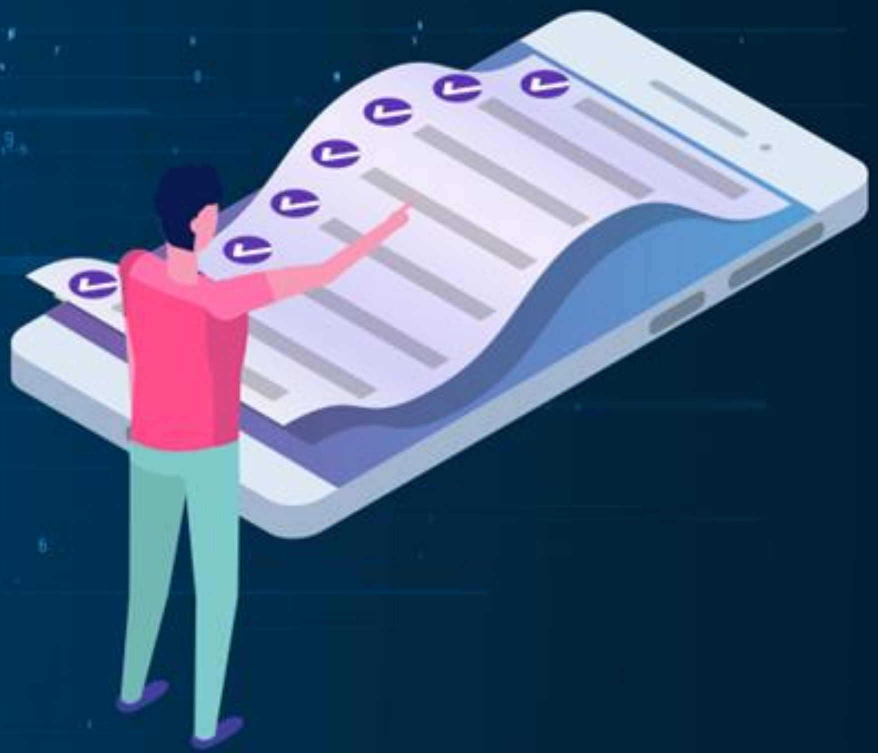
Assisted Practice: Guidelines

Steps to be followed:

1. Creating a Deployment
2. Verifying the Pod object
3. Verifying the Deployment object
4. Exposing the Deployment
5. Viewing the Service object
6. Fetching and storing the Deployment admin in a text file

Key Takeaways

- Kubernetes clusters allow containers to run across multiple machines and environments : virtual, physical, cloud-based, and on-premises.
- Cloud infrastructure technologies let you run Kubernetes on public, private and hybrid clouds .
- The kubectl command-line tool supports several different ways to create and manage Kubernetes objects.
- Connections from the API server to a Node, Pod, or service default to plain HTTP connections are neither authenticated nor encrypted.



Lesson-End Project

Duration: 30 Min

Managing Multiple Namespaces with Respective Roles

Project agenda: To manage multiple namespaces with respective roles.

Description:

Your project requires two namespaces, Simplilearn and CKA. The namespace Simplilearn must have user1 with editing access and user2 with only viewing access. The namespace CKA must have user1 with only viewing access and user4 with admin access.

Steps to perform:

1. Creating a Kubernetes cluster
2. Connecting to a cluster and then creating a Simplilearn namespace with user1 and user2
3. Connecting to a cluster and then creating a CKA namespace with user1 and user4