DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

Docker Kubernetes Service

Learning Objectives

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

- Implement Kubernetes orchestration for Docker Enterprise
- Route traffic to Kubernetes pods
- Deploy containerized workloads as Kubernetes pods
- Configure Role-Based Access (RBAC) with MKE

Kubernetes: Overview

Introduction to Kubernetes

Kubernetes is an open-source orchestration tool for automating the placement, management, deployment, and routing of containers and scales the requirements, availability, failover, and deployment patterns.

Introduction to Kubernetes

Capabilities of Kubernetes:

- Container-level resource management
- Container health management
- Secrets and configuration management
- Service discovery and load balancing
- Automated deployment and rollback
- Service and process definition
- Storage orchestration

Introduction to Kubernetes

Advantages of Kubernetes:

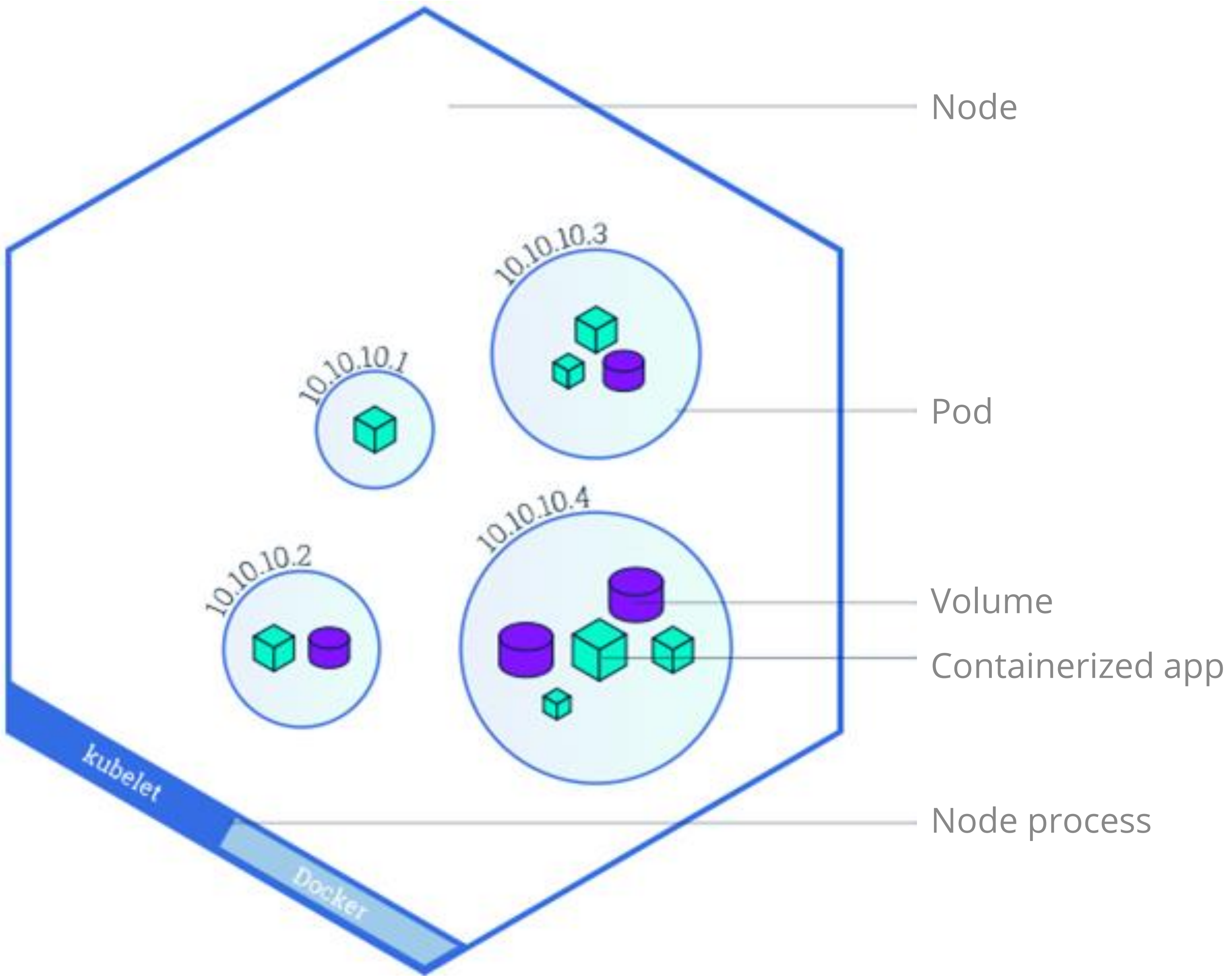
- High availability and fault tolerance
- Auto-scaling based on traffic and server load
- Extensive ecosystem around Container Networking Interface and Container Storage Interface
- Inbuilt logging and monitoring tools

Kubernetes Cluster

Overview of a Kubernetes Cluster:

- A Kubernetes cluster consists of a set of worker machines called **Nodes.**
- A node can either be a physical or a virtual machine and is managed by the **Master**.
- A node can have multiple **Pods** and the kubernetes master automatically schedules the pods across the nodes in a cluster.
- A pod is a group of application containers and includes shared storage, IP address, and information about running these containers.
- The **Control Plane** manages the worker nodes and the pods in a cluster.

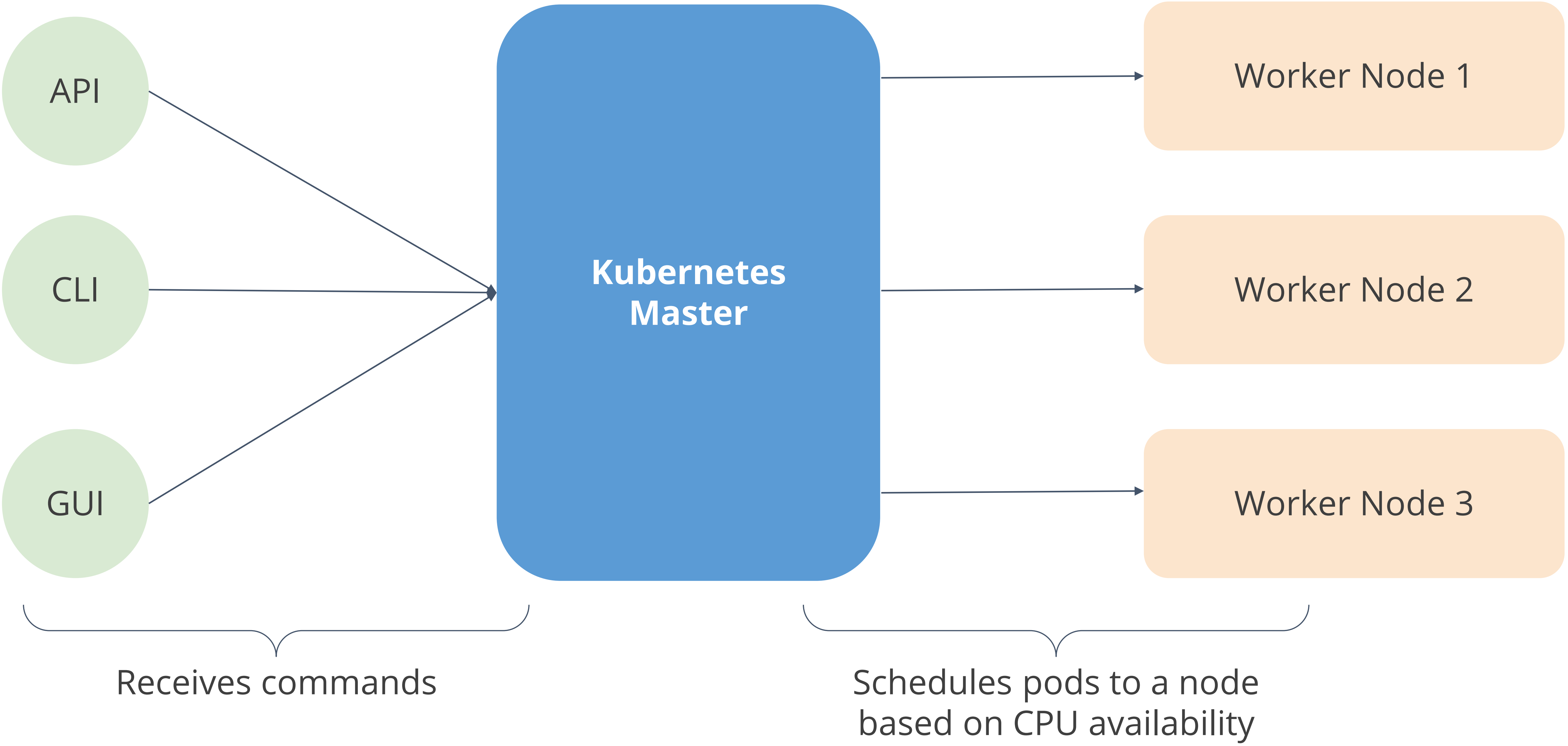
Kubernetes Cluster



Overview of a Kubernetes Node

Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

Kubernetes Architecture



Receives commands

Schedules pods to a node based on CPU availability

Kubernetes CLI

Kubernetes CLI

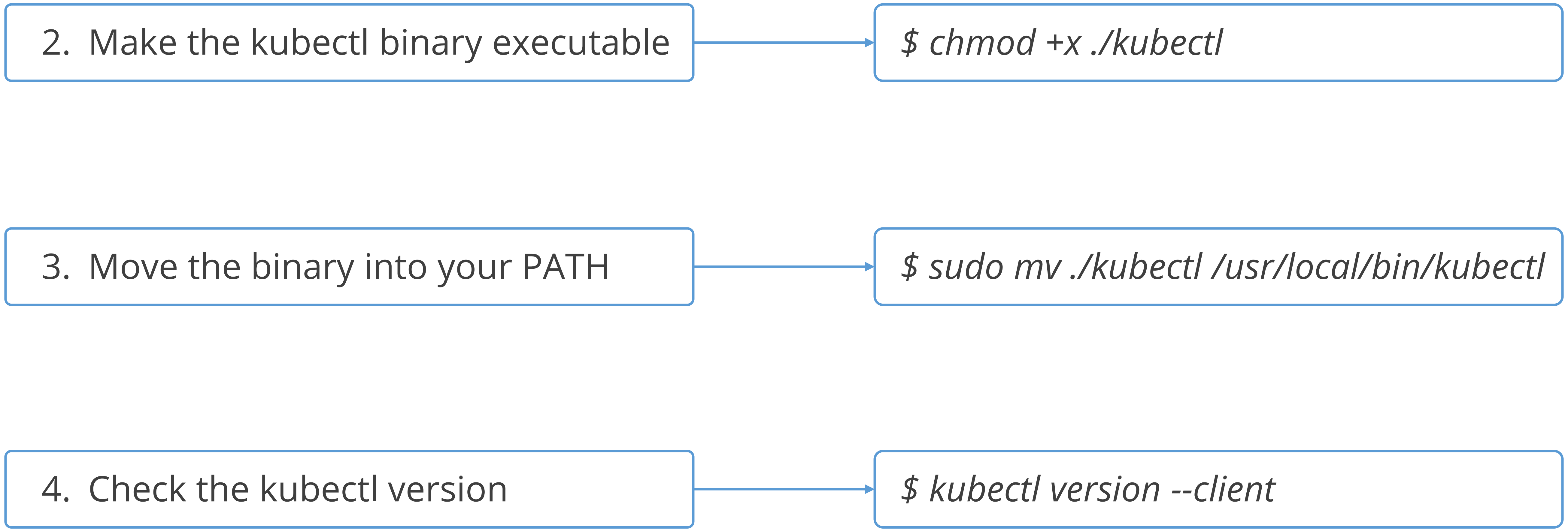
Kubernetes CLI tool, called **kubectl,** interacts with the API server and is also used to run commands for Kubernetes clusters.

Steps to install kubectl binary using curl on Linux:

1. Run the **curl** command to install kubectl binary

```
$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
```

Kubernetes CLI



2. Make the kubectl binary executable

```
$ chmod +x ./kubectl
```

3. Move the binary into your PATH

```
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

4. Check the kubectl version

```
$ kubectl version --client
```

kubectl Commands for Docker

kubectl commands equivalent to docker run command:

```
# run an nginx deployment and expose the deployment
docker run -d --restart=always -e DOMAIN=cluster --name nginx-app -p 80:80 nginx
```

```
# start the pod running nginx
kubectl create deployment --image=nginx nginx-app

# add env to nginx-app
kubectl set env deployment/nginx-app  DOMAIN=cluster

# expose a port through with a service
kubectl expose deployment nginx-app --port=80 --name=nginx-http
```

kubectl Commands for Docker

kubectl command equivalent to docker ps command:

```
# list all the running containers
docker ps -a
```

```
# list all the runnings pods
kubectl get pods
```

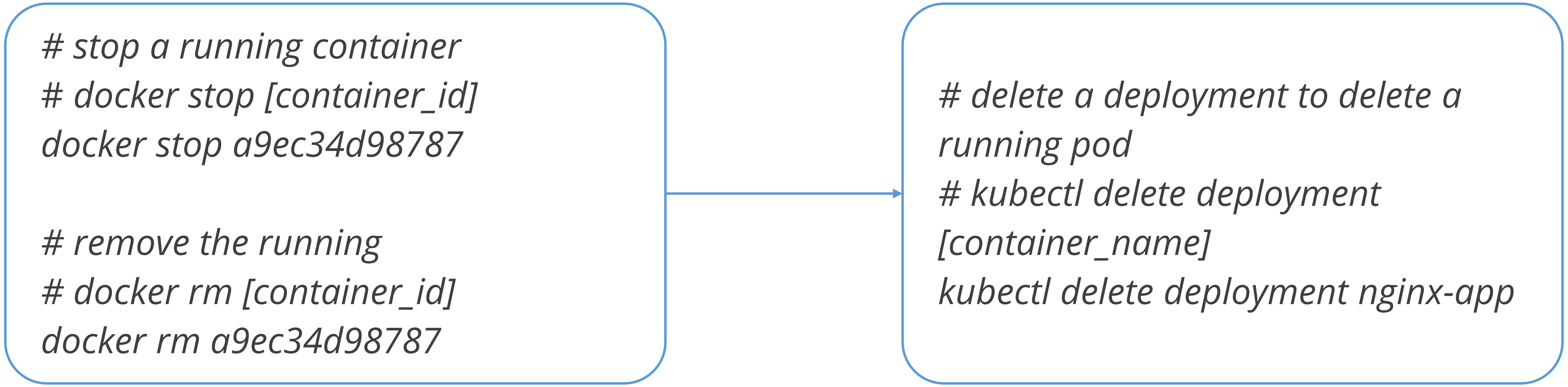
kubectl command equivalent to docker exec command:

```
# execute a command in a container
# docker exec [container_id]
[command_arg]
docker exec 55c103fa1296 cat
/etc/hostname
```

```
# execute a command in a container
# kubectl exec [container_name]
[command_arg]
kubectl exec nginx-app-5jyvm -- cat
/etc/hostname
```

kubectl Commands for Docker

kubectl command equivalent to docker stop and docker rm command:



```
# stop a running container
# docker stop [container_id]
docker stop a9ec34d98787

# remove the running
# docker rm [container_id]
docker rm a9ec34d98787
```

```
# delete a deployment to delete a
running pod
# kubectl delete deployment
[container_name]
kubectl delete deployment nginx-app
```

Assisted Practice

Access Kubernetes Cluster from Docker CLI

Problem Statement: You have been asked to configure and access the Kubernetes cluster from Docker CLI.

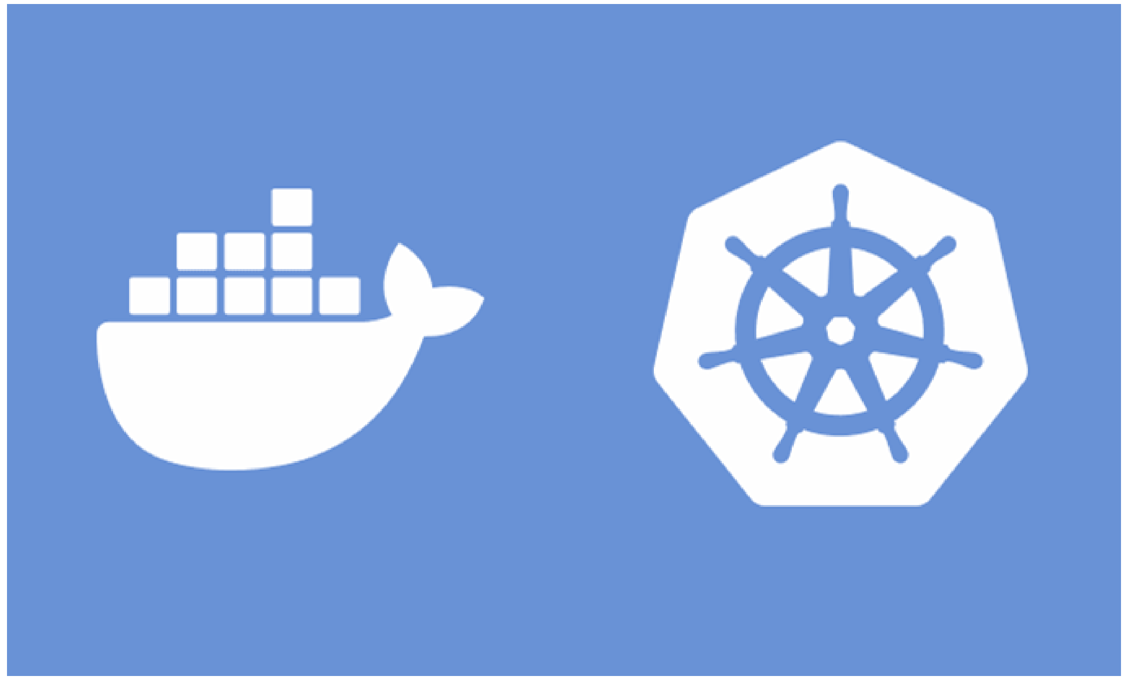
Steps to Perform:

1. Download the client bundle from MKE
2. Extract the environment script for MKE client bundle and execute it
3. Check the nodes on the Kubernetes cluster using kubectl command
4. Create a deployment and list the pod for this deployment
5. Check the newly created pod in the Pods tab of Kubernetes section on the MKE UI

Kubernetes Orchestration for MKE

Docker Kubernetes Service

Docker Kubernetes Service (DKS) is a unified operational model that simplifies the use of Kubernetes for developers and operators and makes it easy for enterprises to secure and manage their Kubernetes environment.



Docker Kubernetes Service

Docker Kubernetes Services

Features of Docker Kubernetes Services:

- Kubernetes orchestration
- CNCF certified Kubernetes conformance
- Kubernetes app deployment
- Compose stack deployment
- Role-based access control
- Blue-green deployments

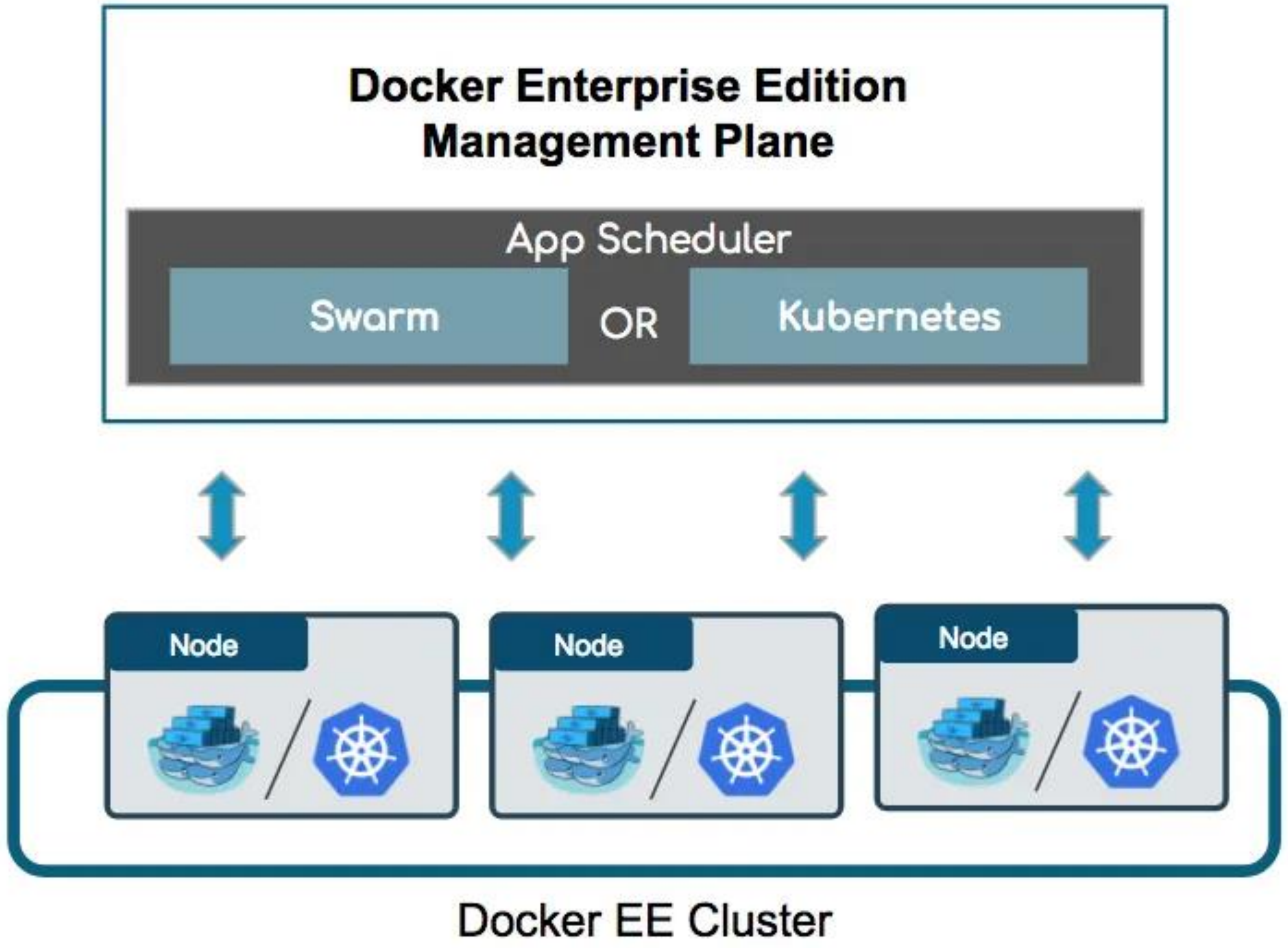
Docker Kubernetes Services

Features of Docker Kubernetes Services:

- Ingress controllers
- Pod Security Policies
- Container Storage Interface (CSI) support
- iSCSI support for Kubernetes
- Non-disruptive Docker Enterprise platform upgrades
- Experimental features such as Kubernetes-native ingress (Istio)

Kubernetes Orchestration for MKE

- MKE provides Kubernetes orchestration through Docker Kubernetes Service.
- It allows organizations to run Kubernetes interchangeably with Swarm orchestration.
- DKS clusters can be attached to a fleet of clusters managed by Mirantis Container Cloud.



Source: https://docs.mirantis.com/docker-enterprise/v3.1/dockeree-products/single/index.html

Kubernetes Orchestration for MKE

Integration of MKE with Kubernetes enables:

- Authenticating user client bundle certificates when communicating directly with the Kubernetes API server
- Authorizing requests via the MKE role-based access control model
- Assigning nodes to a namespace by injecting a NodeSelector automatically to workloads via admission control

Kubernetes Orchestration for MKE

Integration of MKE with Kubernetes enables:

- Keeping all the nodes in both Kubernetes and Swarm orchestrator inventories
- Fine-grained access control and privilege escalation prevention without the PodSecurityPolicy admission controller
- Resolving images of deployed workloads automatically, and accepting or rejecting images based on MKE’s signing-policy feature

Security Features

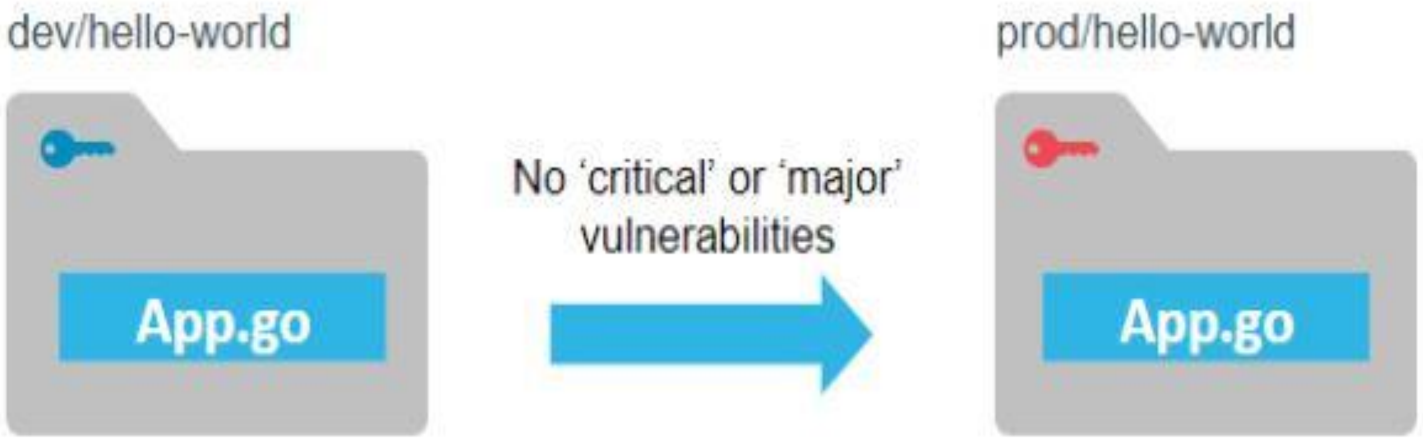
MKE security features for Kubernetes apps:

Image signing and scanning

Source: https://docs.docker.com/ee/#secure-supply-chain

Security Features

MKE security features for Kubernetes apps:

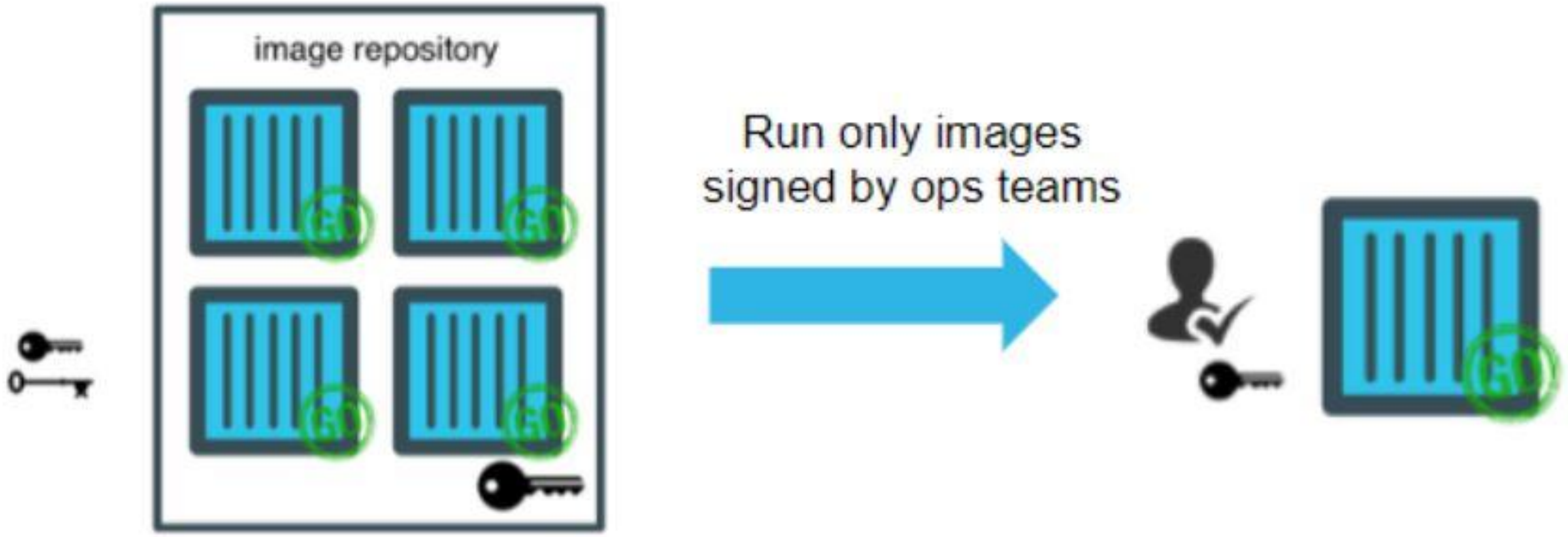


Policies for automating image promotions

Source: https://docs.docker.com/ee/#secure-supply-chain

Security Features

MKE security features for Kubernetes apps:

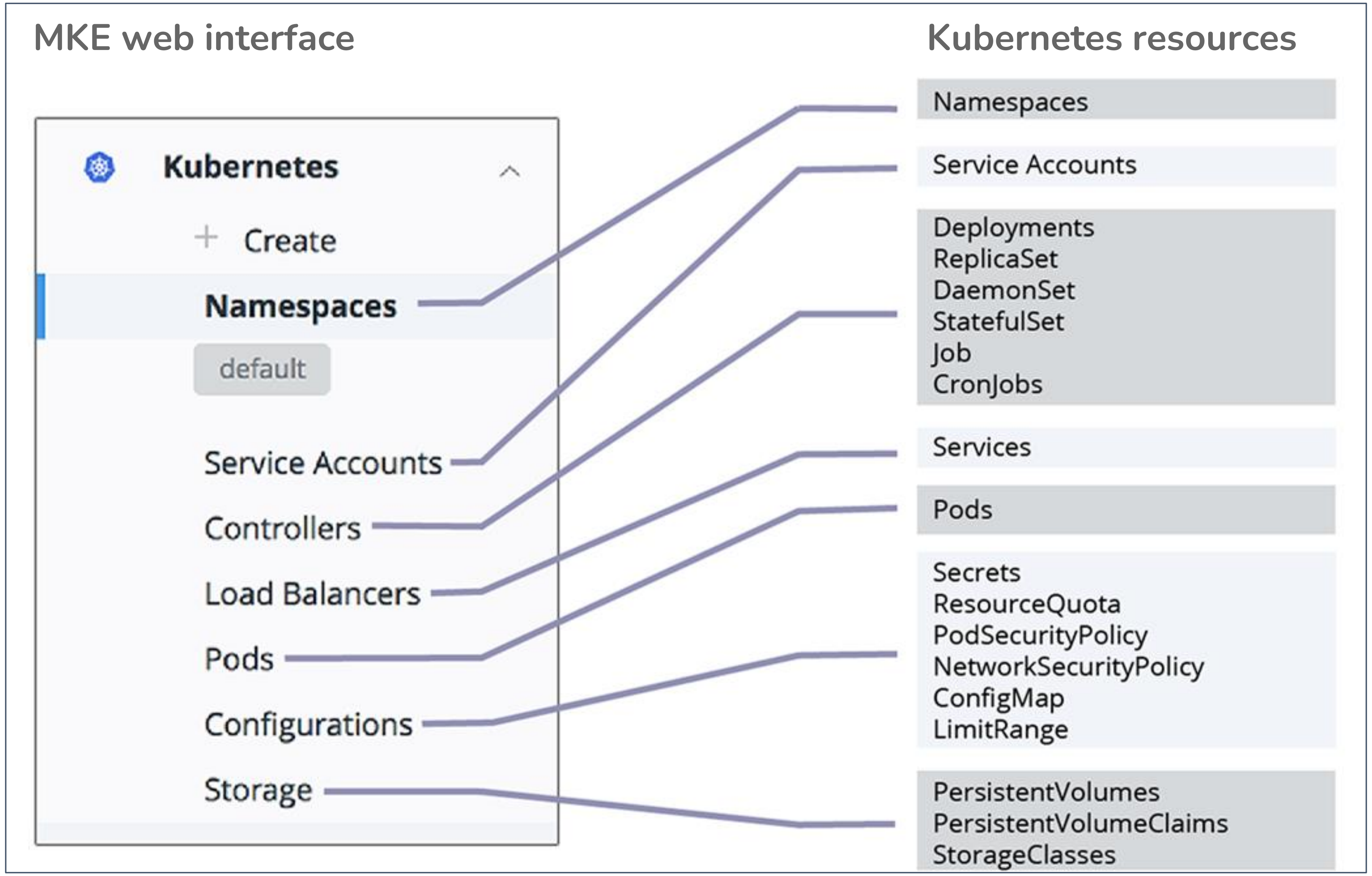


Content trust policy enforcement

Source: https://docs.docker.com/ee/#secure-supply-chain

Kubernetes Orchestration in MKE

Accessing Kubernetes resources from MKE web UI:



Kubernetes Orchestration in MKE

Set the default orchestrator type:

To set the orchestrator for new nodes:

- Log in to the MKE web UI with an administrator account
- Open the **Admin Settings** page, and in the left pane, click **Scheduler**
- Under **Set Orchestrator Type for New Nodes**, click **Swarm** or **Kubernetes**
- Click on the **Save** button

Routing Traffic to Kubernetes Pods

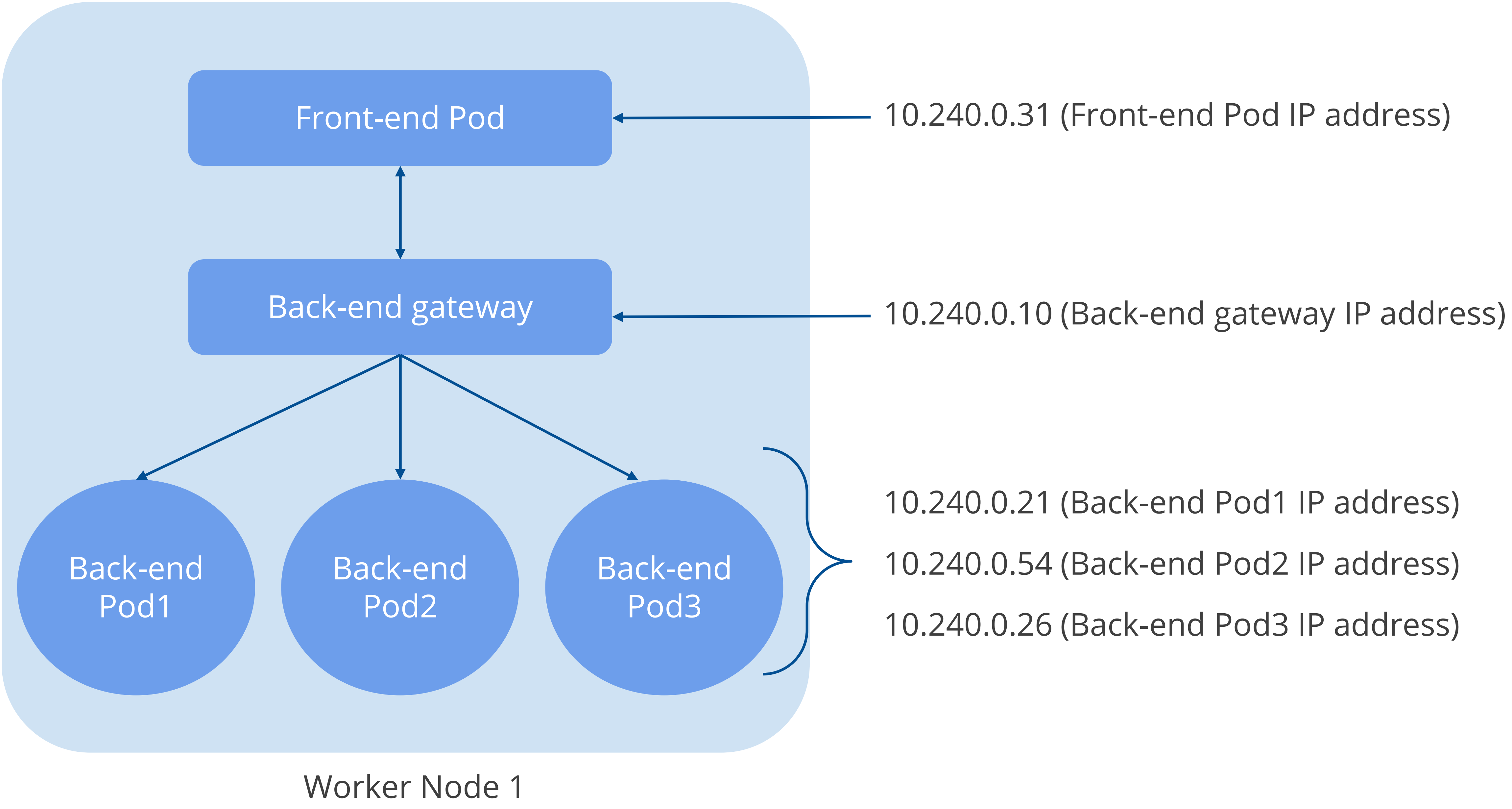
Kubernetes Service

Overview of Kubernetes Service:

- **Kubernetes Service** is an abstraction which acts as a gateway or an interface to communicate with a set of pods.
- A Service is a REST object like JSON and an instance can be created sending its definition to the API server through the POST request.
- The set of pods targeted by a Service is identified by a selector.
- A Service has its own IP address, called ClusterIP, which is used to communicate with the pods using proxies.
- It helps the users to perform operations like load-balancing and pods scaling.

Kubernetes Service

Working of a Kubernetes Service:


Front-end Pod
Back-end gateway
Back-end Pod1
Back-end Pod2
Back-end Pod3
10.240.0.31 (Front-end Pod IP address)
10.240.0.10 (Back-end gateway IP address)
10.240.0.21 (Back-end Pod1 IP address)
10.240.0.54 (Back-end Pod2 IP address)
10.240.0.26 (Back-end Pod3 IP address)
Worker Node 1

Assisted Practice

Create a Service to Route Traffic From Frontend to Backend Pods

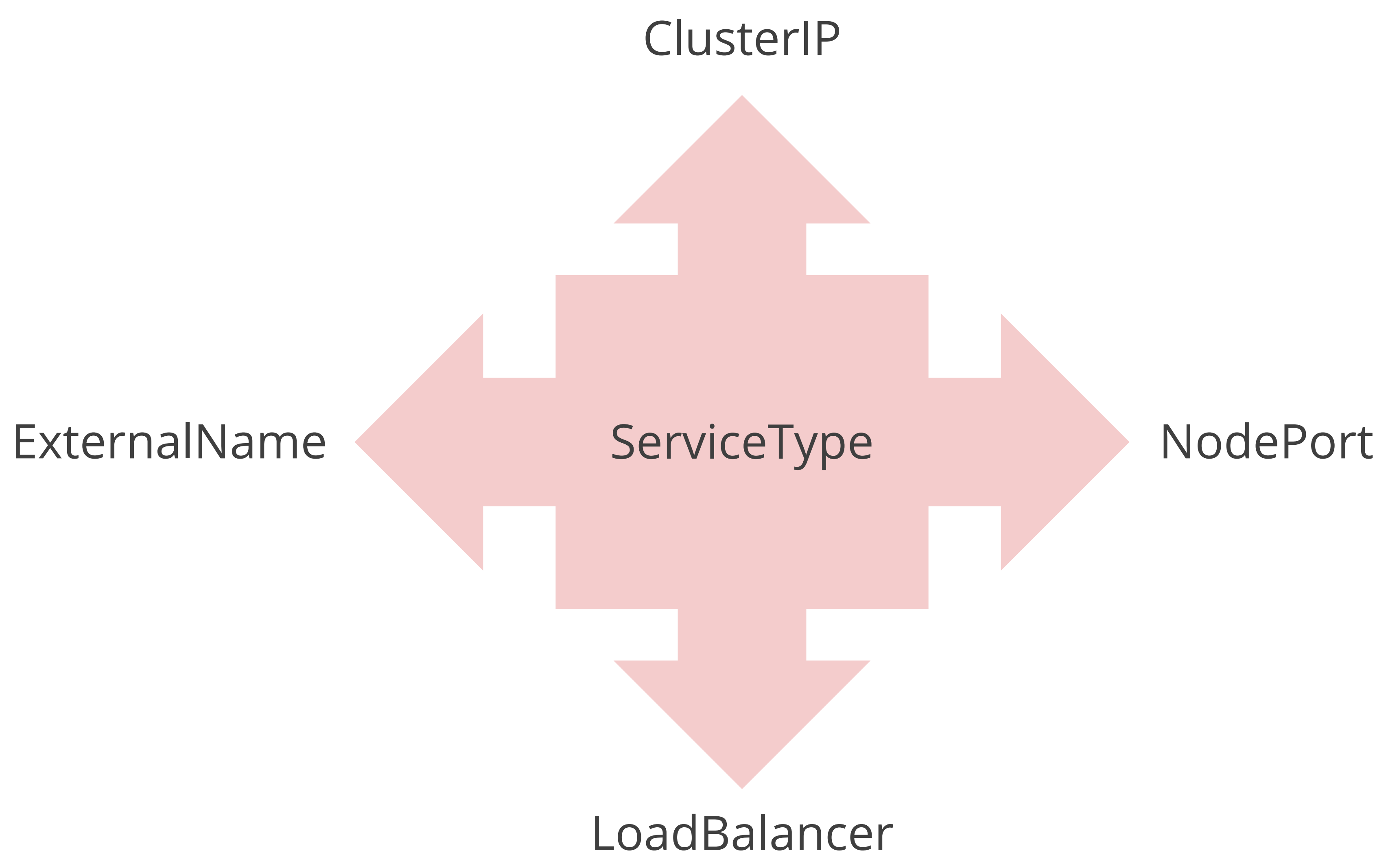
Problem Statement: Your technical manager has asked you to route the traffic of your frontend application. Create a Service to route traffic from frontend application to the backend pods.

Steps to Perform:

1. Create an nginx deployment and scale it to two pods
2. Create a service that can communicate with the backend pods
3. Create an endpoint to send traffic from the service to the backend pods
4. Create a frontend pod and connect it to the backend pods using the newly created service

Kubernetes ServiceTypes

Kubernetes ServiceTypes allow users to specify the type of Service they want, where the Type values are as follows:



Kubernetes ServiceTypes

Kubernetes ServiceTypes:

- **ClusterIP:** The Service is exposed on a cluster-internal IP address and can be reached only from within the cluster. This is the default ServiceType.
- **NodePort:** The Service is exposed on a static port of each worker node's IP address (the NodePort). A ClusterIP Service is automatically created for the NodePort Service routing.
- **LoadBalancer:** The Service is exposed externally through a load balancer. NodePort and ClusterIP Services are automatically created for the external LoadBalancer Service routing.
- **ExternalName:** The Service is mapped to the contents of the externalName field by returning a CNAME record with its value. No proxying of any kind is set up.

Kubernetes Networking Model

Kubernetes Networking Model:

Kubernetes imposes the following fundamental requirements on any networking implementation to restrict any intentional network segmentation policies:

- Pods on a Node can communicate with all the pods on other Nodes without NAT.
- Agents on a Node such as system daemons or kubelet can communicate with all the pods on that Node.
- Pods in the host network of a Node can communicate with all the pods on all other Nodes without NAT.

Kubernetes Networking Model

Four networking challenges to be addressed:

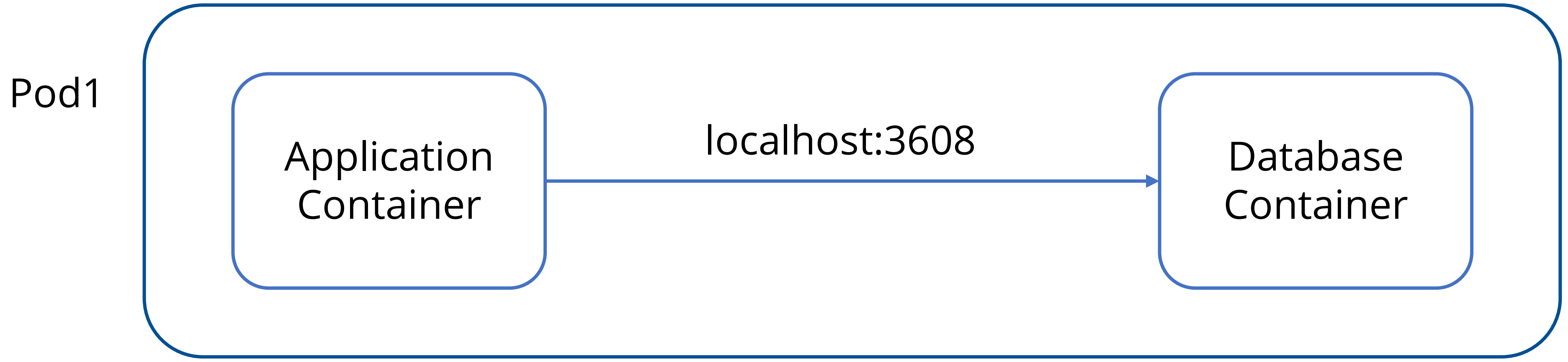
Kubernetes networking model solves the four distinct networking problems:

- Container-to-container communication
- Pod-to-pod communication
- Pod-to-Service communication
- External-to-Service communication

Kubernetes Networking Model

Container-to-container communication:

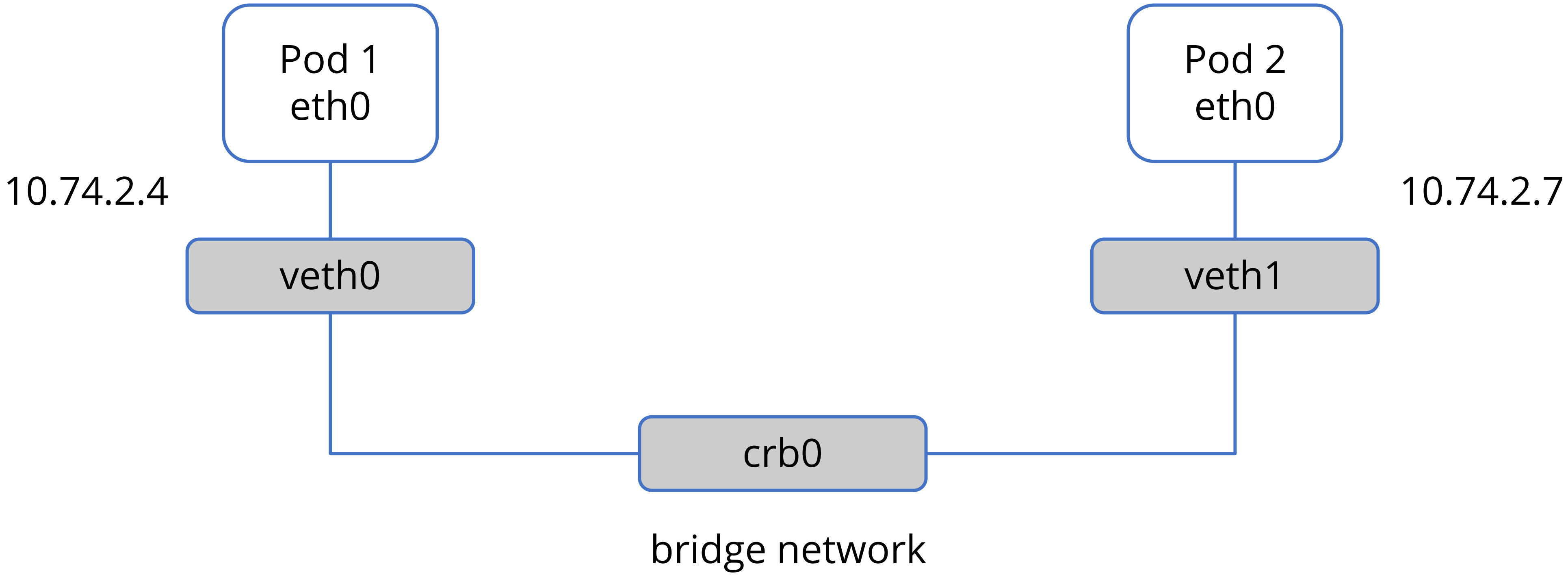
- Container-to-container communication primarily happens inside a pod, where all the containers have the same IP address.
- Communication between the containers inside a pod happens via localhost.



Kubernetes Networking Model

Pod-to-pod communication:

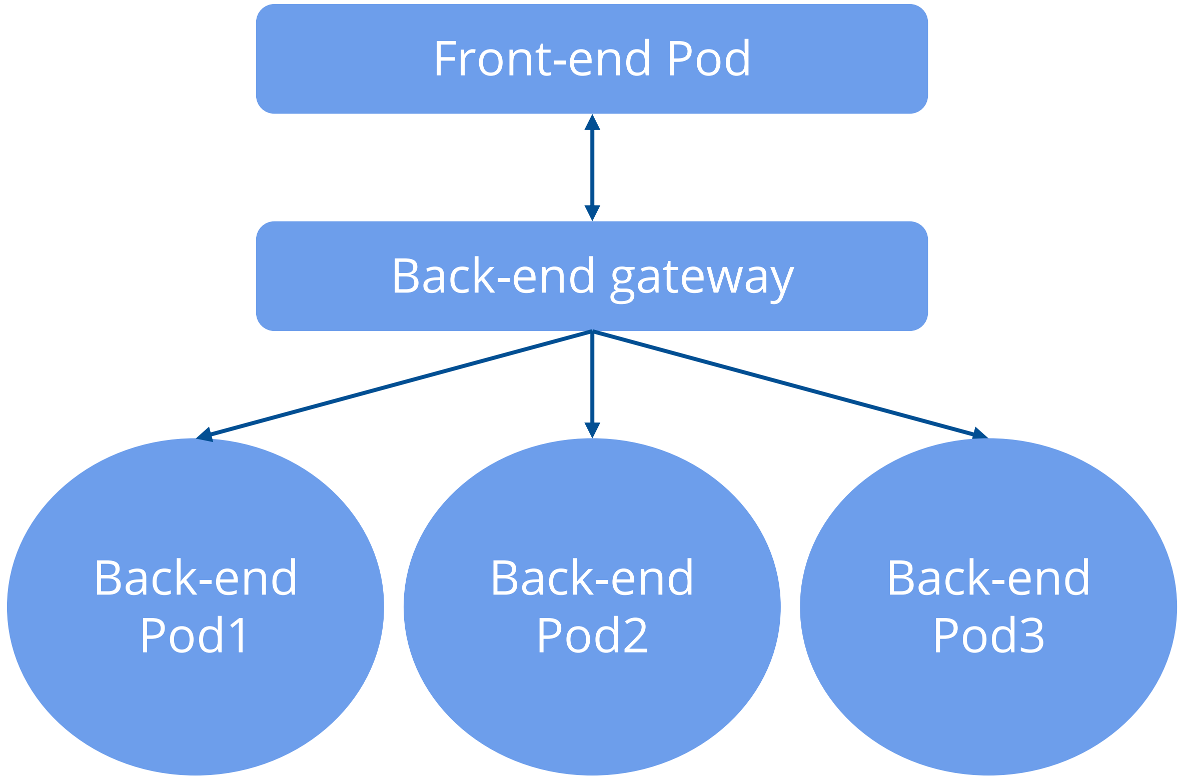
- Pods have their Ethernet namespace (**eth0**) and can be connected using a Linux Virtual Ethernet device or **veth pair** consisting of two virtual interfaces (veth0 and veth1).
- To connect the pods, assign one side of the **veth** pair to the root network namespace (**cbr0** or bridge network), and the other side to the pod's network namespace.



Kubernetes Networking Model

Pod-to-Service communication:

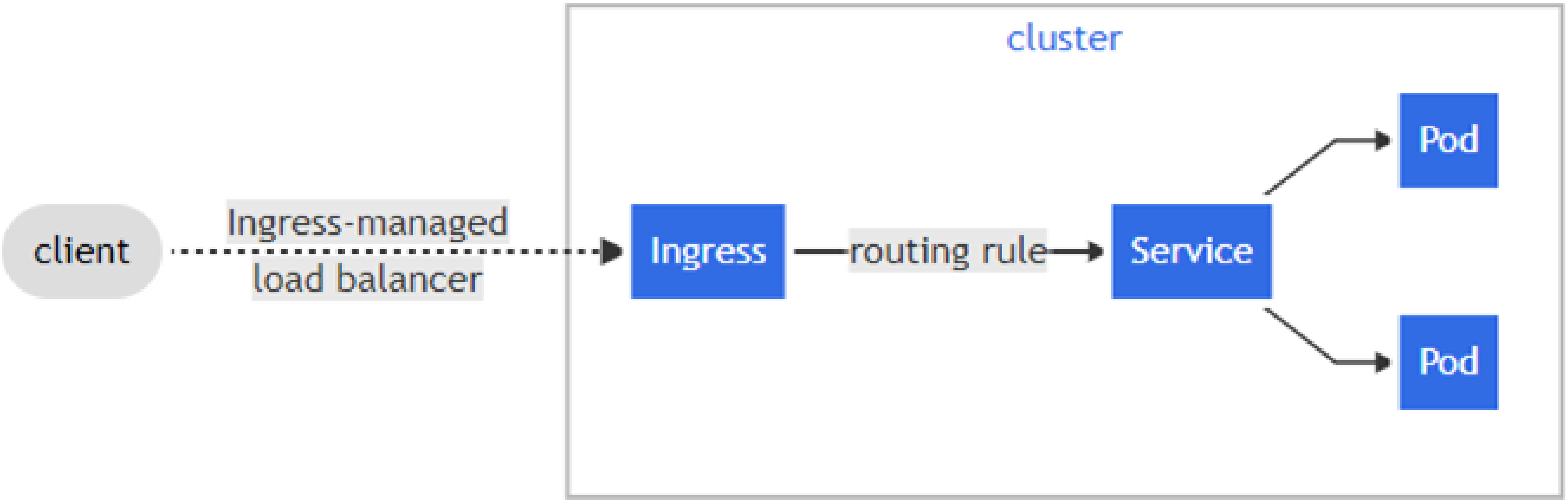
- Kubernetes Service can act as an interface to provide a single IP address and DNS through which pods can be accessed.
- A Service can route traffic to the pods by tracking the pod's IP address through endpoints.



Kubernetes Networking Model

External-to-Service communication:

- Kubernetes **Ingress** provides a collection of routing rules which regulate the external user’s access to the Services running within the Kubernetes cluster.



Source: https://kubernetes.io/docs/concepts/services-networking/ingress/

Persistent Storage to Kubernetes

Persistent Volumes

Overview of PersistentVolume:

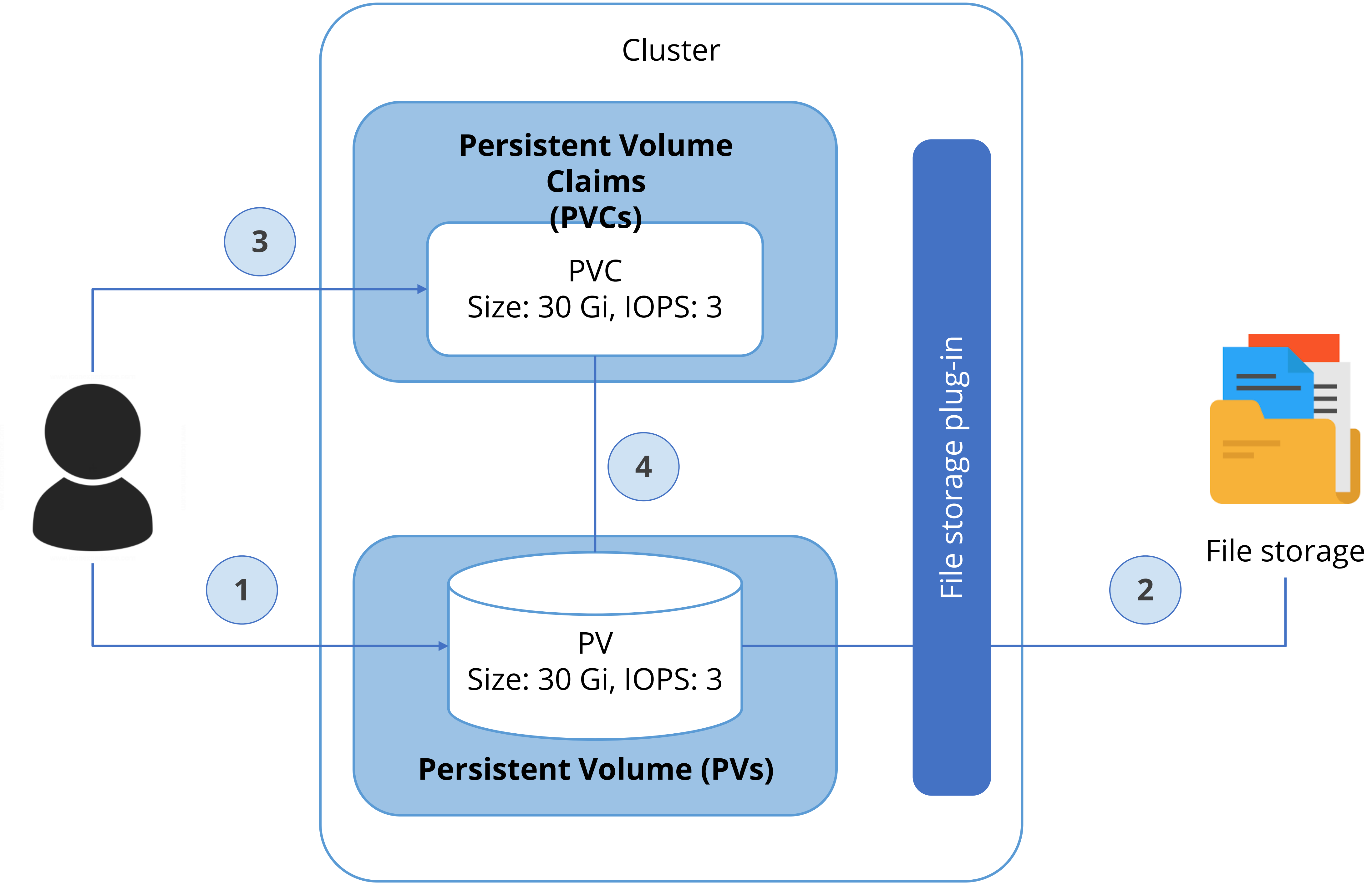
- **PersistentVolume** (PV) is a piece of storage in a cluster provisioned by an administrator or dynamically provisioned using Storage Classes.
- It is a volume plugin and has a life cycle independent of any pod that uses the PV.
- This API object captures the details of storage implementation.
- Every volume which is created can be of a different type.
- This can be managed by the storage administrator or ops team.

Persistent Volumes

Overview of PersistentVolumeClaim:

- A **PersistentVolumeClaim** (PVC) is a request for storage by a user.
- It is similar to a pod and consumes PV resources.
- It can request specific sizes and access modes (ReadWriteOnce, ReadOnlyMany, or ReadWriteMany).
- PVs are resources in the cluster. PVCs are requests for those resources and also act as claim checks to the resources.

Persistent Volumes



Working of persistent storage in Kubernetes

Assisted Practice

Configure a Pod to Use a PersistentVolume for Storage

Problem Statement: You have been asked by your tech lead to configure a Pod to use a PersistentVolume for storage.

Steps to Perform:

1. Attach a label to the master node in order to assign the pods to it
2. Create an html file on the master node
3. Create a PersistentVolume for the storage
4. Create a PersistentVolumeClaim for providing physical storage to a pod
5. Create a Pod that uses the newly created PVC as a volume

Container Storage Interface

Overview of Container Storage Interface (CSI):

- **Container Storage Interface** (CSI) defines a standard interface for container orchestration to expose arbitrary storage systems to their container workloads.
- CSI volume type can be used to attach or mount the volumes exposed by the CSI driver, provided a CSI compatible volume driver is deployed on the cluster.
- It can be used in a pod in three different ways:
  - through a reference to a PersistentVolumeClaim
  - with a generic ephemeral volume
  - with a CSI ephemeral volume

Storage Classes

Overview of Storage Classes:

- A **StorageClass** offers a way for the administrators to describe the **classes** of storage that they offer.
- Different classes might map to quality-of-service levels, backup policies, or arbitrary policies.
- Each StorageClass contains the fields **provisioner, parameters,** and **reclaimPolicy**.
- Administrators can specify a default StorageClass for PVCs that does not request any particular class binding.

Deployments and Configurations

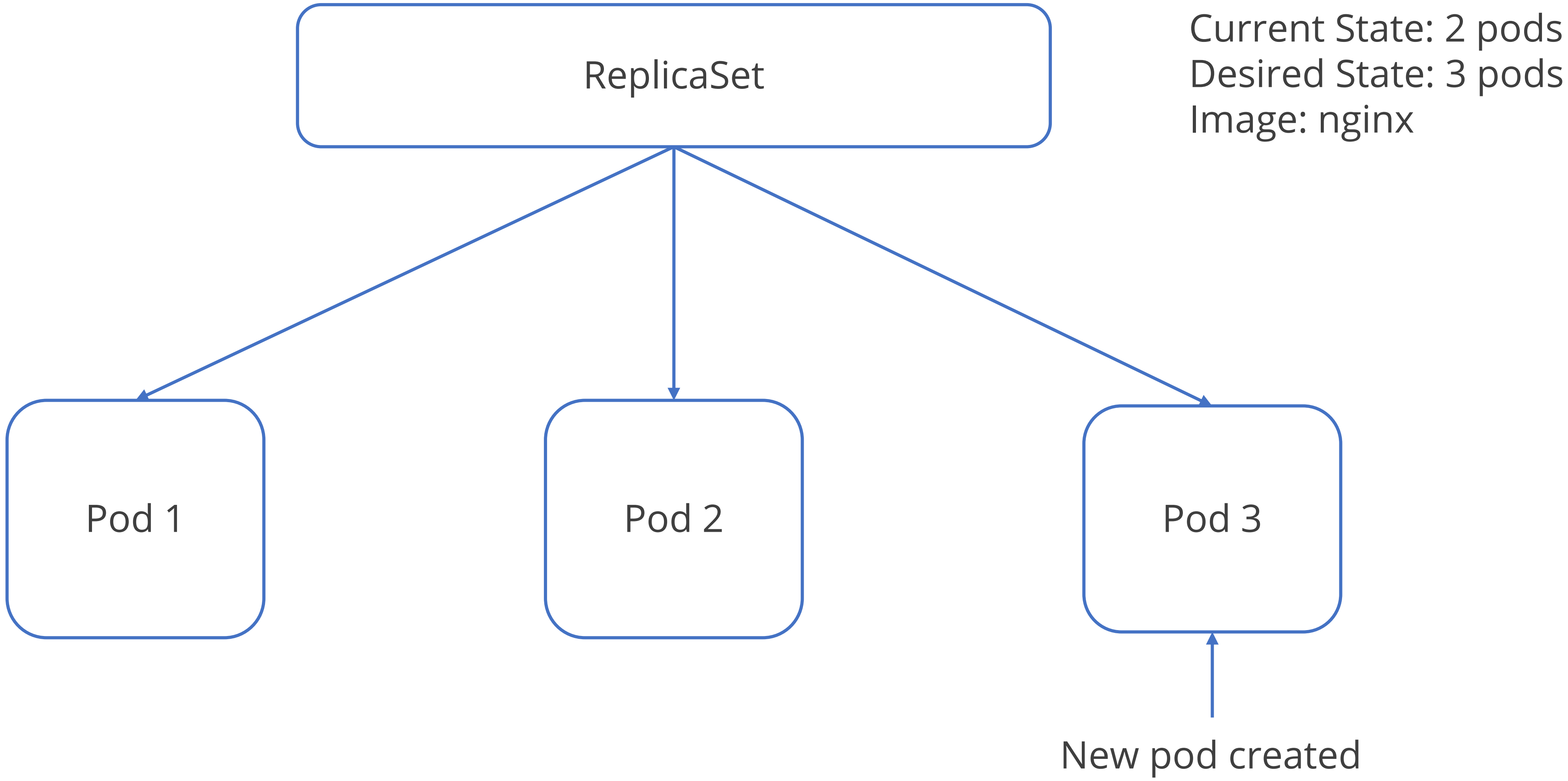
ReplicaSets

Overview of ReplicaSets:

- A **ReplicaSet** maintains a stable set of replica pods running at any given time in a node.
- It is defined with fields such as a selector, number of replicas, and pod template.
- Its main purpose is to create and delete pods as needed to reach the desired state.
- When a ReplicaSet needs to create new pods, it uses its pod template.

ReplicaSets

Working of ReplicaSets:



Current State: 2 pods
Desired State: 3 pods
Image: nginx

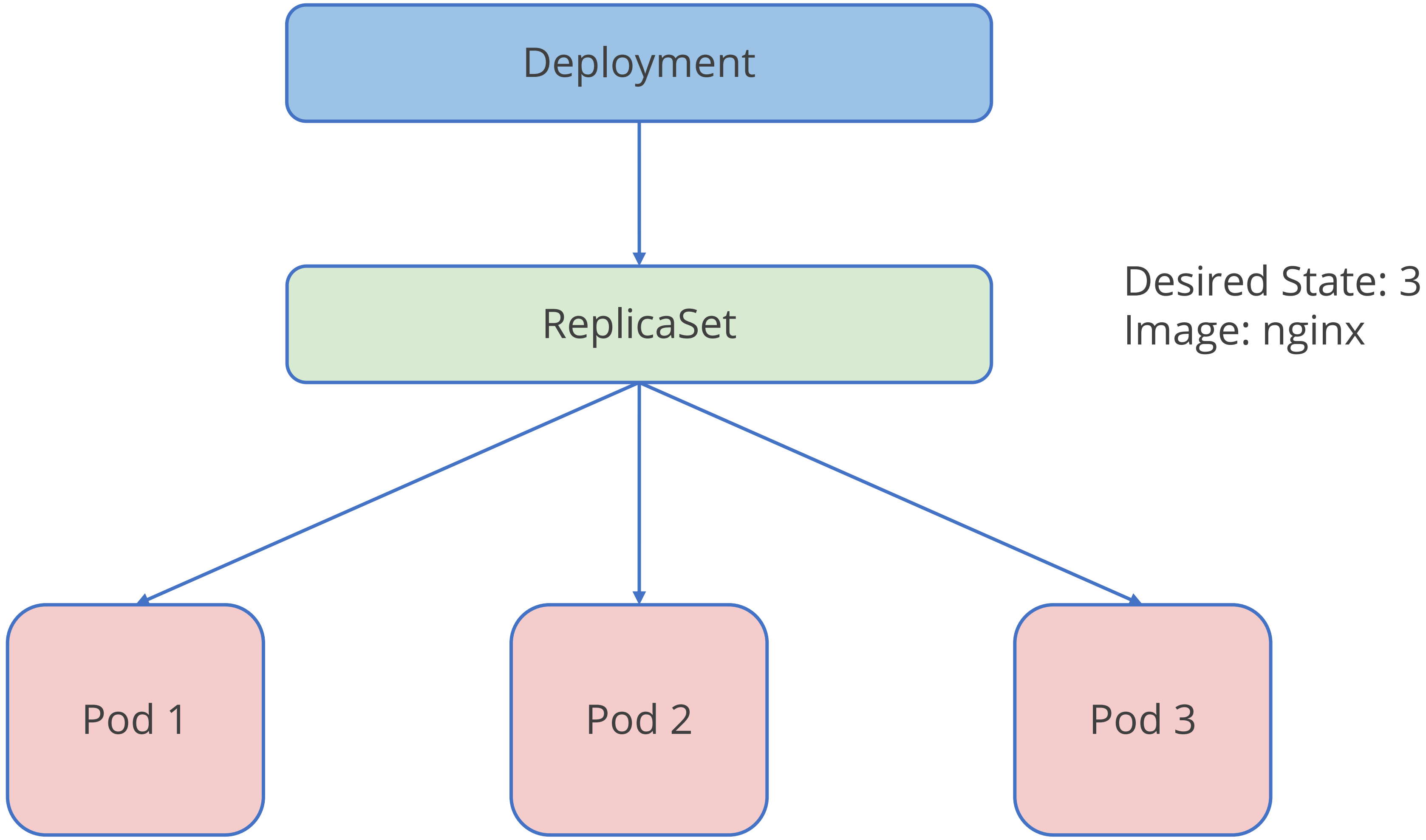
Workload Deployments

Overview of Deployments:

- **Deployments** provide replication functionality with the help of ReplicaSets, along with other capabilities like rolling out changes, rollback changes if required.
- It provides declarative updates for pods ReplicaSets.
- The Deployment Controller changes the actual state to the desired state at a controlled rate, based on the desired state described in a Deployment.
- Deployments can be defined to create new ReplicaSets or to remove existing Deployments and adopt all their resources with new Deployments.

Workload Deployments

Working of Deployments:



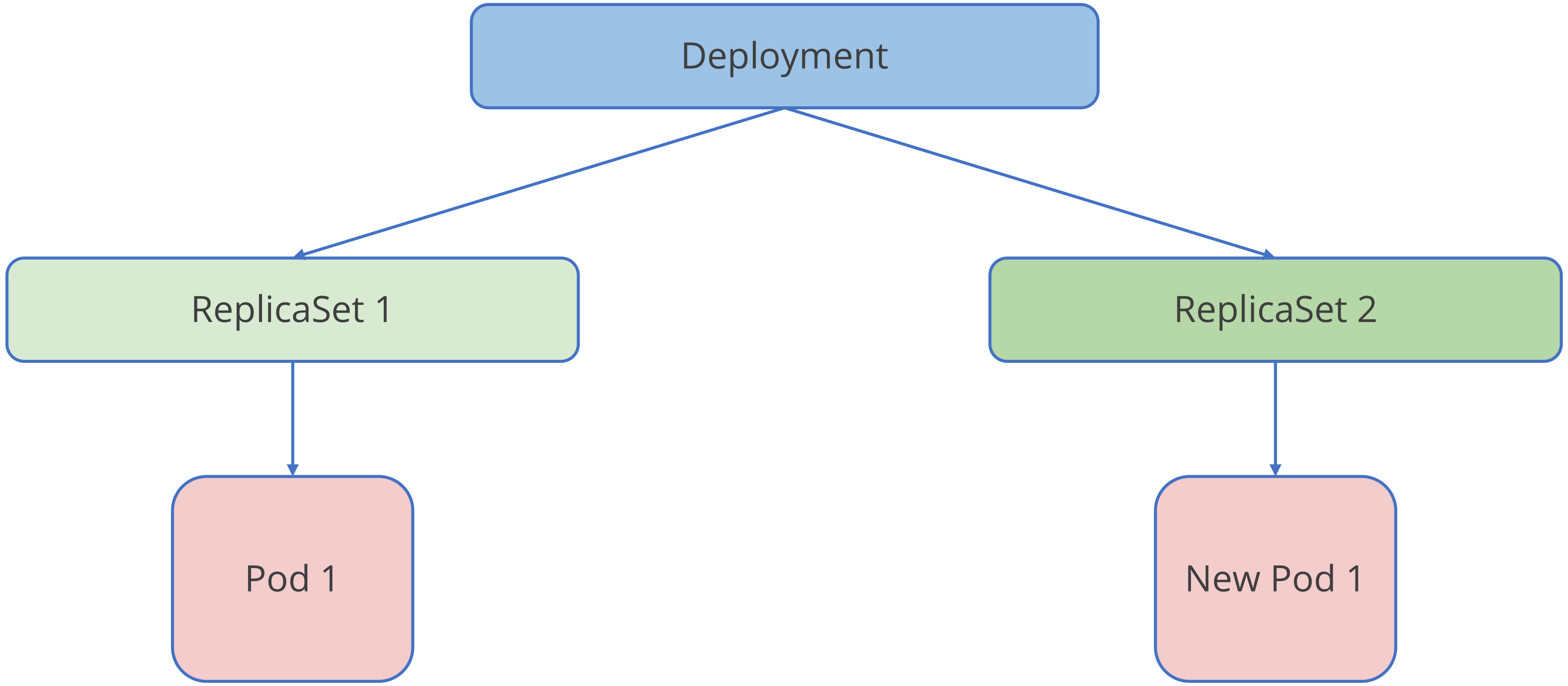
Workload Deployments

Use cases of Deployments:

- Creating a Deployment to rollout a ReplicaSet
- Declaring the new state of the pods
- Rolling back to a previous Deployment revision
- Scaling up the Deployment to facilitate more load
- Pausing the Deployment to apply multiple fixes to its PodTemplateSpec
- Using the Deployment status as an indicator
- Cleaning up the older ReplicaSets

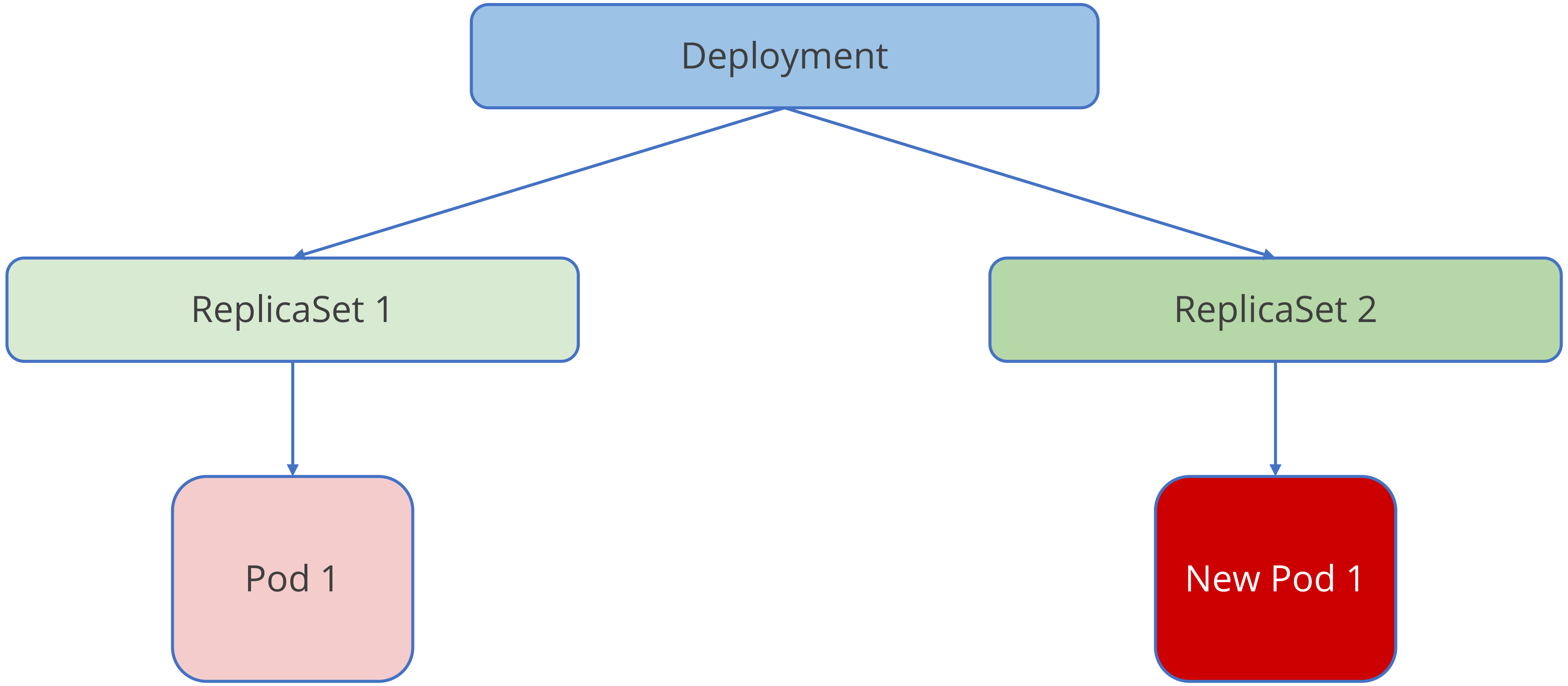
Workload Deployments

Rolling out changes using Deployments:



Workload Deployments

Rolling back changes using Deployments:



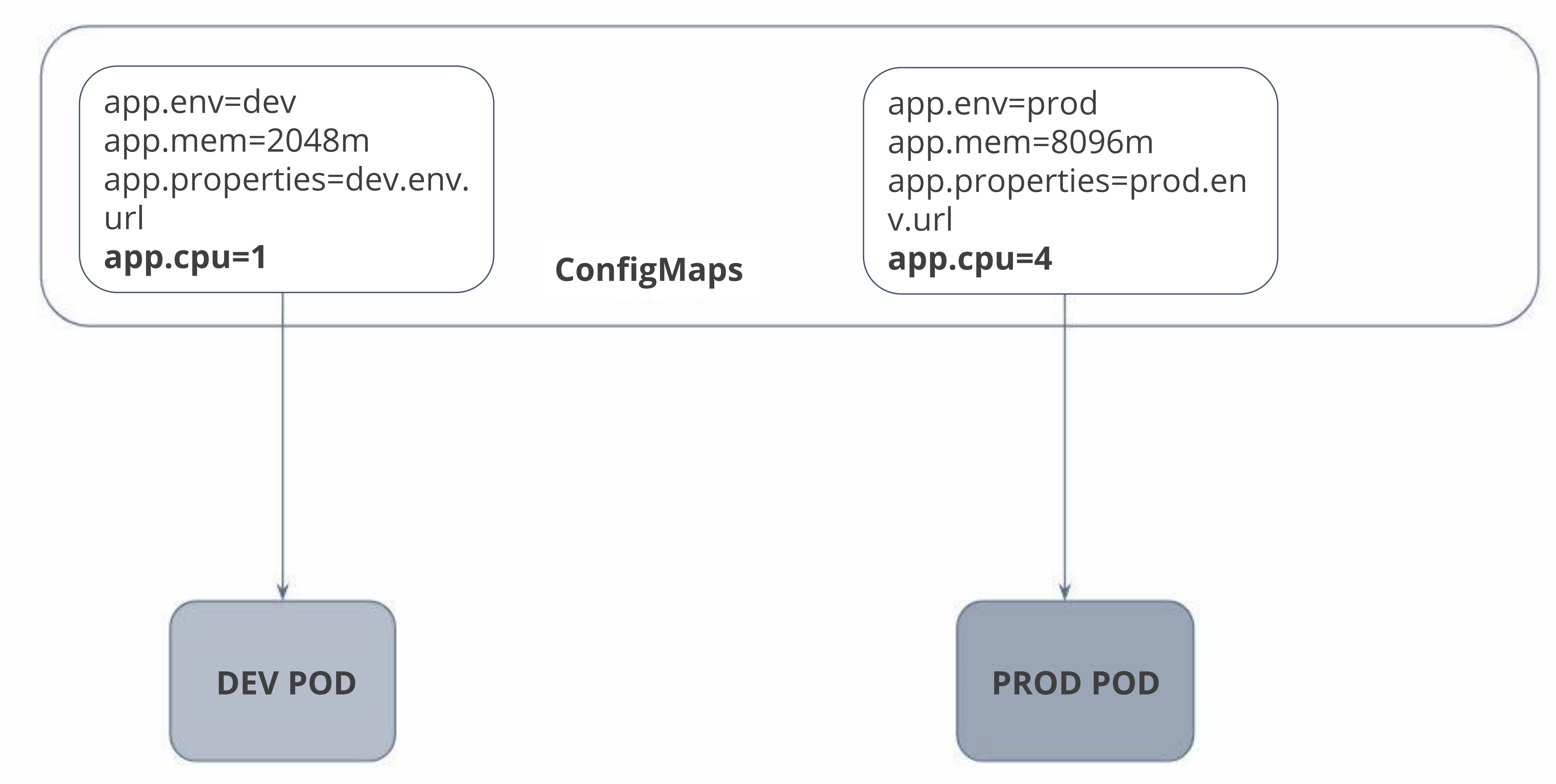
Configurations

Overview of ConfigMaps:

- A **ConfigMap** is an API object used to store non-confidential data in key-value pairs.
- Pods can use it as environment variables, command-line arguments, or configuration files in a volume.
- Users can decouple environment-specific configuration from their container images using a ConfigMap.
- It makes configurations easier to change and manage and prevents hardcoding configuration data to pod specifications.

Configurations

Overview of ConfigMaps:



Configurations

Overview of Secrets:

- A **Secret** is an object that contains a small amount of sensitive data such as a password, token, or key.
- A secret can be used with a pod in three ways:
  - As files in a volume mounted on one or more of its containers
  - As the container environment variable
  - By the kubelet when pulling images for the pod
- The name of a Secret object must be a valid **DNS subdomain name**.
- The keys of data and stringData must contain alphanumeric characters, hyphen (-), underscore (_) or period (.).

Configurations

Overview of Secrets:

Kubernetes Secret

db_user=admin
db_password=admin123

Create a ConfigMap and Secret for a Pod
Assisted Practice

Problem Statement: Your manager has asked you to create a configmap and a secret and configure a Pod to use them.

Steps to Perform:

1. Create a configuration file for configmap
2. Create a configuration file for a pod and configure it to use the configmap
3. Run the newly created pod and list the items of configmap
4. Create a configuration file for secrets
5. Create a configuration file for a pod and configure it to use the secret
6. Run the newly created pod and list the items of the secret

RBAC (Role-Based Access)

Kubernetes Grants

Overview of Kubernetes Grants:

- A Kubernetes **grant** is made up of a subject, role, and namespace. It defines which user has what level of access to particular resources.
- A **subject** can be a user, team, organization, or service account.
- A **role** defines what operations can be performed by whom.
- A **namespace** is a group of cluster resources and acts as a logical area for a cluster.
- Kubernetes comes with a default namespace for the cluster objects, plus two more namespaces for **system** and **public resources**.

RBAC Authorization

Overview of RBAC:

- **Role-Based Access Control** (RBAC) is a method of regulating access to a computer or network resources based on a subject's role.

- RBAC authorization uses the **rbac.authorization.k8s.io** API group to allow users to dynamically configure policies through the Kubernetes API.

- It can be enabled by starting the API server command with the **--authorization-mode** flag set to a comma-separated list of plug-ins including **RBAC.**
  Example: *kube-apiserver --authorization-mode=Example,RBAC --other-options --more-options*

Key Takeaways

- Kubernetes is an orchestration tool for managing containers and scaling the requirements, availability, and deployment patterns.

- kubectl is a CLI tool to interact with the API server and it runs commands for the Kubernetes clusters.

- Kubernetes Service is an abstraction which acts as a gateway to communicate with a set of pods.

- ClusterIP, NodePort, LoadBalancer, and ExternalName are the four ServiceTypes.

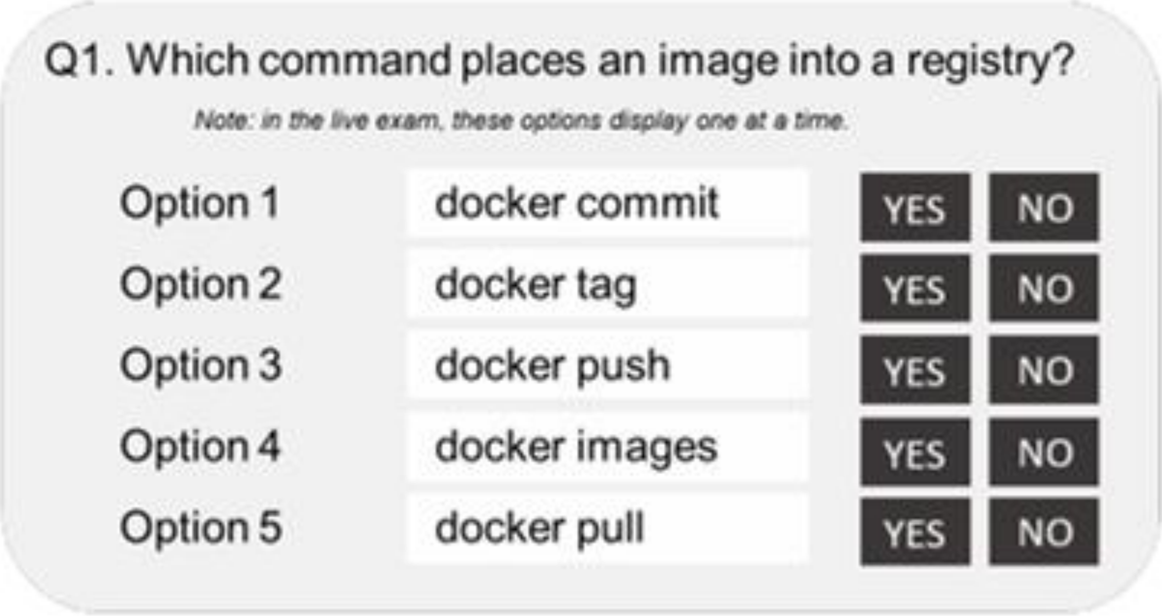
- PersistenVolumes are the resources in a cluster and PersistentVolumeClaims are the requests for those resources.

- A Kubernetes grant is made up of a subject, role, and namespace.

Course-end Assessment Instructions

Test Paper Instructions

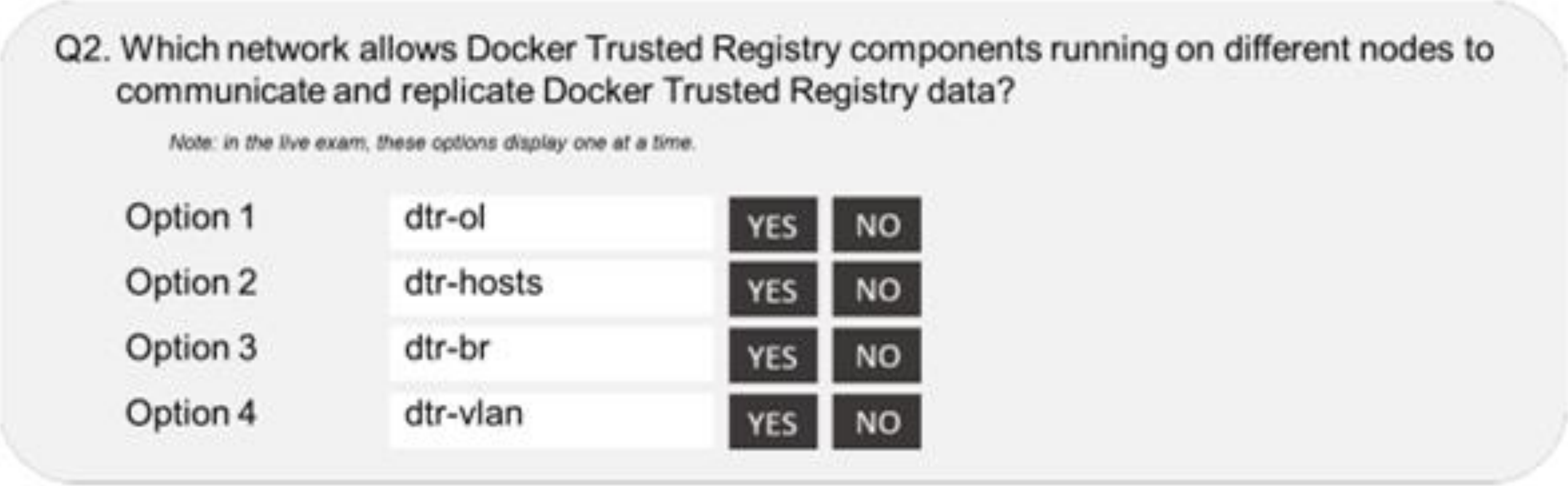
- Docker Enterprise is acquired by Mirantis and the exam format has been changed, the new exam format has DOMC (Discrete Option Multiple Choice) questions as shown in the following image:



Q1. Which command places an image into a registry?

Note: in the live exam, these options display one at a time.

Option 1	docker commit	YES	NO
Option 2	docker tag	YES	NO
Option 3	docker push	YES	NO
Option 4	docker images	YES	NO
Option 5	docker pull	YES	NO



Q2. Which network allows Docker Trusted Registry components running on different nodes to communicate and replicate Docker Trusted Registry data?

Note: in the live exam, these options display one at a time.

Option 1	dtr-ol	YES	NO
Option 2	dtr-hosts	YES	NO
Option 3	dtr-br	YES	NO
Option 4	dtr-vlan	YES	NO

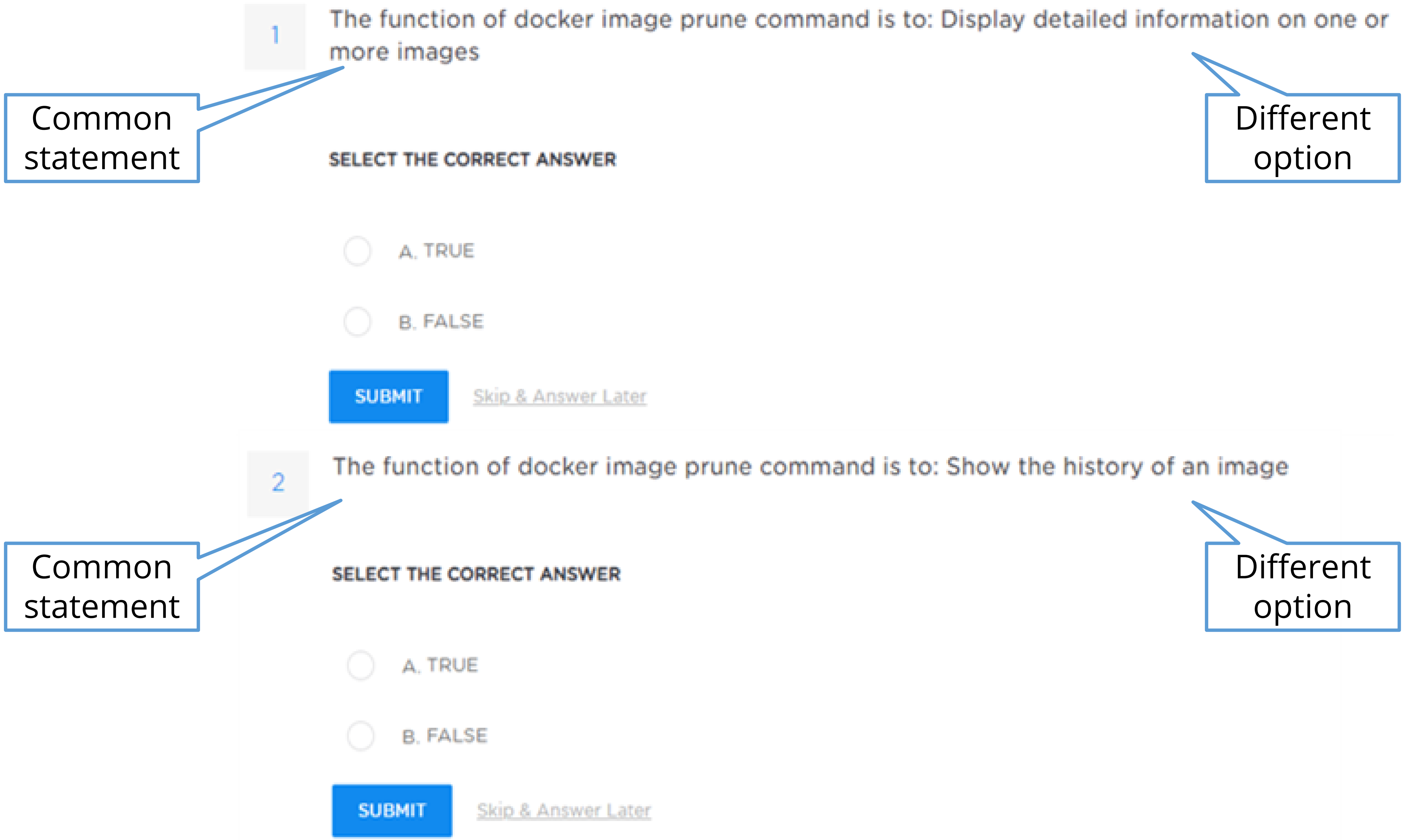
Test Paper Instructions

- To provide a similar kind of experience, a new test paper format is available on the learning platform.

- In the new test paper format, **168 questions** will be of a new format and are grouped in the sets of four questions each. Last **13 questions** will be of MCQ format.

- Each set of four questions will have a common statement and the following options will be different for each question. Learners can select **True or False** for each question.

Test Paper Instructions



1 The function of docker image prune command is to: Display detailed information on one or more images

SELECT THE CORRECT ANSWER

○ A. TRUE

○ B. FALSE

2 The function of docker image prune command is to: Show the history of an image

SELECT THE CORRECT ANSWER

○ A. TRUE

○ B. FALSE

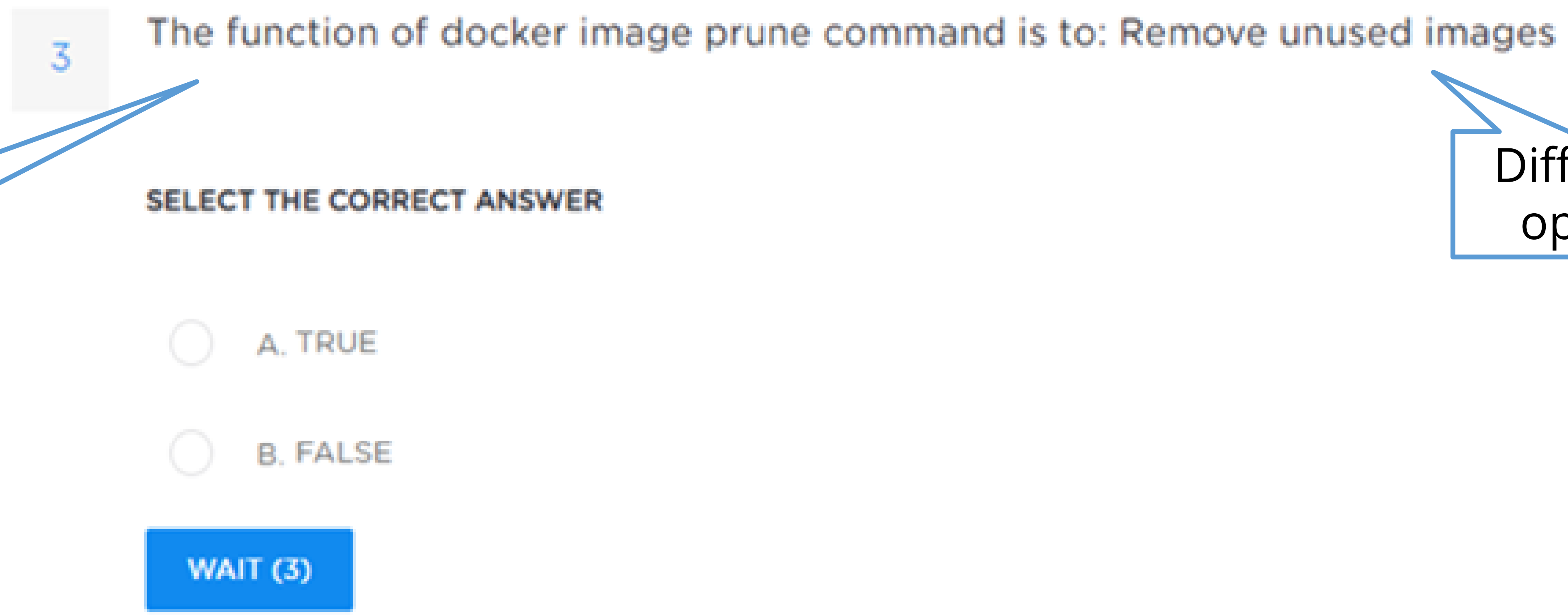
Common statement

Different option

Common statement

Different option

Test Paper Instructions



3 The function of docker image prune command is to: Remove unused images

SELECT THE CORRECT ANSWER

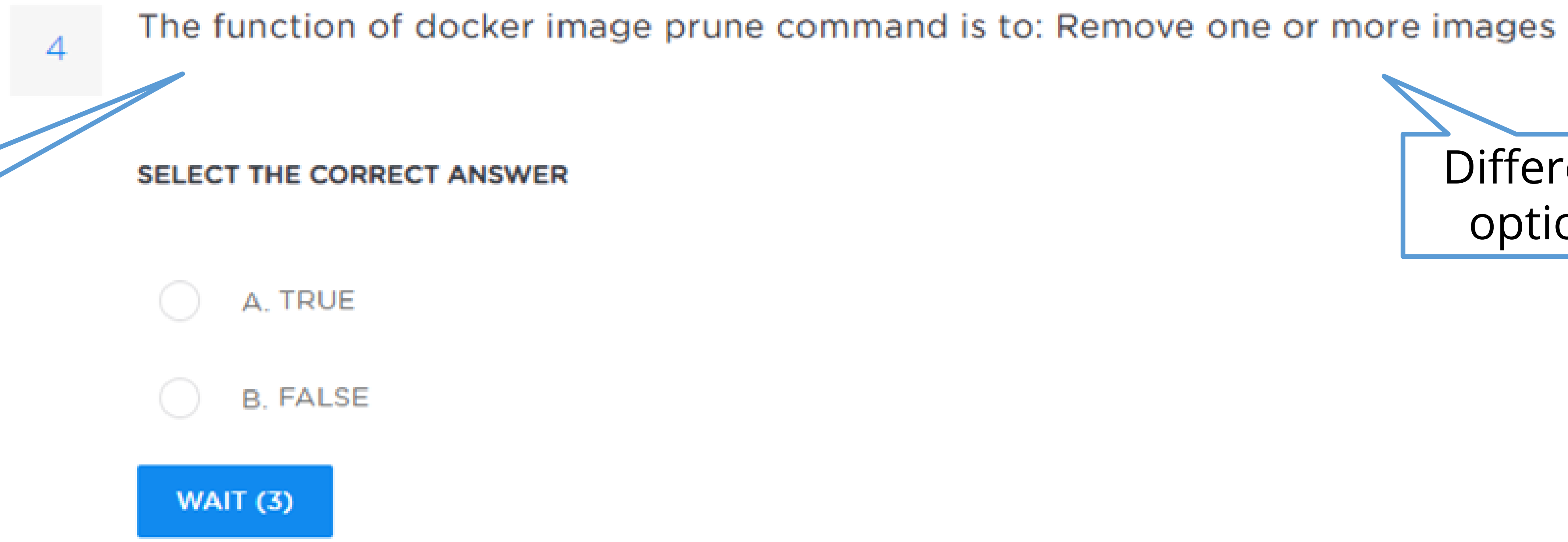
○ A. TRUE

○ B. FALSE

WAIT (3)

Common statement

Different option



4 The function of docker image prune command is to: Remove one or more images

SELECT THE CORRECT ANSWER

○ A. TRUE

○ B. FALSE

WAIT (3)

Common statement

Different option

Thank You