### **Lab Exercise 4- Creating a Kubernetes ReplicaSet with Resource Limits and Requests**

### **Objective:**

In Kubernetes, managing resource allocation is crucial to ensuring that applications run efficiently and do not overwhelm the cluster. This exercise will guide you through creating a ReplicaSet with defined resource limits and requests for each Pod.

* Understand the syntax and structure of a Kubernetes ReplicaSet definition file (YAML).
* Learn how to define resource requests and limits in a ReplicaSet.
* Create and manage a ReplicaSet with resource constraints.

Prerequisites

* Kubernetes Cluster: Have a running Kubernetes cluster (locally using Minikube or kind, or a cloud-based service).
* kubectl: Install and configure kubectl to interact with your Kubernetes cluster.
* Basic Knowledge of YAML: Familiarity with YAML format will be helpful for understanding Kubernetes resource definitions.

**Step-by-Step Guide**

**Step 1: Understanding Resource Requests and Limits**

* **Resource Requests:** These are the minimum amount of CPU and memory that Kubernetes guarantees to a container. The scheduler uses these values to decide on which node to place the Pod.
* **Resource Limits:** These define the maximum amount of CPU and memory that a container is allowed to use. If a container exceeds these limits, it may be throttled (for CPU) or killed (for memory).

**Example: Defining Resource Requests and Limits**

Let's define a ReplicaSet to maintain three replicas of an Nginx web server, with specific resource requests and limits.

**Step 2: Create a ReplicaSet YAML**

Create a YAML file named nginx-replicaset-resources.yaml with the following content:

apiVersion: apps/v1 # Specifies the API version used.

kind: ReplicaSet # The type of resource being defined; here, it's a ReplicaSet.

metadata:

name: nginx-replicaset # The name of the ReplicaSet.

spec:

replicas: 3 # The desired number of Pod replicas.

selector:

matchLabels: # Criteria to identify Pods managed by this ReplicaSet.

app: nginx # The label that should match Pods.

template: # The Pod template for creating new Pods.

metadata:

labels:

app: nginx # Labels applied to Pods created by this ReplicaSet.

spec:

containers:

- name: nginx # Name of the container within the Pod.

image: nginx:latest # Docker image to use for the container.

ports:

- containerPort: 80 # The port the container exposes.

resources: # Resource requests and limits for the container.

requests:

memory: "64Mi" # Minimum amount of memory required.

cpu: "250m" # Minimum amount of CPU required (0.25 CPU).

limits:

memory: "128Mi" # Maximum amount of memory allowed.

cpu: "500m" # Maximum amount of CPU allowed (0.5 CPU). by it.

**Step 3: Apply the YAML to Create the ReplicaSet**

Use the kubectl apply command to create the ReplicaSet based on the YAML file:

kubectl apply -f nginx-replicaset-resources.yaml

Verify the ReplicaSet is running and maintaining the desired number of replicas:

kubectl get replicaset

To check the Pods created by the ReplicaSet:

kubectl get pods -l app=nginx

You should see three Pods running, managed by the nginx-replicaset.

**Step 4: Managing Resource Requests and Limits**

1. Checking Resource Usage

You can describe the Pods to check their resource allocations:

kubectl describe pods -l app=nginx

Look for the Requests and Limits sections in the output to confirm that the resource constraints are applied as expected.