**Installation and Setup Guide for Kubernetes Cluster with Multiple Masters**

**RECORD OF REVISIONS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **VERSION** | | **DATE OF CHANGE** | **CHANGES MADE** |
| **FROM** | **TO** |
| 1 |  |  |  | Initial Version |
| 2 |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference** |  | **Issue No.** |  |
| **Release Date** |  | **Total Number of pages** | 12 |
|  | **Name** | **Signature** | **Date** |
| **Prepared by** |  |  | -11-2023 |
| **Reviewed and approved by** |  |  |  |

Contents

[What is Kubernetes: 4](#_Toc150443713)

[Namespace: 6](#_Toc150443714)

[Creating Namespaces 6](#_Toc150443715)

[PV (Persistent Volume) and PVC (Persistent Volume Claim): 7](#_Toc150443716)

[Persistent Volume (PV): 7](#_Toc150443717)

[Creating a Persistent Volume (PV) 7](#_Toc150443718)

[Persistent Volume Claim (PVC): 7](#_Toc150443719)

[Creating a Persistent Volume Claim (PVC) 7](#_Toc150443720)

[Pod: 8](#_Toc150443721)

[Deployment: 8](#_Toc150443722)

[Creating Deployment 9](#_Toc150443723)

[Service 9](#_Toc150443724)

[Creating Service 10](#_Toc150443725)

[ReplicaSet: 10](#_Toc150443726)

[Desired Replicas: 10](#_Toc150443727)

[Selector: 10](#_Toc150443728)

[Pod Template: 10](#_Toc150443729)

[Self-Healing: 11](#_Toc150443730)

[Creating ReplicaSet 11](#_Toc150443731)

[StatefulSets: 11](#_Toc150443732)

[Stable Network Identifiers: 11](#_Toc150443733)

[Stable Storage: 12](#_Toc150443734)

[Ordered Deployment and Scaling: 12](#_Toc150443735)

[Headless Service: 12](#_Toc150443736)

[Creating StatefulSet 13](#_Toc150443737)

[ConfigMaps: 13](#_Toc150443738)

[Creating ConfigMap 14](#_Toc150443739)

[Using ConfigMap in a Pod: 14](#_Toc150443740)

[Secrets: 14](#_Toc150443741)

[Creating a Secret 14](#_Toc150443742)

[Using Secret in a Pod: 15](#_Toc150443743)

**Introduction**

# What is Kubernetes:

* Kubernetes is an open-source container orchestration engine for automating deployment, scaling, and management of containerized applications.
* A Kubernetes cluster is a set of machines (nodes) that collectively run a set of containerized applications. It consists of at least one master node and one or more worker nodes.

**Master Node**:

* + The master node is responsible for managing the cluster and making global decisions. It hosts components like the API server, controller manager, scheduler, and etcd (a distributed key-value store used for cluster state).

**Worker Node**:

* + Worker nodes are the machines where the application containers run. They are managed by the master node and are responsible for running the actual workloads.

**Multi-Master Kubernetes Blueprint:**

A diagram of a group of people

Description automatically generated

**Prerequisites**

**Server Requirements:**

1. **Operating System:**
   * Linux-based distributions (e.g., Ubuntu, CentOS, Debian) are commonly used and recommended.
2. **Hardware:**
   * Sufficient CPU and RAM resources based on your expected workloads and data volume. Typically, at least 2 CPU cores and 4 GB of RAM are recommended for (Master) & 1 CPU and 2 GB of RAM for (Worker) small to medium setups.

**Networking**

**Network Access:**

• Ensure that the server has internet access to download software packages and updates during the installation process.

**Implementation Plan**

# Namespace:

* Namespace is a virtual cluster inside a Kubernetes cluster. It allows you to partition and isolate resources within the same physical cluster. Namespaces are a way to divide cluster resources between multiple users, teams, or projects (e.g., Deployments, Services, etc.) and not for cluster-wide objects (e.g., Storage Class, Nodes, Persistent Volumes, etc.).

## Creating Namespaces

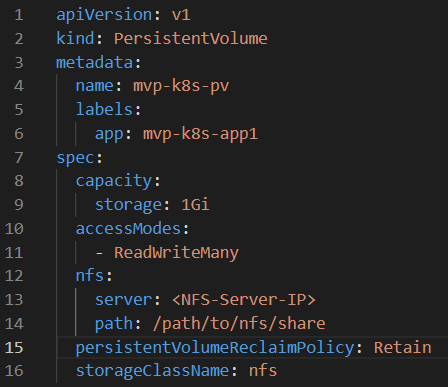
* kubectl create namespace <Mvp-DemoProject-K8s-Namespace-Dev>
* kubectl create namespace < Mvp-DemoProject-K8s-Namespace-Test >

# PV (Persistent Volume) and PVC (Persistent Volume Claim):

## Persistent Volume (PV):

* A Persistent Volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster, just like a node, that can be used to store data. PVs are typically used to represent physical storage resources in a cluster. These resources could be a network disk, a local disk, or a cloud storage system. PVs have a lifecycle independent of any individual pod using the PV.

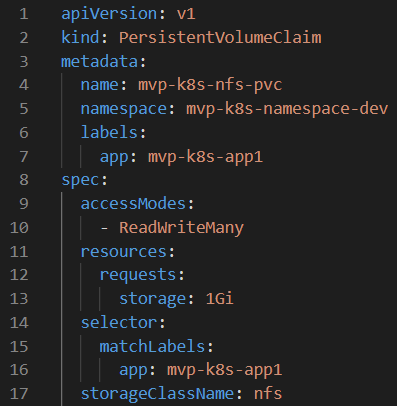
## Creating a Persistent Volume (PV)



## Persistent Volume Claim (PVC):

* A Persistent Volume Claim (PVC) is a request for storage by a user. It is used by pods to request a specific amount of storage from a PV. PVCs act as a request for storage resources. When a pod wants to use persistent storage, it makes a PVC specifying the desired storage size and access mode. The cluster then finds an appropriate PV and binds the PVC to it. Creating a Persistent Volume Claim (PVC)

## Creating a Persistent Volume Claim (PVC)



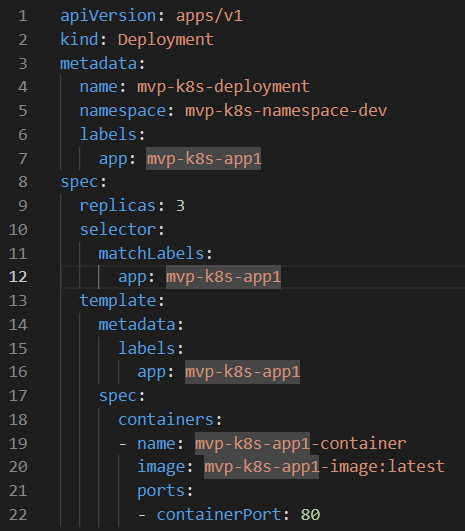
# Pod:

* Pod is the smallest deployable unit that represents a single instance of a running process in a cluster. A Pod can contain one or multiple containers, which share the same network namespace, storage, and configuration, and can communicate with each other.

# Deployment:

* Deployment is an object that provides declarative updates to applications. It allows you to describe an application's desired state, such as the number of desired replicas, and handles the deployment and scaling of the specified Pods. Deployments are a higher-level abstraction that simplifies the management and updates of applications in a Kubernetes cluster.

## Creating Deployment



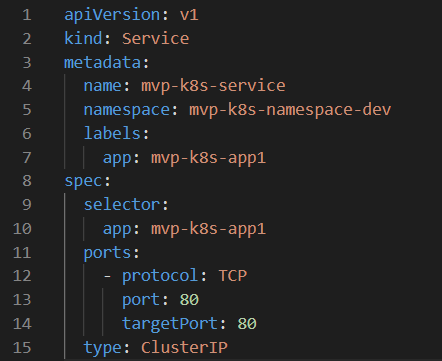
To run Deployment file, execute the command below:

$ kubectl apply -f Mvp-DemoProject-K8s-Deployment-Pod-Frontend.yml

# Service

* Service is an abstraction that defines a logical set of Pods and a policy by which to access them. It enables communication and networking between different parts of an application, allowing them to work together even if they are running in different containers, on different nodes, or in different Pods.

## Creating Service



# ReplicaSet:

* In Kubernetes, a ReplicaSet is a higher-level abstraction built on top of Pods. It ensures that a specified number of pod replicas are always running in your cluster. If Pods created by a ReplicaSet fail or are deleted, the ReplicaSet automatically replaces them to maintain the desired number of replicas.

## Desired Replicas:

* You define the desired number of replicas in the ReplicaSet's configuration. For example, if you specify that you want 3 replicas, the ReplicaSet will ensure that there are always 3 pods running.

## Selector:

* ReplicaSets use label selectors to identify the Pods they are supposed to manage. The ReplicaSet's selector field defines how to select the Pods it should manage. Pods created or modified with matching labels are considered part of the ReplicaSet.

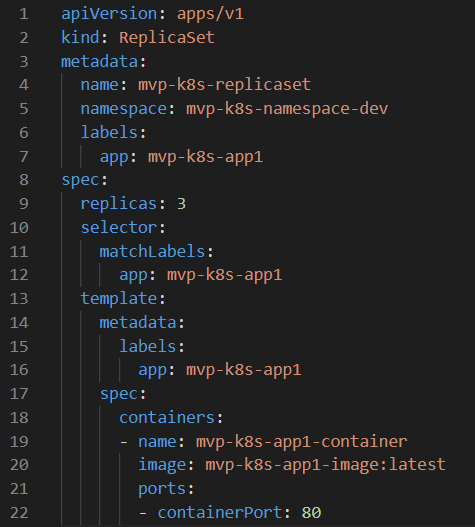
## Pod Template:

* A ReplicaSet uses a pod template to create new Pods when necessary. The pod template defines the specifications for the Pods, such as the container images, volumes, and environment variables. When a ReplicaSet needs to create new Pods, it uses the pod template as a blueprint.

## Self-Healing:

* One of the primary purposes of ReplicaSets is to provide high availability. If a Pod managed by a ReplicaSet crashes or is deleted, the ReplicaSet controller notices the change and creates a new Pod to replace it, ensuring that the desired number of replicas is always maintained.

## Creating ReplicaSet



# StatefulSets:

* StatefulSets is a higher-level abstraction in Kubernetes used to manage stateful applications. Unlike ReplicaSets or Deployments, StatefulSets provide guarantees about the ordering and uniqueness of pods. StatefulSets are particularly useful for applications that require stable network identities, stable storage, or ordered pod deployment and scaling.

## Stable Network Identifiers:

* StatefulSets give each pod a stable hostname based on its ordinal index and a stable network identity in the form of a DNS name. For example, if you have a StatefulSet named `web`, the pods will have hostnames like `web-0`, `web-1`, and so on. This stability is crucial for applications that rely on stable network identifiers.

## Stable Storage:

* StatefulSets can use Persistent Volumes (PVs) to provide stable storage for pods. When a StatefulSet is deleted, the PVs are not deleted by default, preserving the data. This is beneficial for databases and other stateful applications that need persistent storage.

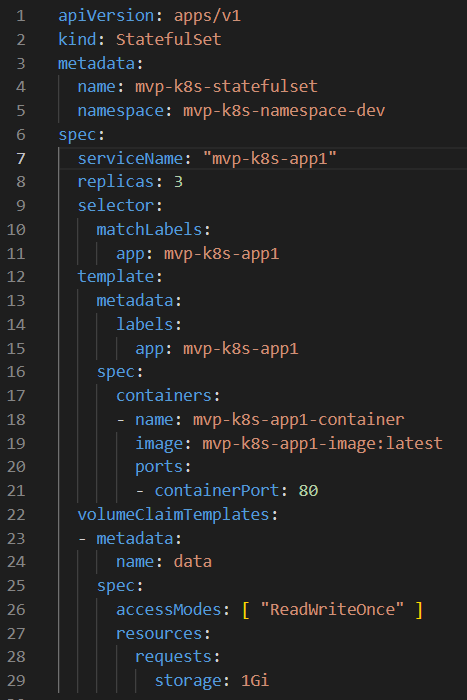
## Ordered Deployment and Scaling:

* StatefulSets deploy pods in order, starting with index 0 and incrementing sequentially. This ordered deployment and scaling can be important for applications that rely on a specific order during startup or shutdown processes.

## Headless Service:

* StatefulSets automatically create a Headless Service, which means that the service does not load balance traffic across pods. Instead, it allows direct communication with individual pods using their stable network identities.

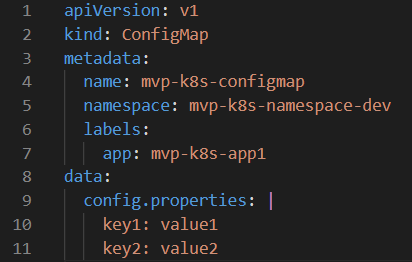
## Creating StatefulSet



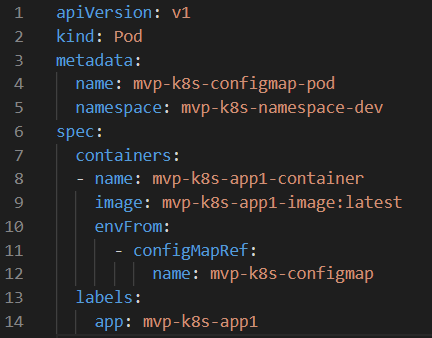
# ConfigMaps:

* ConfigMaps are used to store non-sensitive configuration data in key-value pairs. This data can be used by Pods and containers in a Kubernetes cluster. ConfigMaps are particularly useful for storing configuration files, environment variables, or any other configuration data that your application needs.

## Creating ConfigMap



# Using ConfigMap in a Pod:



# Secrets:

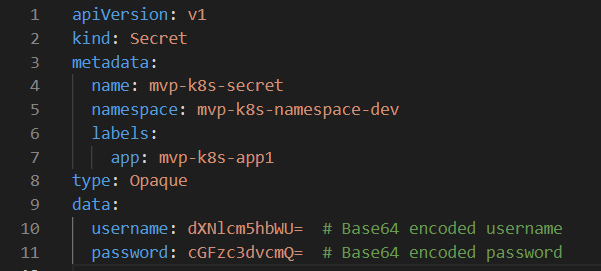
Secrets are used to store sensitive information, such as passwords, API keys, and certificates. Secrets are base64 encoded, but they are not encrypted, so it's essential to control access to secrets in your cluster.

# Creating a Secret

for encoding the required secret

syntax:

$ echo -n "secret" | base64 -i -  
example:  
$ echo -n "username" | base64 -i -  
$ echo -n "password" | base64 -i –



## Using Secret in a Pod:

