Assignment – 3

Tasks:

What does YAML stand for, and what is its role in Kubernetes resource definition? Provide a brief explanation.

YAML stands for "YAML Ain't Markup Language." It is a human-readable data serialization standard that is commonly used for configuration files. In the context of Kubernetes, YAML is used for defining resources such as pods, deployments, services, etc.

YAML files in Kubernetes serve as a declarative way to define the desired state of Kubernetes objects. These files specify the configuration settings for various resources, including their properties and relationships with other resources. Kubernetes reads these YAML files and uses the information within them to create, update, or delete the corresponding resources in the cluster.

Kubernetes reads these YAML files and acts upon them, creating, updating, or deleting resources as necessary to ensure the cluster matches the desired state described in the YAML files. In essence, YAML serves as a declarative way to configure and manage Kubernetes resources.

List the key components of a typical YAML file for defining Kubernetes resources. Explain the purpose of each component.

A typical YAML file for defining Kubernetes resources consists of several key components, each serving a specific purpose:

API Version: This field specifies the version of the Kubernetes API you are targeting. It helps Kubernetes understand how to interpret the rest of the resource definition. For example, you might see apiVersion: v1 for basic resource types

Kind: The kind field specifies the type of resource you are defining. For instance, it can be kind: Pod, kind: Service, kind: Deployment, or any other supported Kubernetes resource type.

Metadata: The metadata section includes information about the resource, such as its name, labels, and annotations. Metadata helps identify and manage the resource.

Spec: This section defines the desired state of the resource. It contains the configuration parameters specific to the resource type. For example, a pod's spec section may include the container image, environment variables, and resource limits.

Status: The "status" field is not typically included in the YAML file written by users. Instead, Kubernetes updates this field to reflect the current state of the resource. It provides information about the resource's status, such as conditions, observed state, and other runtime details.

Other optional fields: Depending on the type of resource being defined, there may be additional optional fields specific to that resource type. These fields provide further customization and configuration options. Examples include "selector" for Services, "replicas" for Deployments, and "containers" for Pods.

Review the provided YAML example for a Kubernetes Pod (as shown in the blog). Break down the YAML structure, highlighting the apiVersion, kind, metadata, and spec sections.

apiVersion:

Specifies the version of the Kubernetes API that the resource is built against. In this example, it's v1.

kind:

Indicates the type of Kubernetes resource being defined. Here, it's a Pod.

metadata:

Contains information about the resource such as its name, labels, and annotations.

In this example:

name: Specifies the name of the Pod as my-pod.

spec:

Describes the desired state of the resource, including its specification.

In this example:

containers: Specifies the containers to be run within the Pod.

name: Assigns a name to the container, which is my-container.

image: Specifies the Docker image to be used for the container, here it's nginx:latest.

Here's the breakdown of the YAML file:

apiVersion: v1

kind: Pod

metadata:

name: my-pod

spec:

containers:

- name: my-container

image: nginx:latest

Create a simple YAML file for defining a Kubernetes Service. The Service should have the following properties: apiVersion of v1, kind of Service, metadata with the name "my-service," and a spec section that exposes a service for port 80, targeting a pod named "my-pod."

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: myapp

ports:

- port: 80

targetPort: 80

Using the provided YAML file above one apply it to your Kubernetes cluster using the kubectl apply command. Ensure that the service is created successfully.

```
C:\Users\promact\Desktop>kubectl apply -f my-service.yaml
service/my-service created
```

```
C:\Users\promact\Desktop>kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   2m11s
my-service   ClusterIP   10.107.115.143   <none>        80/TCP    24s
```