Assignment-3

Tasks:

1. What does YAML stand for, and what is its role in Kubernetes resource definition? Provide a brief explanation.
2. List the key components of a typical YAML file for defining Kubernetes resources. Explain the purpose of each component.
3. 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.
4. 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 ex poses a service for port 80, targeting a pod named "my-pod."
5. 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.

1. YAML stands for "YAML Ain't Markup Language." It is a human-readable data serialization language commonly used for configuration files and data exchange formats. In Kubernetes, YAML is used to define resources such as pods, deployments, services, etc.

- YAML's role in Kubernetes resource definition is to provide a structured and readable way to describe the desired state of Kubernetes objects. It allows users to define configurations for various Kubernetes resources in a declarative manner, specifying attributes such as metadata, desired state, and specifications for each resource.

2. apiVersion: Specifies the version of the Kubernetes API that the object uses. It defines the schema of the object and how it should be processed by the Kubernetes API server.

kind: Specifies the type of Kubernetes object being created, such as Pod, Deployment, Service, etc. The "kind" field determines the behavior of the object and how it will be handled by Kubernetes.

metadata: Contains metadata for the Kubernetes object, including the name, namespace, labels, and annotations. Metadata provides information about the object to Kubernetes and is used for identification and organization purposes.

spec: Contains the desired state of the Kubernetes object. It defines the configuration parameters and settings that Kubernetes should apply to the object in order to achieve the desired state. The structure of the "spec" field varies depending on the type of Kubernetes object being defined.

3.

```
apiVersion: v1
kind: pod
metadata:
    name:my-pod
spec:
    containers:
        -name: my-container
         Image: nginx:latest
```

apiVersion: This field specifies the version of the Kubernetes API that the object uses. In this example, it's v1, indicating the core API group version.

kind: The kind field specifies the type of Kubernetes object being created. Here, it's Pod, indicating that we're defining a Pod object.

metadata: The metadata section contains information about the Pod, including its name (my-pod). Metadata is used for identifying and organizing resources within Kubernetes.

spec: The spec section defines the desired state of the Pod. It contains specifications for the Pod's containers, volumes, and other attributes. In this example, under spec, we define the container(s) running within the Pod.

containers: This sub-section under spec defines the containers that should run within the Pod. It's a list of containers, and in this case, there's only one container defined.

name: The name field specifies the name of the container (my-container). This name is used to identify the container within the Pod.

image: The image field specifies the Docker image to use for the container. Here, it's nginx:latest, which means it will use the latest version of the Nginx Docker image. This image will be pulled from the default Docker registry (Docker Hub) unless otherwise specified.

4. Here we have created a service as per requirements which has apiVersion, kind, v1, metadata, Service type as "LoadBalancer" and a spec section which exposes port 80 and target pod is "my-pod".

Assignment3.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  selector:
    app: my-pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80

```

5. Use kubectl apply command to apply the file. After that use kubectl get services to check whether service has been created or not.

```
PS C:\Users\venis\OneDrive\Documents\kubernetes\project1> kubectl apply -f ./Assignment3.yml
service/nginx-service created
PS C:\Users\venis\OneDrive\Documents\kubernetes\project1> kubectl get services
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP      10.96.0.1       <none>        443/TCP        3s
nginx-service   LoadBalancer   10.106.121.83   <pending>     80:31998/TCP   5s
```