

Assignment 3

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 exposes 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. 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 language that is often used for writing configuration files**. YAML uses indentation to represent data structures and has become the standard for defining Kubernetes resources.

In the context of Kubernetes, YAML plays a crucial role in defining resources. Here's how:

- **Resource Definition:** Kubernetes uses YAML as a way for users to express configurations. This is where you define your Kubernetes resources such as Pods, Deployments, and Services.
- **Declarative Configuration:** Kubernetes follows a declarative model, and YAML files provide a convenient way to declare the desired state of your resources.
- **Portability:** With YAML, you can define your application's resources and configurations in a file, which can be version controlled and easily moved across environments.
- **Ease of Use:** YAML's human-readable format makes it easier to understand and manage complex configurations.

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

A typical YAML file for a Kubernetes resource consists of the following key components:

1. **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.
2. **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.
3. **Metadata:** The metadata section includes information about the resource, such as its name, labels, and annotations. Metadata helps identify and manage the resource.
4. **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.

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.

Break down the provided YAML example for a Kubernetes Pod:

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

- **apiVersion:** This is the version of the Kubernetes API you're using to create this object. In this case, it's `v1`.
- **kind:** This field specifies the type of the Kubernetes object you want to create. Here, it's a `Pod`.
- **metadata:** This section is used to provide additional information about the object. Here, it includes the name of the Pod, which is `my-pod`.
- **spec:** This is where you specify the desired state of the object. In this case, it includes the specification for the containers that should be running in the Pod. Here, there's one container named `my-container` that uses the `nginx:latest` image.

The structure and fields of a Kubernetes YAML file can vary depending on the kind of object you're defining. The above breakdown is specific to a Pod object.

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 exposes a service for port 80, targeting a pod named "my-pod."

1. Create a new my-pod.yaml file and add the following:

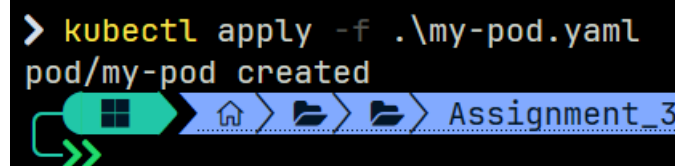
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-pod
  ports:
    - port: 80
      targetPort: 80
```

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. Create a new file named my-pod.yaml and add the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-pod
spec:
  containers:
    - name: my-pod
      image: nginx:1.25.4-alpine
      resources:
        limits:
          memory: "128Mi"
          cpu: "500m"
      ports:
        - containerPort: 80
```

2. Now apply my-pod.yaml
kubectl apply -f .\my-pod.yaml



```
> kubectl apply -f .\my-pod.yaml
pod/my-pod created
```

The screenshot shows a terminal window with a dark background. The command `kubectl apply -f .\my-pod.yaml` is entered, and the output is `pod/my-pod created`. Below the terminal output, there is a Windows taskbar with icons for a file explorer, a folder, and a document titled "Assignment_3".

```
> kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-pod	1/1	Running	0	23m

- Now apply my-service.yaml, use the following command:
kubectl apply -f ./my-service.yaml

```
> kubectl apply -f ./my-service.yaml
```

```
service/my-service created
```

```
> kubectl get services -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	2d3h	<none>
my-service	ClusterIP	10.104.17.103	<none>	80/TCP	27m	app=my-pod

- Now let's check if it is working, by enter the following command:
minikube service my-service

```
> minikube service my-service
```

NAMESPACE	NAME	TARGET PORT	URL
default	my-service		No node port

```
🐱 service default/my-service has no node port
🚀 Starting tunnel for service my-service.
```

NAMESPACE	NAME	TARGET PORT	URL
default	my-service		http://127.0.0.1:65018

```
🌐 Opening service default/my-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org
Commercial support is available at nginx.com.

Thank you for using nginx.