KUBERNETES MANIFESTS

Lesson Plan for learning manifest
Prepared by Mr. Rohit Prakash

OVERVIEW & PURPOSE

A Kubernetes manifest is a YAML or JSON file used to define resources and configurations for a Kubernetes cluster. This manifest declares the state that you want Kubernetes to enforce on your cluster. Each manifest typically consists of several key components:

BASIC STRUCTURE

A Kubernetes manifest typically has these fields:

- **apiVersion**: Specifies the version of the Kubernetes API to use.
- **kind**: The type of Kubernetes resource (e.g., `Pod`, `Service`, `Deployment`, etc.).
- **metadata**: Contains information like the name, namespace, and labels for the resource.
- **spec**: Specifies the desired state of the resource, such as the container's image, ports, or replica count for deployments.

Here's a detailed breakdown of each section:

1. apiVersion

Indicates which version of the Kubernetes API to use. Different Kubernetes objects may support different API versions.

> Example:
> `apiVersion: v1`
>
> `apiVersion: apps/v1`

2. kind

The kind of resource you're defining (e.g., `Pod`, `Service`, `Deployment`, `ConfigMap`, etc.).

> Example:
> `kind: Pod`
>
> `kind: Deployment`

3. metadata

Describes the object, including its name, namespace (optional), labels, and annotations. Labels are key-value pairs that can be used for querying and organizing resources, while annotations are used to store metadata that tools and libraries can interpret.

> Example:

```
metadata:
  name: my-app
  namespace: default
  labels:
    app: my-app
  annotations:
    description: "A simple example application"
```

4. Spec

The specification defines the desired state of the resource, and this part varies greatly depending on the resource you're creating (Pod, Deployment, Service, etc.). This section

could include containers, volumes, replicas, or ports.

Here's an example for a **Pod**:

```
spec:
  containers:
    - name: nginx-container
      image: nginx:1.14.2
      ports:
        - containerPort: 80
```

For a **Deployment**, you could specify the number of replicas and the container configuration:

```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

```
    ports:
    - containerPort: 80
```

Kubernetes Resource Types

1. **Pod Manifest** A Pod is the smallest deployable unit in Kubernetes. A Pod manifest defines one or more containers that should be scheduled on the same node.

Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx-container
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

2. **Deployment Manifest** A Deployment ensures a certain number of pod replicas are running at all times. You can specify a strategy for rolling updates or rollbacks.

Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
```

```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

3. **Service Manifest** A Service defines how to expose a set of Pods as a network service. Services can be of different types, like ClusterIP, NodePort, and LoadBalancer.

Example:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
```

```
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP
```

4. **ConfigMap Manifest** A ConfigMap is used to store configuration data for your applications. It allows you to decouple environment-specific configurations from your container images.

Example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  key1: value1
  key2: value2
```

5. **Secret Manifest** A Secret stores sensitive information, such as passwords, tokens, or keys, which should not be exposed in the code.

Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
```

```
type: Opaque

data:

  username: YWRtaW4=  # base64-encoded value of "admin"

       password: MWYyZDFlMmU2N2Rm  # base64-encoded value of
     "my_password"
```

Additional Key Fields

Selector: Used in resources like `Service` or `Deployment` to select Pods based on labels. It connects different components (e.g., connecting a Service to the correct Pod).
Example:

```
spec:

  selector:

    app: my-app
```

Affinity/Anti-Affinity: Used to specify how Pods should be scheduled on nodes. For example, you can use node affinity to ensure that Pods are placed on nodes with specific labels.

Labels: **labels** are key-value pairs attached to resources like Pods, Nodes, Services, and ConfigMaps. Labels provide a way to organize and select objects based on meaningful attributes. These are widely used to manage and group resources dynamically.

```
NOTE: Additional key fields in detail will be shared later.
```