



Kubernetes

Manifest File Overview

overview

A manifest file in Kubernetes is a configuration file used to define the desired state of various resources within a Kubernetes cluster.

These files are written in **YAML** or **JSON** format and provide a declarative way to specify the configuration of pods, services, deployments, and other Kubernetes objects.

Here's a detailed breakdown of the key components and concepts related to manifest files in Kubernetes:

1. Basic Structure of a Manifest File

A typical manifest file consists of the following sections:

- **apiVersion:** Specifies the version of the Kubernetes API to use.
- **kind:** Defines the type of Kubernetes object (e.g., Pod, Service, Deployment).
- **metadata:** Contains metadata about the object, such as its name, namespace, and labels.
- **spec:** Describes the desired state of the object.



2. Common Kubernetes Objects in Manifest Files

a. Pod

A Pod is the smallest and simplest Kubernetes object.

It represents a single instance of a running process in your cluster.

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

b. Deployment

A Deployment ensures that a specified number of pod replicas are running at any given time.


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```



c. Service

A Service defines a logical set of Pods and a policy by which to access them.


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

3. Detailed Components

Metadata

The metadata section includes attributes like name, namespace, labels, and annotations.



```
metadata:
```

```
  name: my-pod
```

```
  namespace: default
```

```
  labels:
```

```
    app: my-app
```

```
  annotations:
```

```
    description: "This is my pod"
```


- name: Unique name for the object.
- namespace: The namespace the object belongs to (default is default).
- labels: Key-value pairs for organizing and selecting objects.
- annotations: Key-value pairs for storing arbitrary metadata.

Spec

The spec section varies depending on the kind of object. It defines the desired state.

For a Pod:



```
spec:
```

```
  containers:
```

```
    - name: my-container
```

```
      image: nginx:1.14.2
```

```
      ports:
```

```
        - containerPort: 80
```


For a Deployment:

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

For a Service:

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

4. Advanced Topics

ConfigMaps and Secrets


These are used to manage configuration data and sensitive information.

ConfigMap:

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

Secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDF1MmU2N2Rm
```

b. Volumes and Persistent Storage

Volumes are used to provide storage to Pods.


```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx:1.14.2
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: my-volume
  volumes:
  - name: my-volume
    persistentVolumeClaim:
      claimName: my-pvc
```

c. StatefulSets

Used for stateful applications, providing stable network identities and persistent storage.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-statefulset
spec:
  serviceName: "my-service"
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:1.14.2
          ports:
```



```
    - containerPort: 80
volumeClaimTemplates:
- metadata:
    name: my-pvc
  spec:
    accessModes: [ "ReadWriteOnce" ]
    resources:
      requests:
        storage: 1Gi
```

5. Applying and Managing Manifest Files

You can apply a manifest file using the `kubectl` command:

```
kubectl apply -f my-manifest.yaml
```

To update an object, modify the manifest file and reapply it with the same command. You can also delete resources defined in a manifest file:

```
kubectl delete -f my-manifest.yaml
```


Conclusion

Manifest files in Kubernetes are essential for defining the desired state of your applications and infrastructure in a declarative manner. Understanding the structure and components of these files allows you to effectively manage and scale your applications in a Kubernetes environment.