HPA Pod Object and Horizontal Autoscaling in Kubernetes

Horizontal Pod Autoscaler (HPA) in Kubernetes is used to scale pods horizontally based on resource utilization, such as CPU or memory usage. This helps ensure that your application can handle varying loads effectively by scaling up or down automatically.

Steps to Set Up Horizontal Autoscaling:

1. **Manifest File**:
   - Write a manifest file for each service to define its deployment and scaling configuration.
2. **Resource Requests and Limits**:
   - Specify resource requests (minimum required) and limits (maximum allowed) in the `resources` section of the manifest file.

Example:
yaml
Copy code
```yaml
resources:
  requests:
    memory: "128Mi"
    cpu: "250m"
  limits:
    memory: "256Mi"
    cpu: "500m"
```

   -
     - **CPU (250m)**: Represents 250 millicores, or 25% of a CPU.
     - **Memory (128Mi)**: Represents 128 Mebibytes.
3. **Metrics Server**:
   - Install the **Metrics Server**, which collects resource usage data (CPU, memory, etc.) across pods and nodes. HPA relies on this data for scaling.
   - The Metrics Server can be installed via manifests available on its GitHub repository.
4. **HPA Configuration**:
   - Define an HPA object in a manifest file to specify scaling policies.

Example:
yaml
Copy code
```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
```

```
  apiVersion: apps/v1
  kind: Deployment
  name: my-app
minReplicas: 1
maxReplicas: 10
metrics:
- type: Resource
  resource:
    name: cpu
    target:
      type: Utilization
      averageUtilization: 80
```

-
- This configuration scales the my-app deployment between 1 and 10 replicas based on CPU utilization.

Secrets and ConfigMaps in Kubernetes

What Are Secrets and ConfigMaps?

1. **Secrets**:
    - Used to store sensitive information like passwords, API keys, and tokens.
    - Secrets are stored in Base64-encoded form but should be encrypted in transit and at rest (e.g., by encrypting etcd).
2. **ConfigMaps**:
    - Used to store non-sensitive configuration data in key-value pairs, such as environment variables or application settings.

Why Use Secrets and ConfigMaps?

- **Secrets**: Avoid exposing sensitive data in application manifests or code.
- **ConfigMaps**: Centralize non-sensitive configurations to make applications more dynamic and easier to manage.

Commands for Handling Secrets:

Encrypting:
bash
Copy code
```
echo -n "password" | base64
```

-

Decrypting:
bash
Copy code
```
echo -n "cGFzc3dvcmQ=" | base64 -d
```

-
    - -n ensures no newline is added, which is critical for correct encoding/decoding.

Example Secret Manifest:
yaml
Copy code
```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: dXNlcm5hbWU= # Base64-encoded username
  password: cGFzc3dvcmQ= # Base64-encoded password
```

Example ConfigMap Manifest:
yaml
Copy code
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  app.properties: |
    setting1=value1
    setting2=value2
  log.level: INFO
```

Using Secrets and ConfigMaps in Pods:
Environment Variable Example:
yaml
Copy code
```
env:
  - name: SECRET_KEY
    valueFrom:
      secretKeyRef:
        name: my-secret
    key: password
```

-

File Mount Example:
yaml
Copy code

```
volumeMounts:
  - name: config-volume
    mountPath: /etc/config
volumes:
  - name: config-volume
    configMap:
```

-
  ```
        name: my-config
  ```