Kubernetes Cheatsheet for DevOps Engineers

Table of contents

Kubernetes (often abbreviated as K8s) is an open-source container orchestration platform that automates deployment, scaling, and operations of application containers across clusters of hosts. Here's a complete cheatsheet covering the basics, commands, and real-life scenarios.

1. Basic Concepts in Kubernetes

- **Pod**: The smallest deployable unit in Kubernetes, which can contain one or more containers.
- **Node**: A worker machine (VM or physical machine) that runs pods.
- **Cluster**: A set of nodes managed by a Kubernetes control plane.
- **Namespace**: A way to divide cluster resources between multiple users.
- **Deployment**: An API object that defines a pod template for deployment with scaling and update options.
- **Service**: A stable endpoint to expose your applications (pod) to the external world or within the cluster.
- **ConfigMap**: Stores configuration files as key-value pairs.
- **Secret**: Stores sensitive information like passwords and tokens.

2. Kubernetes Basic Commands

2.1 Cluster Operations

```
# To get cluster information
kubectl cluster-info

# To check nodes in the cluster
kubectl get nodes

# To view more details about nodes
kubectl describe nodes
```

2.2 Namespace Commands

```
# List all namespaces
kubectl get namespaces

# Create a new namespace
kubectl create namespace <namespace-name>

# Switch to a different namespace
kubectl config set-context --current --namespace=<namespace-name>

# Delete a namespace
kubectl delete namespace <namespace-name>
```

Real-life Example: If you have multiple teams sharing the same Kubernetes cluster, namespaces can help isolate their environments. For example, dev and prod can be separate namespaces to prevent conflicts.

3. Managing Pods

3.1 Pod Operations

```
# Create a pod using a YAML file
kubectl apply -f <pod.yaml>

# List all pods in the current namespace
kubectl get pods

# Describe a specific pod
kubectl describe pod <pod-name>

# Check logs for a pod
kubectl logs <pod-name>

# Delete a pod
kubectl delete pod <pod-name>
```

Real-life Example: Let's say you deploy an Nginx server. Create a pod with a basic YAML file:

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

3.2 Scaling Pods with Deployment

```
# Create a deployment
kubectl create deployment <deployment-name> --image=<image>

# Scale a deployment (change replicas)
kubectl scale deployment <deployment-name> --replicas=<number>

# Update a deployment (rolling update)
kubectl set image deployment/<deployment-name> <container-name>=<new-
image>

# Rollback to a previous version
kubectl rollout undo deployment/<deployment-name>
```

Real-life Example: If you have an application that sees high traffic during specific hours, scaling up the replicas of your pods will ensure enough resources to handle the traffic.

4. Service Management

4.1 Exposing Pods with Services

```
# Create a service for a pod (ClusterIP, NodePort, or LoadBalancer)
kubectl expose pod <pod-name> --type=ClusterIP --port=80 --target-
port=8080
```

```
# List services
kubectl get services

# Describe a service
kubectl describe service <service-name>

# Delete a service
kubectl delete service <service-name>
```

Real-life Example: You've deployed a web application in a pod. To make it accessible from outside the cluster, create a **NodePort** or **LoadBalancer** service.

```
kubectl expose deployment nginx-deployment --type=NodePort --port=80
```

5. ConfigMap and Secrets

5.1 Working with ConfigMaps

```
# Create a ConfigMap from a file
kubectl create configmap <config-name> --from-file=<filename>

# View all ConfigMaps
kubectl get configmaps

# View ConfigMap details
kubectl describe configmap <config-name>

# Delete a ConfigMap
kubectl delete configmap <config-name>
```

Real-life Example: You have a configuration file (config.json) that needs to be shared between different pods. Use ConfigMaps to store and manage this file centrally.

5.2 Working with Secrets

```
# Create a secret
kubectl create secret generic <secret-name> --from-
literal=<key>=<value>
```

```
# View all secrets
kubectl get secrets

# View the decoded value of a secret
kubectl get secret <secret-name> -o jsonpath="{.data.<key>}" | base64
--decode

# Delete a secret
kubectl delete secret <secret-name>
```

Real-life Example: You need to store a password or API key that your application pods use. Secrets help manage sensitive data securely.

6. Helm Commands (Kubernetes Package Manager)

```
# Install a Helm chart
helm install <release-name> <chart-name>

# List Helm releases
helm list

# Uninstall a Helm release
helm uninstall <release-name>

# Upgrade a Helm release
helm upgrade <release-name> <chart-name>

# Rollback a Helm release
helm rollback <release-name> <revision>
```

Real-life Example: If you want to deploy a commonly used application like MySQL, Helm can simplify the process using predefined charts.

7. Resource Quota and Limits

7.1 Applying Resource Limits

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-range
spec:
  limits:
  - default:
      cpu: 500m
      memory: 512Mi
    defaultRequest:
      cpu: 200m
      memory: 256Mi
    type: Containerkubectl apply -f resource-limit.yaml
```

Real-life Example: In a shared cluster environment, it's critical to limit how much CPU or memory a pod can consume to prevent it from monopolizing cluster resources.

8. Ingress Management

```
# Apply Ingress manifest
kubectl apply -f ingress.yaml

# Check Ingress
kubectl get ingress

# Describe an Ingress resource
kubectl describe ingress <ingress-name>
```

Real-life Example: You have multiple services, and you want to route traffic based on different URL paths. Ingress provides HTTP and HTTPS routing to services in your cluster.

9. Managing Kubernetes RBAC

9.1 Creating a Role and RoleBinding

```
# Create a Role
kubectl create role <role-name> --verb=get,list,watch --resource=pods

# Bind the role to a user
kubectl create rolebinding <binding-name> --role=<role-name> --
user=<user-name>
```

Real-life Example: Suppose you want to allow a specific user to only list and view pods in a namespace but not create or delete them. You can use RBAC to set these permissions.

Conclusion

Kubernetes is a powerful platform, and learning to work with it efficiently can greatly enhance your ability to manage containerized applications. This cheatsheet covers the fundamental commands and real-life examples to help you understand the key concepts of Kubernetes.

In real-world scenarios, Kubernetes simplifies the deployment and scaling of applications across large clusters of nodes. The flexibility to handle various configurations (from resource quotas to RBAC) ensures it can adapt to any organizational requirement, making Kubernetes an essential tool for modern DevOps engineers.

By mastering these commands and concepts, you can confidently manage, troubleshoot, and scale applications in a Kubernetes environment.

Connect and Follow Me on Socials

LINKDIN | GITHUB |TWITTER