Date: 26.06.2025

Kubernetes Commands

Contents

1. Local Setup with minikube and kubectl

Minikube is a lightweight Kubernetes implementation that creates a single-node Kubernetes cluster on your local machine. This simplifies the local Kubernetes Development.
Kubectl is a CLI tool for K8s cluster. It is used to send request to Kube API to interact with the K8s cluster.

Minikube – Create, Stop, Delete Cluster
Kubectl – Interact with anything in the Cluster

Starting local cluster	minikube start
Verify status of local cluster	minikube status

2. Commands

Status of different components
    Syntax: kubectl get <resource>

Additional information about component
    Syntax: kubectl describe <resource>

Creating a deployment
    Syntax: kubectl create deployment NAME --image=IMAGE
    Eg: kubectl create deployment nginx-depl --image=nginx

Status of Deployment
    Syntax: kubectl get deployment
    You may also check ReplicaSet created by the deployment: kubectl get replicaset

Edit image in a deployment directly
    Syntax: kubectl edit deployment NAME
    Eg: kubectl get nginx-depl

Viewing logs
    Syntax: kubectl logs PODNAME
    Note: get name of pod by `kubectl get pods`

Accessing terminal within pod
    Syntax: kubectl exec -it PODNAME – bin/bash

Delete a deployment
    Syntax: kubectl delete deployment NAME

Creating a deployment with YAML file
    Kubectl apply -f YAMLfile

Deleting a deployment with YAML file
    Syntax: kubectl delete -f YAMLfile

3. YAML Config files

Each Configuration file has 3 parts

1. Metadata – apiVersion, kind, metadata
2. Specification – Every configuration you want to apply
3. Status – automatically generated and edited by K8s

Demo with mongo and mongo-express

mongo-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  mongo-root-username: <base64 encoded string>
  mongo-root-password: <base64 encoded string>
```

Note:
use `echo –n STRING | base64` then write that into <base64 encoded string>
Secret must be created before Deployment

mongo-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
  spec:
    containers:
    - name: mongodb
      image: mongo
      ports:
        - containerPort: 27107
      env:
      - name: MONGO_INITDB_ROOT_USERNAME
      valueFrom:
          secretKeyRef:
            name: mongodb-secret
            key: mongo-root-username
        - name: MONGO_INITDB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-password
---
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb
  ports:
  - protocol: TCP
    port: 27107
    targetPort: 27107
```

Note:
template is used to create pods within the deployment
Use ports to expose services

mongo-config.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  database_url: mongodb-service
```

Note:
Server name is same as service name

mongo-express-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express-deployment
  labels:
    app: mongo-express
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
    spec:
      containers:
      - name: mongo-express
        image: mongo-express
        ports:
        - containerPort: 8081
        env:
        - name: ME_CONFIG_MONGODB_ADMIN_USERNAME
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-username
        - name: ME_CONFIG_MONGODB_ADMIN_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-password
        - name: ME_CONFIG_MONGODB_SERVER
          valueFrom:
            configMapKeyRef:
              name: mongodb-configmap
              key: mongodb-service
---
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 8081
    targetPort: 8081
    nodePort: 30000
```

Note:
Make the service external by defining type as LoadBalancer
Assign external IP to service by running `minikube service mongo-express-service`

Run `kubectl apply –f YAMLfile` for each of the above files.

4. Namespaces

Organize resources in cluster. Virtual cluster within a cluster

Types of Namespaces

kubernetes-dashboard	Added by minikube installation.
kube-system	Not meant for user. Consists of components deployed as system processes.
kube-public	Publicly accessible data like configmap that contains cluster info. It is accessible without any authentication.
kube-node-lease	Holds info of heartbeats of node. (Availability of nodes)
default	Default namespace for all components created if it is not specified.

Create namespace

Syntax: kubectl create namespace NS
Or
Add `namespace: NS` in YAML file in `metadata` section after `name`

List resources in a NS

Syntax: kubectl api-resources –namespaced=<true/false>
True = resources bound to NS
False = resources not bound to NS

Changing active NS

Syntax: kubectl config set-contedxt --current --namespace=<NS>

5. Ingress

In production, app should be exposed to a domain.
IP+port is OK for development but not production.

Configure Ingress in minikube: minikube addons enable ingress

dashboard-ingess.yaml (in minikube)

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: dashboard-ingress
  namespace: kubernetes-dashboard
spec:
  rules:
  - host: dashboard.com
    http:
      paths:
      - backend:
        serviceName: kubernetes-dashboard
        servicePort: 80
```

Note:
Run `kubectl apply –f dashboard-ingress.yaml`
Get IP by `kubectl get ingress -n kubernetes-dashboard`
Configure to resolve IP as ‘dashboard.com’ doesn’t exist
`vi /etc/host` → add the IP address at the end.

6. Helm

Sharing of Helm Charts
- Package of pre-configured kubernetes resources.
- Either private or public

Templating Engine
- Creates `.Values` object using values.yaml
- Practical for CI/CD
- Injected as `{{ .Values.key1.key2 }}`

Deploying same apps across different environments
- For eg: You need to deploy your app in the following 3 environments
  - Dev → Staging → Prod

Helm Chart Structure

```
mychart/
  |-- chart.yaml        ← Meta info about chart name, version, dependency list, etc
  |-- values.yaml       ← Values for template file (Default values can be overridden)
  |-- charts/           ← Required Chart Dependencies
  |-- templates/        ← Actual template files
  …                     ← Readme, License file, etc
```

7. Volumes

PersistentVolume
1. Cluster Resource
2. Not Namespaced
3. Provisioned by the K8s Admin
4. Created by YAML file
   - kind: PersistentVolume
   - spec: Specification as needed

PersistentVolumeClaim
1. Claims volume from PV
2. Matches `spec` then claims that PV
3. Must be in same NS
4. Step-by-step breakdown
   a. Pod request volume through PVC
   b. Find a PV that satisfies the request
   c. Select that PV (it has actual storage backend)

StorageClass

1. Provisions persistent volume dynamically whenever PVC claims it
2. Requested by PVC
3. Step-by-step breakdown
    a. Pod request volume through PVC
    b. PVC requests to SC
    c. SC creates PV
    d. PVC connects to PV

8. StatefulSet

K8s component for Stateful apps.
Kubernetes, Docker or other containerized environments are not suitable for stateful apps.
They are much more suitable for stateless apps.