## Deploying a Web Application on Minikube: A Step-by-Step Guide

### Prerequisites

* **Minikube:** Installed and running.
* **kubectl:** Installed and configured.
* **Docker:** Installed and running.
* **A built web application Docker image:** This guide assumes you have a Docker image ready. If not, build one using a Dockerfile.
* **Basic understanding of Kubernetes concepts:** Deployment, Service.

### Steps

#### 1. Create a Deployment YAML file

A Deployment defines the desired state of your application. Create a YAML file named deployment.yaml with the following content:

YAML

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app-deployment

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: your-image-name:tag

ports:

- containerPort: 80

Replace your-image-name:tag with the actual name and tag of your Docker image.

#### 2. Create a Service YAML file

A Service exposes your application to the outside world. Create a YAML file named service.yaml with the following content:

YAML

apiVersion: v1

kind: Service

metadata:

name: my-app-service

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 80

type: **NodePort**

#### 3. Deploy the Application

Apply the Deployment and Service manifests to your Minikube cluster:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

#### 4. Check the Deployment Status

Verify that your Deployment and Pods are running:

kubectl get deployments

kubectl get pods

#### 5. Access the Application

Get the NodePort of the Service:

kubectl get services

Find the NodePort value (e.g., 30001).

Open your web browser and access the application using the following URL: http://<minikube-ip>:30001, replacing <minikube-ip> with the IP address of your Minikube instance.

### Additional Considerations

* **Persistent Storage:** If your application requires persistent storage, you'll need to create PersistentVolumeClaims (PVCs) and PersistentVolumes (PVs).
* **Ingress:** For production environments, consider using Ingress to expose your application externally.
* **Configuration Management:** Use ConfigMaps or Secrets to manage configuration values.
* **Scaling:** Adjust the replicas field in the Deployment to scale your application.
* **Debugging:** Use kubectl logs to view container logs.

### Example with a Simple Node.js Application

If you don't have a Docker image ready, here's a basic Node.js application example:

1. **Create a Node.js application:**

JavaScript

// index.js

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('Hello from Minikube!');

});

app.listen(80, () => {

console.log('Server listening on port 80');

});

1. **Create a Dockerfile:**

Dockerfile

FROM node:18-alpine

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 80

CMD [ "node", "index.js" ]

Use code [with caution.](/faq#coding)

1. **Build and push the Docker image:**

docker build -t my-node-app .

docker push my-node-app

1. \*\*Replace your-image-name:tag in the Deployment YAML with my-node-app and proceed with the steps above.

### ****Generate Deployment YAML****

To create a Deployment YAML file using the kubectl command:

#### Example:

Let's say you want to create a Deployment for an Nginx web server.

kubectl create deployment nginx-deployment --image=nginx --dry-run=client -o yaml > deployment.yaml

* **--dry-run=client**: This flag ensures that the command only simulates the creation and doesn’t actually apply it to the cluster.
* **-o yaml**: This specifies that the output should be in YAML format.
* **> deployment.yaml**: This redirects the output to a file named deployment.yaml.

After running the command, you'll have a deployment.yaml file with the Deployment configuration.

#### Sample deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

### ****Generate Service YAML****

To expose your Deployment using a Service, you can generate a Service YAML file:

#### Example:

Let's create a Service for the Nginx Deployment created above.

kubectl expose deployment nginx-deployment --port=80 --type=NodePort --dry-run=client -o yaml > service.yaml

* **--port=80**: Specifies the port on which the service will be exposed.
* **--type=NodePort**: This type of Service makes the application accessible outside of the cluster, using a port on the Minikube node.
* **--dry-run=client** and **-o yaml**: Similar to the Deployment command, these ensure that the command outputs the YAML configuration without applying it.
* **> service.yaml**: This saves the output to a file named service.yaml.

#### Sample service.yaml:

apiVersion: v1

kind: Service

metadata:

name: nginx-deployment

spec:

type: NodePort

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

nodePort: 30007

### ****Apply the YAML Files****

Once you have generated the YAML files, you can apply them to your Minikube cluster using:

bash

Copy code

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

### ****Check the Status****

You can verify that the Deployment and Service have been created by running:

kubectl get deployments

kubectl get services

This will show you the running Deployment and Service in your Minikube cluster.

### ****Accessing the Service****

For a Service of type NodePort, you can access it via the Minikube IP and the allocated node port:

minikube service nginx-deployment

This command will automatically open your default web browser and navigate to the service's external URL.

### ****Conclusion****

These steps allow you to generate and manage Kubernetes resources using kubectl commands in Minikube, making it easier to define and version-control your configurations.

### Module 3: Advanced Topics – Deep Dive

#### 1. **Service Directory**

* **Example 1: Exposing a Service using ClusterIP**

kubectl create deployment hello-world --image=gcr.io/google-samples/hello-app:1.0

kubectl expose deployment hello-world --type=**ClusterIP** --port=8080

kubectl get services

* **Example 2: Exposing a Service using NodePort**

kubectl expose deployment hello-world --type=**NodePort** --port=8080

minikube service hello-world

* **Example 3: Exposing a Service using LoadBalancer**

kubectl expose deployment hello-world --type=**LoadBalancer** --port=8080

minikube service hello-world

#### 2. **ConfigMap**

* **Example 1: Creating a ConfigMap from Literal Values**

kubectl create configmap app-config --from-literal=APP\_COLOR=blue --from-literal=APP\_MODE=production

kubectl get configmaps app-config -o yaml

* **Example 2: Using ConfigMap in a Pod**

apiVersion: v1

kind: Pod

metadata:

name: configmap-demo-pod

spec:

containers:

- name: configmap-demo-container

image: busybox

command: [ "sh", "-c", "echo The app color is $(APP\_COLOR) && sleep 3600" ]

env:

- name: APP\_COLOR

valueFrom:

configMapKeyRef:

name: app-config

key: APP\_COLOR

kubectl apply -f configmap-demo-pod.yaml

kubectl logs configmap-demo-pod

#### 3. **Ingress Controller**

* **Example 1: Enabling Ingress in Minikube**

minikube addons enable ingress

* **Example 2: Creating an Ingress Resource**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: webapp-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: hello-world.local

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: hello-world

port:

number: 8080

kubectl apply -f ingress.yaml

* **Example 3: Accessing the Service via Ingress**

echo "$(minikube ip) hello-world.local" | sudo tee -a /etc/hosts

curl http://hello-world.local

#### 4. **External DNS**

* **Example 1: Setting Up External DNS (with Google Cloud DNS)**

# Example requires setting up a GCP project and credentials

kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/external-dns/master/docs/tutorials/nginx.md

* **Example 2: Configuring DNS for Ingress**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: external-dns-ingress

annotations:

external-dns.alpha.kubernetes.io/hostname: "myapp.example.com"

spec:

rules:

- host: myapp.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: hello-world

port:

number: 8080

kubectl apply -f external-dns-ingress.yaml

#### 5. **Volumes**

* **Example 1: Static Volume Provisioning**

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-volume

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data"

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-volume

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

---

apiVersion: v1

kind: Pod

metadata:

name: volume-demo

spec:

containers:

- name: volume-demo-container

image: busybox

command: [ "sleep", "3600" ]

volumeMounts:

- mountPath: "/mnt/data"

name: volume

volumes:

- name: volume

persistentVolumeClaim:

claimName: pvc-volume

kubectl apply -f static-volume.yaml

#### 6. **Volumes Auto-Provisioning**

* **Example 1: Using Dynamic Volume Provisioning with StorageClass**

yaml

Copy code

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: standard

provisioner: kubernetes.io/minikube-hostpath

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-volume

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

storageClassName: standard

kubectl apply -f dynamic-volume.yaml

#### 7. **Pod Presets**

* **Example 1: Injecting Environment Variables using PodPresets**

apiVersion: settings.k8s.io/v1alpha1

kind: PodPreset

metadata:

name: allow-db

spec:

env:

- name: DB\_HOST

value: mysql

apiVersion: v1

kind: Pod

metadata:

name: db-pod

spec:

containers:

- name: db-container

image: mysql

command: [ "sh", "-c", "echo DB\_HOST is $(DB\_HOST)" ]

kubectl apply -f podpreset.yaml

kubectl apply -f pod.yaml

#### 8. **StatefulSets**

* **Example 1: Deploying a StatefulSet**

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: web

spec:

serviceName: "nginx"

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

name: web

kubectl apply -f statefulset.yaml

#### 9. **DaemonSets**

* **Example 1: Running a DaemonSet**

yaml

Copy code

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: node-exporter

spec:

selector:

matchLabels:

name: node-exporter

template:

metadata:

labels:

name: node-exporter

spec:

containers:

- name: node-exporter

image: prom/node-exporter

ports:

- containerPort: 9100

kubectl apply -f daemonset.yaml

#### 10. **Resource Usage Monitoring**

* **Example 1: Using Metrics Server**

minikube addons enable metrics-server

kubectl top nodes

kubectl top pods

* **Example 2: Monitoring with Prometheus**

# Installing Prometheus using Helm

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

helm install prometheus prometheus-community/kube-prometheus-stack

#### 11. **Auto Scaling**

* **Example 1: Horizontal Pod Autoscaler (HPA)**

kubectl create deployment php-apache --image=k8s.gcr.io/hpa-example

kubectl expose deployment php-apache --port=80

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

kubectl get hpa

* **Example 2: Triggering HPA with Load**

kubectl run -i --tty load-generator --image=busybox /bin/sh

while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done

#### 12. **Affinity / Anti-Affinity**

* **Example 1: Node Affinity**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-affinity

spec:

replicas: 2

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/e2e-az-name

operator: In

values:

- e2e-az1

- e2e-az2

kubectl apply -f node-affinity.yaml

#### 13. **Interpod Affinity and Anti-Affinity**

* **Example 1: Interpod Affinity**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-affinity

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

affinity:

podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

labelSelector:

matchExpressions:

- key: app

operator: In

values:

- nginx

kubectl apply -f interpod-affinity.yaml

#### 14. **Taints and Tolerations**

* **Example 1: Applying Taints and Tolerations**

kubectl taint nodes node1 key=value:NoSchedule

yaml

Copy code

apiVersion: v1

kind: Pod

metadata:

name: toleration-pod

spec:

containers:

- name: nginx

image: nginx

tolerations:

- key: "key"

operator: "Equal"

value: "value"

effect: "NoSchedule"

kubectl apply -f toleration-pod.yaml

#### 15. **Custom Resource Definitions (CRDs)**

* **Example 1: Defining a Custom Resource**

apiVersion: apiextensions.k8s.io/v1

kind: CustomResourceDefinition

metadata:

name: crontabs.stable.example.com

spec:

group: stable.example.com

versions:

- name: v1

served: true

storage: true

schema:

openAPIV3Schema:

type: object

properties:

spec:

type: object

properties:

cronSpec:

type: string

image:

type: string

replicas:

type: integer

scope: Namespaced

names:

plural: crontabs

singular: crontab

kind: CronTab

shortNames:

- ct

kubectl apply -f crd.yaml

#### 16. **Operators**

* **Example 1: Creating an Operator using Operator SDK**

operator-sdk init --domain=example.com --repo=github.com/example/memcached-operator

operator-sdk create api --group=cache --version=v1alpha1 --kind=Memcached --resource --controller

make install

kubectl apply -f config/samples/cache\_v1alpha1\_memcached.yaml

### Module 4: Kubernetes Administrator

#### 1. **The Kubernetes Master Services**

* **Example 1: Checking the Kubernetes Master Components**

kubectl get pods -n kube-system

kubectl describe pod -n kube-system kube-apiserver-minikube

kubectl describe pod -n kube-system kube-scheduler-minikube

#### 2. **Resource Quotas**

* **Example 1: Setting a Resource Quota for a Namespace**

apiVersion: v1

kind: ResourceQuota

metadata:

name: mem-cpu-quota

namespace: default

spec:

hard:

requests.cpu: "1"

requests.memory: 1Gi

limits.cpu: "2"

limits.memory: 2Gi

kubectl apply -f resource-quota.yaml

#### 3. **Namespaces**

* **Example 1: Creating and Managing Namespaces**

kubectl create namespace dev

kubectl config set-context --current --namespace=dev

kubectl get namespaces

#### 4. **User Management**

* **Example 1: Creating a Service Account**

kubectl create serviceaccount demo-sa

kubectl get serviceaccounts demo-sa -o yaml

* **Example 2: Assigning Roles to a Service Account**

kubectl create rolebinding demo-sa-admin --clusterrole=admin --serviceaccount=default:demo-sa --namespace=default

kubectl get rolebindings

#### 5. **RBAC**

* **Example 1: Creating Roles and RoleBindings**

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: default

name: pod-reader

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "watch", "list"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: read-pods

namespace: default

subjects:

- kind: User

name: jane

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: pod-reader

apiGroup: rbac.authorization.k8s.io

kubectl apply -f rbac.yaml

#### 6. **Networking**

* **Example 1: Configuring a Network Policy**

yaml

Copy code

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-nginx

namespace: default

spec:

podSelector:

matchLabels:

app: nginx

ingress:

- from:

- ipBlock:

cidr: 192.168.1.0/24

ports:

- protocol: TCP

port: 80

kubectl apply -f network-policy.yaml

#### 7. **Node Maintenance**

* **Example 1: Draining a Node**

kubectl drain <node-name> --ignore-daemonsets --force

kubectl get nodes

* **Example 2: Uncordoning a Node**

bash

Copy code

kubectl uncordon <node-name>

kubectl get nodes

#### 8. **High Availability**

* **Example 1: Setting Up a Highly Available Cluster**

# This setup requires multiple master nodes

kubectl get nodes

#### 9. **TLS on ELB using Annotations**

* **Example 1: Configuring TLS via Annotations**

apiVersion: v1

kind: Service

metadata:

name: tls-service

annotations:

service.beta.kubernetes.io/aws-load-balancer-ssl-cert: arn:aws:acm:us-west-2:123456789012:certificate/abcd1234-5678-90ab-cdef-11111EXAMPLE

service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "https"

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: 443

targetPort: 80

type: LoadBalancer

kubectl apply -f tls-service.yaml

This should provide you with more hands-on examples and code snippets to explore advanced Kubernetes topics. Let me know if you'd like to dive deeper into any specific topic or need further assistance!