Kubernetes Essentials Tutorial

Hello Minikube - Your First Cluster

What is Minikube? Minikube creates a local Kubernetes cluster for development and learning.

Setup & Start:

```
# Start minikube
minikube start

# Open the Kubernetes dashboard
minikube dashboard
```

Create Your First Deployment:

```
# Create a deployment
kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.53 -- /agnhost netexec --http-port=8080

# View deployment
kubectl get deployments

# View pods
kubectl get pods
```

Expose Your App:

```
# Create a service to expose the deployment
kubectl expose deployment hello-node --type=LoadBalancer --port=8080

# Access the service
minikube service hello-node
```

Clean up:

```
kubectl delete service hello-node
kubectl delete deployment hello-node
```

Deploying Applications with kubectl

Deployments manage Pods and ensure your application stays running. They handle scaling, rolling updates, and self-healing.

Basic kubectl Commands:

```
# Create deployment
kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1

# View deployments
kubectl get deployments

# View pods
kubectl get pods

# Create proxy to access pods
```

kubectl proxy

Access your application:

```
# Get pod name
export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')

# Access through proxy
curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME:8080/proxy/
```

Exploring Pods and Nodes

Pods are the smallest deployable units in Kubernetes, containing one or more containers.

Nodes are worker machines (virtual or physical) that run your Pods.

Key Commands:

```
# Get detailed pod information
kubectl describe pods

# View logs
kubectl logs <pod-name>

# Execute commands in pod
kubectl exec <pod-name> -- env
kubectl exec -ti <pod-name> -- bash
```

Exposing Applications with Services

Services provide stable network access to Pods, handling load balancing and service discovery.

Service Types:

- **ClusterIP**: Internal cluster access only (default)
- **NodePort**: Exposes service on each node's IP at a static port
- **LoadBalancer**: Exposes service externally using cloud provider's load balancer

Create and expose a service:

```
# Expose deployment as NodePort service
kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port=8080

# Get service details
kubectl describe services/kubernetes-bootcamp

# Get the node port
export NODE_PORT="$(kubectl get services/kubernetes-bootcamp -o go-template='{{(index .spec.ports 0).nodePort}}')"

# Access the service
curl http://"$(minikube ip):$NODE_PORT"
```

Working with Labels:

```
# Label a pod
kubectl label pods <pod-name> version=v1
```

 # Query by label
kubectl get pods -l app=kubernetes-bootcamp
kubectl get services -l app=kubernetes-bootcamp

 # Delete service by label
kubectl delete service -l app=kubernetes-bootcamp
Scaling Applications
Manual Scaling:
 # Scale to 4 replicas
kubectl scale deployments/kubernetes-bootcamp --replicas=4

 # Check deployment status
kubectl get deployments

 # View replica sets
kubectl get rs

 # Check load balancing (run multiple times)
curl http://"$(minikube ip):$NODE_PORT"
Scale down:
 # Reduce to 2 replicas
kubectl scale deployments/kubernetes-bootcamp --replicas=2
Rolling Updates
Update application version:
 # Update to version 2
kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=docker.io/jocatalin/kubernetes-bootcamp:v2

 # Check rollout status
kubectl rollout status deployments/kubernetes-bootcamp

 # Verify update
curl http://"$(minikube ip):$NODE_PORT"
Rollback if needed:
 # Rollback to previous version
kubectl rollout undo deployments/kubernetes-bootcamp

 # Check pod status after rollback
kubectl get pods
Essential kubectl Commands Summary
 # Cluster info
kubectl version
kubectl get nodes

 # Deployments
kubectl create deployment <name> --image=<image>
kubectl get deployments
kubectl describe deployment <name>
kubectl delete deployment <name>

 # Pods
kubectl get pods
kubectl describe pods <name>
kubectl logs <name>
kubectl exec -ti <name> -- bash

 # Services
kubectl expose deployment <name> --type=<type> --port=<port>
kubectl get services
kubectl describe service <name>

 # Scaling
kubectl scale deployments/<name> --replicas=<number>

 # Updates
kubectl set image deployments/<name> <container>=<image>:<tag>
kubectl rollout status deployments/<name>
kubectl rollout undo deployments/<name>

Open the Dashboard

Open the Kubernetes dashboard. You can do this two different ways:

Launch a browser | URL copy and paste

Open a **new** terminal, and run:

```
# Start a new terminal, and leave this running.
minikube dashboard
```

Now, switch back to the terminal where you ran `minikube start`.

> **Note:**
> The `dashboard` command enables the dashboard add-on and opens the proxy in the default web browser. You can create Kubernetes resources on the dashboard such as Deployment and Service.
>
> To find out how to avoid directly invoking the browser from the terminal and get a URL for the web dashboard, see the "URL copy and paste" tab.
>
> By default, the dashboard is only accessible from within the internal Kubernetes virtual network. The `dashboard` command creates a temporary proxy to make the dashboard accessible from outside the Kubernetes virtual network.
>
> To stop the proxy, run `Ctrl+C` to exit the process. After the command exits, the dashboard remains running in the Kubernetes cluster. You can run the `dashboard` command again to create another proxy to access the dashboard.

Using kubectl to Create a Deployment

Objectives

- Learn about application Deployments.
- Deploy your first app on Kubernetes with kubectl.

Kubernetes Deployments

> *A Deployment is responsible for creating and updating instances of your application.*

> **Note:**
> This tutorial uses a container that requires the AMD64 architecture. If you are using minikube on a computer with a different CPU architecture, you could try using minikube with a driver that can emulate AMD64. For example, the Docker Desktop driver can do this.

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment**. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

Viewing Pods and Nodes

Objectives

- Learn about Kubernetes Pods.
- Learn about Kubernetes Nodes.
- Troubleshoot deployed applications.

Kubernetes Pods

> *A Pod is a group of one or more application containers (such as Docker) and includes shared storage (volumes), IP address and information about how to run them.*

When you created a Deployment in Module 2, Kubernetes created a **Pod** to host your application instance. A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker), and some shared resources for those containers. Those resources include:

- Shared storage, as Volumes
- Networking, as a unique cluster IP address
- Information about how to run each container, such as the container image version or specific ports to use