Introduction to Kubernetes & Container Orchestration

Deploying, Scaling, and Industry Context

Student Lab Manual

Duration: 2 Hours
Target Audience: Student Group

Contents

Lab Overview

This lab introduces **Kubernetes (K8s)**, the industry standard for container orchestration, using Minikube for hands-on local experience. Students will deploy and scale a microservice, understand the declarative architecture, and learn how Kubernetes compares to proprietary cloud services like AWS EC2 Auto Scaling and Azure App Service.

Duration: 2 Hours

Learning Objectives: Students will understand Kubernetes architecture, deploy applications declaratively, demonstrate self-healing capabilities, and articulate Kubernetes' advantages in the real world.

Part I

Part 1: Conceptual Foundation (20 Minutes)

1 Kubernetes Architecture (High-Level Overview)

Kubernetes operates on a **desired state model**. You declare what you want (e.g., "I want 3 replicas of my web app running"), and Kubernetes continuously works to make the actual state match the desired state.

1.1 Control Plane (Master Node)

The **Control Plane** is the brain of the cluster. It manages the desired state, schedules applications, and performs auto-healing.

1.1.1 API Server

The **API Server** is the only component you interact with (via `kubectl`). It is the central management hub for all cluster operations.

1.1.2 etcd

The **etcd** database is the cluster's single source of truth. It stores the entire desired state of your cluster—all Deployments, Services, Pods, and configurations.

Important

If etcd fails, the cluster loses its memory. In production, etcd is replicated across multiple Control Plane nodes for high availability. In Minikube, there is a single etcd instance.

1.2 Worker Nodes

Worker Nodes are the machines (physical servers or VMs) that actually run your applications. Each Worker Node contains:

1.2.1 Kubelet

The **Kubelet** is an agent running on every Worker Node. It communicates with the API Server to receive the desired state and ensures Pods are running and healthy.

1.2.2 Container Runtime

The **Container Runtime** (e.g., Docker or containerd) is the software that pulls container images and runs them. The Kubelet uses it to manage container lifecycles.

1.3 Pods

A **Pod** is the smallest deployable unit in Kubernetes. Although a Pod can contain multiple containers, it typically contains one application container. Pods are ephemeral—they are created and destroyed as needed.

Tip

Think of a Pod as a lightweight wrapper around your Docker container. Multiple Pods run on each Worker Node, distributed by the scheduler.

Kubernetes Cluster Structure		
Control Plane	**Worker Node 1**	**Worker Node 2**
API Server Scheduler etcd	Pod 1 Pod 2 Container Runtime	Pod 3 Pod 4 Container Runtime

2 How Autoscaling Works (HPA)

The **Horizontal Pod Autoscaler (HPA)** is a Controller that constantly monitors metrics (like CPU usage) of a Deployment.

2.1 The HPA Workflow

1. **Monitoring**: The HPA checks the average CPU utilization of all running Pods against a target you define (e.g., 50%).
2. **Calculation**: If the current average exceeds the target, the HPA calculates the ideal number of replicas needed:

$$\text{desired\_replicas} = \left\lceil \frac{\text{current\_utilization}}{\text{target\_utilization}} \times \text{current\_replicas} \right\rceil$$

3. **Action**: The HPA updates the `replicas` field in the Deployment object.
4. **Self-Healing**: The Deployment Controller sees the updated replica count and automatically creates new Pods to match the desired state.

Important

The HPA doesn't directly create Pods. Instead, it updates the Deployment's desired replica count, and the Deployment Controller handles actual Pod creation. This is the power of Kubernetes' declarative model.

3 Kubernetes in the Real World

In this section, we compare Kubernetes with proprietary cloud services to understand how Kubernetes fits into the industry landscape.

3.1 Comparison Table

Feature	AWS EC2 Auto Scaling	Azure App Service	Kubernetes (EKS, AKS, GKE)
Abstraction Level	Infrastructure (VMs). Scales the underlying virtual machines.	Platform (Web Apps). Fully managed web apps/APIs.	Container (Pods). Scales containerized workload within VMs.
Scaling Unit	EC2 Virtual Machines. Scaling takes minutes.	App Service Plan Instances. Scaling is relatively fast.	Pods. Scaling takes seconds. Extremely fast.
Vendor Lock-in	High. Tied deeply to AWS APIs.	High. Tied deeply to Azure APIs.	Low (Portable). YAML files work across AWS (EKS), Azure (AKS), GCP (GKE), and on-premise clusters.
Control & Flexibility	High control over OS/VM layer.	Low control; highly opinionated.	High control over deployment, networking, and storage.
Primary Use Case	Scaling monolithic applications where VM is the boundary.	Simple web applications and APIs with zero infrastructure overhead.	Complex Microservices architectures requiring cross-cloud portability and advanced deployment strategies.

Table 1: Kubernetes vs. Cloud Scaling Services

3.2 Key Insights

3.2.1 Pod vs. VM Scaling

AWS EC2 Auto Scaling scales at the VM level—launching new EC2 instances takes minutes and is expensive. **Kubernetes** scales at the Pod level (which runs inside existing VMs)—spinning up a new Pod takes seconds and is much cheaper.

3.2.2 Portability

Azure App Service and **AWS EC2 Auto Scaling** are proprietary solutions tightly coupled to their cloud providers. Migration requires rewriting infrastructure.

Kubernetes is open-source. Your YAML files work identically across AWS (EKS), Azure (AKS), Google Cloud (GKE), and on-premise clusters. This avoids vendor lock-in.

3.2.3 Complexity vs. Control

- **Azure App Service**: Simplest—upload your code, Azure manages everything. Limited control.
- **AWS EC2 Auto Scaling**: More control but operates at VM level (coarse-grained).
- **Kubernetes**: Fine-grained control over deployment, networking, and storage. Steeper learning curve.

3.3 When to Use Kubernetes

Kubernetes is the right choice when you need:

- **Microservices Architecture**: Multiple independent services scaling at different rates.
- **Cross-Cloud Portability**: Avoid vendor lock-in and maintain deployment flexibility.
- **Fine-Grained Control**: Need control over networking, storage, and deployment strategies.
- **Fast Scaling**: Need to scale in seconds without waiting for VM provisioning.

Part II

Part 2: Hands-On Deployment (100 Minutes)

4 Step 1: Local Kubernetes Environment Setup (15 Minutes)

4.1 1.1: Install Minikube and kubectl

4.1.1 Prerequisites

- Docker Desktop (with Docker daemon running)
- Homebrew (macOS) or appropriate package manager

Important

If you don't have Docker Desktop, download it from `https://www.docker.com/products/docker-desktop`.

4.1.2 macOS Installation

```
# Install Minikube using Homebrew
brew install minikube

# Install kubectl
brew install kubectl

# Verify installations
minikube version
kubectl version --client
```

Listing 1: Install Minikube and kubectl on macOS

4.1.3 Linux Installation

```
# Download Minikube
curl -Lo minikube
    https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64
chmod +x minikube
sudo mv minikube /usr/local/bin/

# Install kubectl
sudo apt-get update
sudo apt-get install -y kubectl

# Verify installations
```

```
minikube version
kubectl version --client
```

Listing 2: Install Minikube and kubectl on Linux

4.1.4 Windows Installation

```
# Using Chocolatey
choco install minikube
choco install kubernetes-cli

# Or download from GitHub:
# https://github.com/kubernetes/minikube/releases
# https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/

minikube version
kubectl version --client
```

Listing 3: Install Minikube and kubectl on Windows

4.2 1.2: Start the Local Kubernetes Cluster

```
# Start the Minikube cluster
minikube start

# This creates a local Kubernetes cluster with:
# - Control Plane components
# - One Worker Node
# - Networking and storage configured
# - kubectl configured to point to the cluster
```

Listing 4: Start Minikube cluster

Tip

First startup takes a few minutes. Subsequent starts are faster.

4.3 1.3: Connect to Minikube's Docker Daemon

Minikube includes its own Docker daemon. To use it:

```
# Evaluate environment variables to point to Minikube's Docker
eval $(minikube docker-env)

# Verify you're using Minikube's Docker
docker info | grep "Operating System"
# Output should show: minikube
```

Listing 5: Connect to Minikube Docker

Important

This step is critical! If you skip it, Kubernetes will try to pull your image from Docker Hub and fail.

4.4 1.4: Build the Container Image

Assume you have a Spring Boot microservice (from previous labs) with a Dockerfile.

```
# Make sure you're connected to Minikube's Docker
eval $(minikube docker-env)

# Build the image
docker build -t microservice:v1 .

# Verify the image was created
docker images | grep microservice
```

Listing 6: Build Docker image

4.5 1.5: Verify Cluster Status

```
# Check nodes in the cluster
kubectl get nodes

# Expected output:
# NAME STATUS ROLES AGE VERSION
# minikube Ready control-plane 2m v1.xx.x

# Get detailed node information
kubectl describe node minikube
```

Listing 7: Verify cluster status

Tip

If status shows `NotReady`, wait for the cluster to fully initialize.

5 Step 2: Deployment and Service (25 Minutes)

5.1 2.1: Create Deployment YAML

A **Deployment** describes the desired state of your application: container image, number of replicas, port mappings, and resource limits.

Create `deployment.yaml`:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: microservice-deployment
  labels:
    app: microservice
spec:
  # Initial number of replicas
  replicas: 1

  # Selector identifies Pods managed by this Deployment
  selector:
    matchLabels:
      app: microservice

  # Template for creating Pods
  template:
    metadata:
      labels:
        app: microservice
    spec:
      containers:
      - name: microservice
        image: microservice:v1
        imagePullPolicy: Never # Don't pull from Docker Hub
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "500m"
```

Listing 8: deployment.yaml - Kubernetes Deployment

Key Configuration:

imagePullPolicy: Never Use only locally-built images. Don't try to pull from Docker Hub.

replicas: 1 Start with 1 Pod. We'll scale this later.

resources.requests Minimum CPU/memory the Pod needs.

resources.limits Maximum CPU/memory the Pod can use.

5.2 2.2: Create Service YAML

A **Service** exposes your Deployment to the network, providing stable IP addresses and load balancing.

Create `service.yaml`:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: microservice-service
  labels:
    app: microservice
spec:
  type: NodePort

  # Selector matches Pods with this label
  selector:
    app: microservice

  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
    nodePort: 30080
    name: http
```

Listing 9: service.yaml - Kubernetes Service

Service Types:

ClusterIP Internal only (default).

NodePort Exposes on each Node's IP address.

LoadBalancer Provisions external load balancer (cloud only).

For Minikube, we use `NodePort`.

5.3 2.3: Apply Configuration

```bash
# Apply the Deployment
kubectl apply -f deployment.yaml

# Apply the Service
kubectl apply -f service.yaml

# Verify Deployment was created
kubectl get deployments

# Verify Service was created
```

```
kubectl get services
```

Listing 10: Deploy to cluster

5.4 2.4: Verify Pod Creation

```
# List all Pods
kubectl get pods

# Get detailed Pod info
kubectl describe pod <pod-name>

# View Pod logs
kubectl logs <pod-name>
```

Listing 11: Check Pods

5.5 2.5: Access the Service

```
# Get the external URL
minikube service microservice-service --url

# Test the endpoint
curl <returned-url>/

# Or open in browser
minikube service microservice-service
```

Listing 12: Access the service

Tip

If you see “Connection refused”, wait a few seconds for the Pod to initialize.

6 Step 3: Declarative Management and Self-Healing (20 Minutes)

6.1 3.1: Manual Scaling (Scale Up)

Instead of manually creating Pods, declare the desired number:

```
# Scale the Deployment
kubectl scale deployment microservice-deployment --replicas=3

# Verify 3 Pods are running
kubectl get pods

# Expected output (after a few seconds):
# NAME READY STATUS AGE
# microservice-deployment-abc123def456 1/1 Running 3m
# microservice-deployment-ghi789jkl012 1/1 Running 10s
# microservice-deployment-mno345pqr678 1/1 Running 10s
```

Listing 13: Scale to 3 replicas

Important

Kubernetes creates new Pods according to the desired state. The Service automatically includes these new Pods in load balancing.

6.2 3.2: Demonstrating Self-Healing

Kubernetes' most powerful feature: automatic Pod replacement if one fails.

```
# Get Pod name
kubectl get pods

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

# Check Pods immediately
kubectl get pods

# Within seconds:
# - Deleted Pod disappears
# - NEW Pod is created with different name
# - Total count returns to 3

# This is self-healing!
```

Listing 14: Pod self-healing

Tip

The Pod name changes (Kubernetes generates random suffixes), but the label (`app: microservice`) remains the same.

6.3 3.3: Scaling Down

```
# Reduce to 1 replica
kubectl scale deployment microservice-deployment --replicas=1

# Observe Pods
kubectl get pods

# 2 Pods are terminated, 1 remains
```

Listing 15: Scale down

7 Step 4: Horizontal Pod Autoscaler (HPA) (40 Minutes)

7.1 4.1: Enable Metrics Server

The HPA uses metrics from the Metrics Server:

```
# Enable the addon
minikube addons enable metrics-server

# Verify it's running
kubectl get deployment metrics-server -n kube-system

# Wait for it to be ready (30+ seconds)
kubectl get deployment metrics-server -n kube-system --watch
# Press Ctrl+C to stop watching
```

Listing 16: Enable Metrics Server

Important

The Metrics Server collects CPU/memory usage from Pods. Without it, the HPA won't work.

7.2 4.2: Create the HPA

```
# Create HPA for automatic scaling
kubectl autoscale deployment microservice-deployment \
  --cpu-percent=50 \
  --min=1 \
  --max=5

# This means:
# - Scale when CPU > 50% of requested CPU
# - Minimum 1 Pod
# - Maximum 5 Pods
```

Listing 17: Create HPA with kubectl

7.3 4.3: Monitor HPA Status

```
# Check HPA status
kubectl get hpa

# Expected output (after metrics are available):
# NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS
# microservice-deployment Deployment/microservice-deployment 5%/50% 1 5 1

# Watch in real-time
kubectl get hpa --watch
```

```
# Press Ctrl+C to stop
```

Listing 18: Monitor HPA

Tip

Initially you may see `<unknown>/50%`. This is normal—the Metrics Server needs time to collect data. Wait 1-2 minutes.

7.4 4.4: Generate Load

Simulate CPU-intensive traffic to trigger scaling:

```
# Terminal 1: Get service URL
SERVICE_URL=$(minikube service microservice-service --url)

# Terminal 2: Generate load with curl loop
while true; do
  curl -s $SERVICE_URL/ > /dev/null
done

# Terminal 3: Watch HPA scale up
kubectl get hpa --watch

# You should see:
# REPLICAS increasing: 1 -> 2 -> 3 -> 4 -> 5
# TARGETS showing high CPU usage
```

Listing 19: Generate load with curl loop

7.5 4.5: Observe Scaling Down

```
# In Terminal 2, press Ctrl+C to stop load generation

# In Terminal 3, continue watching
kubectl get hpa --watch

# After 5 minutes (stabilization period):
# REPLICAS decreases back to 1
# TARGETS shows low CPU usage
```

Listing 20: Stop load and watch scale-down

Tip

Kubernetes includes a 5-minute stabilization period to prevent rapid scaling oscillations—this is intentional behavior.

8 Step 5: Clean Up

After the lab, free up resources:

```
# Delete the Service
kubectl delete service microservice-service

# Delete the Deployment (also deletes Pods)
kubectl delete deployment microservice-deployment

# Delete the HPA
kubectl delete hpa microservice-deployment

# Verify everything is deleted
kubectl get deployments
kubectl get services
kubectl get pods

# Stop Minikube (optional)
minikube stop

# Delete Minikube cluster entirely (optional)
minikube delete
```

Listing 21: Clean up resources

Part III

Part 3: Lab Demonstration and Evaluation

9 What Students Must Demonstrate

Component	Evidence Required	Focus Area
K8s Core Objects	Show running Pods with `kubectl get pods`. Access service URL successfully.	Deployment and networking configuration.
Self-Healing	Delete a Pod with `kubectl delete pod <pod-name>`. Show new Pod automatically created. Replica count unchanged.	Declarative state and Deployment Controllers.
Horizontal Autoscaling	Show HPA creation. Display `kubectl get hpa` output. Demonstrate replica count increasing under simulated load.	Orchestration power and automated scaling.
Conceptual Understanding	Verbally explain: How HPA differs from AWS EC2 Auto Scaling (Container vs. VM scaling). Why Kubernetes is portable.	Real-world context and industry relevance.

10 Sample Interview Questions

1. **Q: Explain the Kubernetes desired state model.**

   A: You declare what you want (desired state) in YAML files. Kubernetes continuously monitors the actual state and makes changes to match the desired state. If a Pod fails, Kubernetes replaces it automatically.

2. **Q: What is the difference between the Control Plane and Worker Nodes?**

   A: The Control Plane manages the cluster (scheduling, desired state, health). Worker Nodes run the Pods. The Control Plane decides where Pods go; Worker Nodes execute those decisions.

3. **Q: Why is Kubernetes faster to scale than AWS EC2?**

   A: Kubernetes scales Pods in seconds (they're lightweight containers already inside VMs). EC2 takes minutes because it must provision entire new virtual machines, including OS boot time.

4. **Q: What does "imagePullPolicy: Never" do and why did we use it?**

   A: It tells Kubernetes to only use locally-built images, not pull from Docker Hub. We used it because our image doesn't exist in Docker Hub—it's only in Minikube's local Docker daemon.

5. **Q: How is Kubernetes more portable than AWS services?**

   A: Kubernetes is cloud-agnostic. The same YAML files work on AWS (EKS), Azure (AKS), Google Cloud (GKE), and on-premise. AWS services are proprietary and tied to their APIs.

6. **Q: What does the HPA do when CPU usage drops below 50%?**

   A: After a 5-minute stabilization period, the HPA scales down by reducing replicas. This prevents thrashing (rapid scale up/down) when load fluctuates.

11 Expected Outcomes Checklist

By the end of this lab, students should be able to:

- ☐ Set up and interact with a local Kubernetes cluster using Minikube and kubectl
- ☐ Deploy an application using Kubernetes Deployment and expose it using a Service
- ☐ Demonstrate the declarative, self-healing, and scaling principles of Kubernetes
- ☐ Articulate the high-level K8s architecture (Control Plane vs. Worker Node)
- ☐ Compare and contrast Kubernetes with proprietary cloud scaling solutions
- ☐ Explain why Kubernetes is portable and avoids vendor lock-in
- ☐ Manually scale a Deployment and observe automatic Pod creation
- ☐ Demonstrate Kubernetes self-healing by deleting a Pod and observing replacement
- ☐ Create and monitor a Horizontal Pod Autoscaler
- ☐ Explain the difference between Pod-level scaling and VM-level scaling

A Troubleshooting Guide

A.1 Minikube Issues

A.1.1 Error: Minikube fails to start

Warning

Solution:

1. Ensure Docker Desktop is running
2. Check available disk space: `df -h`
3. Delete old cluster: `minikube delete`
4. Try again: `minikube start`

A.1.2 Error: kubectl cannot connect

Warning

Solution:

1. Verify Minikube is running: `minikube status`
2. Reset kubectl: `kubectl config use-context minikube`
3. Check kubectl is installed: `kubectl version --client`

A.2 Docker Issues

A.2.1 Error: Docker image not found when deploying

Warning

Solution:

1. Verify you ran: `eval $(minikube docker-env)`
2. Verify image exists: `docker images | grep microservice`
3. Rebuild image: `docker build -t microservice:v1 .`
4. Delete failed Pod: `kubectl delete pod <pod-name>`
5. Kubernetes creates new Pod with correct image

A.3 Pod Issues

A.3.1 Error: Pod is in CrashLoopBackOff

Warning

Solution:

1. Check logs: `kubectl logs <pod-name>`
2. Check resources: `kubectl describe pod <pod-name>`
3. Resource limits may be too low—increase in deployment.yaml
4. Redeploy: `kubectl apply -f deployment.yaml`

A.3.2 Error: Service URL returns connection refused

Warning

Solution:

1. Wait for Pod to initialize (30 seconds)
2. Check Pod status: `kubectl get pods`
3. Verify Service points to correct port: `kubectl get service`
4. Check logs: `kubectl logs <pod-name>`

A.4 HPA Issues

A.4.1 Error: HPA shows `<unknown>/50%`

Warning

Solution:

1. Metrics Server is still collecting data—wait 1-2 minutes
2. Verify running: `kubectl get deployment metrics-server -n kube-system`
3. Check metrics: `kubectl top pods`
4. Restart if needed: disable and re-enable addon

A.4.2 Error: Pods not scaling up under load

Warning

Solution:

1. Verify HPA exists: `kubectl get hpa`
2. Verify Deployment has resource requests (HPA uses these)
3. Check HPA status: `kubectl describe hpa <hpa-name>`
4. Generate more aggressive load
5. Check events: `kubectl describe hpa <hpa-name>`

B Quick Reference: kubectl Commands

Command	Description
`kubectl get nodes`	List cluster nodes
`kubectl get pods`	List Pods
`kubectl get deployments`	List Deployments
`kubectl get services`	List Services
`kubectl get hpa`	List Horizontal Pod Autoscalers
`kubectl describe pod <name>`	Detailed Pod information
`kubectl logs <pod-name>`	View Pod logs
`kubectl delete pod <name>`	Delete a Pod
`kubectl scale deployment <name> --replicas=N`	Scale Deployment
`kubectl apply -f <file.yaml>`	Apply YAML configuration
`kubectl edit deployment <name>`	Edit Deployment

C Key Kubernetes Concepts

Desired State What you declare in YAML (e.g., 3 replicas running)

Actual State What's currently running in the cluster

Reconciliation Kubernetes continuously adjusting actual state to match desired state

Declarative You say WHAT you want, Kubernetes figures out HOW to achieve it

Imperative You say HOW to do something (traditional scripts)

Controller A Kubernetes component that watches desired state and makes corrections

Replica A copy of your Pod running independently

Scale Increase/decrease number of replicas

Self-Healing Automatic replacement of failed Pods