# Day 1 Detailed Lab Implementation Guide

---

## Prerequisites

1. **Local Machine Setup**

   - Install **kubectl** (v1.23+)

**Detailed Guide: Installing kubectl (v1.23+)**

This guide provides platform-specific instructions to install **kubectl**, the Kubernetes command-line tool, version 1.23 or higher.

---

**1. Verify Prerequisites**

- A supported operating system: macOS, Linux (Debian/Ubuntu, CentOS/RHEL, Fedora), or Windows.
- Internet access to download binaries.
- Sufficient privileges to install system packages or place binaries.

**2. Install on Linux**

**2.1 Using Package Manager (Debian/Ubuntu)**

1. Update package index and install dependencies:

```
sudo apt-get update
sudo apt-get install -y ca-certificates curl
```

2. Download v1.23 binary:

```
curl -LO "https://dl.k8s.io/release/v1.23.17/bin/linux/amd64/kubectl"
```

3. Validate checksum:

```
curl -LO "https://dl.k8s.io/v1.23.17/bin/linux/amd64/kubectl.sha256"
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
```

4. Install:

```
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

5. Confirm:

kubectl version --client --short

**3. Post-Installation Configuration**

1. Autocomplete (optional):
    - macOS/Linux (bash):

source <(kubectl completion bash)

    - macOS/Linux (zsh):

source <(kubectl completion zsh)

    - Windows (PowerShell):

kubectl completion powershell | Out-String | Invoke-Expression

2. Verify connectivity once you configure a cluster context:

kubectl cluster-info

---

**kubectl** is now installed and ready for use with Kubernetes v1.23 or newer.

- Install **Helm** (v3+)

**Detailed Guide: Installing Helm (v3+)**

This guide provides platform-specific instructions to install **Helm**, the Kubernetes package manager, version 3 or higher.

---

**1. Verify Prerequisites**

- A supported operating system: macOS, Linux (Debian/Ubuntu, CentOS/RHEL, Fedora), or Windows.
- Internet access to download binaries.
- Sufficient privileges to install system packages or place binaries.

**2. Install on Linux**

**2.1 Using Package Manager (Debian/Ubuntu)**

1. Update package index and install dependencies:

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl

2. Add Helm GPG key and repository:

curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -

sudo apt-get install -y software-properties-common

sudo add-apt-repository "deb https://baltocdn.com/helm/stable/debian/ all main"

3. Install Helm:

sudo apt-get update

sudo apt-get install -y helm

4. Verify version:

helm version --short

**3. Post-Installation Configuration**

1. **Add Stable Repository** (optional but recommended):

helm repo add stable https://charts.helm.sh/stable

helm repo update

2. **Enable Autocompletion**:
    - **bash**:

source <(helm completion bash)

    - **zsh**:

source <(helm completion zsh)

3. **Verify Connectivity**:
   Deploy a test chart (e.g., nginx) to confirm Helm can install releases:

helm install test-nginx stable/nginx-ingress --namespace default

helm list

Helm is now installed and ready for managing Kubernetes packages with version 3 or higher.

- Install **Docker** and login to a public registry (optional)

**Detailed Guide: Installing Docker & Logging In to a Public Registry**

This guide provides **step-by-step instructions** to install Docker on macOS, Linux, and Windows, and authenticate with Docker Hub (or any public registry).

---

**1. Verify Prerequisites**

- Supported OS: macOS, Linux (Debian/Ubuntu, CentOS/RHEL, Fedora), Windows 10/11.
- Internet connectivity.
- Sufficient privileges to install applications or run elevated commands.

**2. Install Docker on Linux**

**2.1 Debian/Ubuntu**

1. Update package index and install dependencies:

```
sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release
```

2. Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

3. Add Docker repository:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Install Docker Engine:

```
sudo apt-get update

sudo apt-get install -y docker-ce docker-ce-cli containerd.io
```

5. Add your user to the Docker group (optional, for non-root usage):

```
sudo usermod -aG docker $USER
```

newgrp docker

6. Verify:

docker version

**3. Log In to a Public Registry (Docker Hub)**

1. Open a terminal, PowerShell, or Docker CLI.
2. Execute login command:

docker login

3. When prompted, enter your Docker Hub **username** and **password**.
4. On success, you will see:

Login Succeeded

5. (Optional) To log in to another registry, specify its URL:

docker login myregistry.example.com

**6. Push & Pull a Test Image**

1. Pull an official image:

docker pull alpine:latest

2. Tag it for your namespace:

docker tag alpine:latest <your-username>/alpine:test

3. Push to Docker Hub:

docker push <your-username>/alpine:test

4. Verify by pulling it again:

docker pull <your-username>/alpine:test

Docker and registry authentication are now configured, enabling you to build, push, and pull images seamlessly.

- Install **Zeebe CLI** (`zbctl`): `brew install zeebe/zeebe/zbctl` or download from https://github.com/camunda-cloud/zeebe/releases

- Ensure access to a Kubernetes cluster (e.g., Minikube, kind, or a managed cluster)

kubectl cluster-info

kubectl get nodes

kubectl get pods --all-namespaces

2. **Public Repositories**

  - **Camunda 8 Helm Charts**: https://github.com/camunda/camunda-platform-deployment

  - **Sample BPMN & Worker Project**: https://github.com/camunda-community-hub/camunda-8-get-started-spring

---

## Lab 1: Deploy Zeebe Cluster & Console on Kubernetes

### 1. Clone Helm Chart Repo

git clone https://github.com/camunda/camunda-platform-deployment.git

cd camunda-platform-deployment/helm

```

### 2. Add Camunda Helm Repo & Update

helm repo add camunda https://helm.camunda.io

helm repo update

```

### 3. Create Namespace

kubectl create namespace camunda

```
```

### 4. Deploy Zeebe Cluster

```
helm install zeebe camunda/zeebe \
  --namespace camunda \
  --set broker.replicas=3 \
  --set gateway.replicas=2 \
  --set resources.broker.requests.cpu="500m" \
  --set resources.broker.requests.memory="1Gi" \
  --set resources.broker.limits.cpu="1" \
  --set resources.broker.limits.memory="2Gi"
```

### 5. Deploy Console (Operate, Tasklist, Optimize)

```
helm install console camunda/console \
  --namespace camunda \
  --set operate.enabled=true \
  --set tasklist.enabled=true \
  --set optimize.enabled=true
```

### 6. Verify Deployment

```
kubectl get pods -n camunda
```

All Zeebe broker and console pods should be in **Running** state.

---

## Lab 2: Model & Deploy a Sample BPMN Process

### 1. Clone Sample Project

git clone https://github.com/camunda-community-hub/camunda-8-get-started-spring.git

cd camunda-8-get-started-spring

```

### 2. Open BPMN in Web Modeler

1. Navigate to `model/` folder

2. Open `order-process.bpmn` in Camunda Web Modeler (https://camunda.com/web-modeler)

3. Review the **Order Process** diagram: start event, service task, user task, end event

### 3. Export BPMN File

- In Web Modeler, click **Save** → **Download File** → save as `order-process.bpmn` in project root.

### 4. Deploy Process via Zeebe CLI

zbctl deploy order-process.bpmn

```

### 5. Instantiate Process

zbctl create instance OrderProcess

```

### 6. Access Tasklist

- Port-forward Tasklist: `kubectl port-forward svc/console-tasklist 8081:80 -n camunda`

- Open http://localhost:8081 and complete the **Accept Order** user task.
```

---

## Lab 3: External Elasticsearch Configuration

### 1. Deploy Elasticsearch via Helm

```
helm repo add elastic https://helm.elastic.co
helm repo update
helm install elasticsearch elastic/elasticsearch \
  --namespace camunda \
  --set replicas=3 \
  --set resources.requests.cpu="500m" \
  --set resources.requests.memory="1Gi" \
  --set resources.limits.cpu="1" \
  --set resources.limits.memory="2Gi"
```

### 2. Update Console Values

Create `external-es-values.yaml`:

```
operate:
  elasticsearch:
    addresses:
      - elasticsearch-master.camunda.svc.cluster.local:9200
tasklist:
  elasticsearch:
    addresses:
      - elasticsearch-master.camunda.svc.cluster.local:9200
```

### 3. Upgrade Console

helm upgrade console camunda/console \

  --namespace camunda \

  -f external-es-values.yaml

```

### 4. Validate

- In Operate UI (port-forward on port 8080), ensure you see process instances and job logs.

---

## Lab 4: Integrate Keycloak for Cockpit Authentication

### 1. Deploy Keycloak

docker run -d --name keycloak -p 8081:8080 quay.io/keycloak/keycloak:latest start-dev

```

### 2. Configure Keycloak

1. Open http://localhost:8081

2. Login default user `admin`/`admin`

3. Create realm `camunda`

4. Create client `cockpit-client` with:

   - Access Type: `confidential`

   - Valid Redirect URI: `*`

5. Obtain **Client ID** and **Secret** from **Credentials** tab.

### 3. Update Helm Values

Create `keycloak-values.yaml`:

```
console:
  authentication:
    enabled: true
    clientId: "cockpit-client"
    clientSecret: "<SECRET_FROM_KEYCLOAK>"
    issuerUri: "http://keycloak.local:8081/realms/camunda"
```

### 4. Redeploy Console

```
helm upgrade console camunda/console \
  --namespace camunda \
  -f keycloak-values.yaml
```

### 5. Verify

- Access Cockpit (port-forward on port 8082) and login via Keycloak.

---

## Lab 5: Secrets Management with HashiCorp Vault

### 1. Deploy Vault

```
helm repo add hashicorp https://helm.releases.hashicorp.com
helm repo update
helm install vault hashicorp/vault \
  --namespace camunda \
  --set server.dev.enabled=true
```

### 2. Initialize & Unseal (Dev Mode skips unseal)

kubectl exec -n camunda vault-0 -- vault status

```

### 3. Store Broker Password in Vault

export VAULT_ADDR="http://$(kubectl get svc vault -n camunda -o jsonpath='{.status.loadBalancer.ingress[0].ip}'):8200"

vault kv put secret/camunda brokerPassword=mySecretPass

```

### 4. Deploy External Secrets Operator

kubectl apply -f https://raw.githubusercontent.com/external-secrets/kubernetes-external-secrets/main/deploy/crds.yaml

kubectl apply -f https://raw.githubusercontent.com/external-secrets/kubernetes-external-secrets/main/deploy/operator.yaml

```

### 5. Create ExternalSecret

Save `external-secret.yaml`:

apiVersion: external-secrets.io/v1beta1

kind: ExternalSecret

metadata:

  name: zeebe-broker-secret

  namespace: camunda

spec:

  refreshInterval: "1h"

  secretStoreRef:

    name: vault
```

```
    kind: ClusterSecretStore
  target:
    name: zeebe-broker-secret
    creationPolicy: Merge
  data:
    - secretKey: brokerPassword
      remoteRef:
        key: secret/camunda
        property: brokerPassword
```

Apply:

```
kubectl apply -f external-secret.yaml
```

### 6. Update Zeebe Deployment to Use Secret

Set in Helm override `vault-secret-values.yaml`:

```
zeebe:
  broker:
    env:
      - name: ZEEBE_BROKER_SECURITY_SSL_KEY_PASSWORD
        valueFrom:
          secretKeyRef:
            name: zeebe-broker-secret
            key: brokerPassword
```

Upgrade:

```
helm upgrade zeebe camunda/zeebe -n camunda -f vault-secret-values.yaml
```

```

### 7. Validate

- Ensure Zeebe brokers start without errors and connect securely.
```