Kind Cluster Scripts

1. Install prerequisites

sudo apt update && sudo apt -y install docker.io curl git gnupg ca-certificates

sudo usermod -aG docker "$USER"

newgrp docker # refresh group membership in this shell

1.1 kubectl

curl -LO "https://dl.k8s.io/release/v1.29.6/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

kubectl version --client

1.2 kind

curl -Lo kind https://kind.sigs.k8s.io/dl/v0.23.0/kind-linux-amd64

chmod +x kind

sudo mv kind /usr/local/bin/

kind version

1.3 Helm

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

helm version

1.4 Istio CLI (istioctl)

curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.22.1 sh -

export PATH="$PWD/istio-1.22.1/bin:$PATH"

istioctl version

1.5 k6 (choose one)

Option A – native:

sudo apt update

sudo apt install -y gnupg ca-certificates

curl -s https://dl.k6.io/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/k6-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable main" \

| sudo tee /etc/apt/sources.list.d/k6.list

sudo apt update && sudo apt install -y k6

k6 version

Option B – Docker alias:

alias k6='docker run --rm -it --network host -v "$PWD":/work -w /work -e BASE_URL grafana/k6:latest'

k6 version

---

2. Create the kind cluster (1 CP + 2 workers)

Create a config file:

cat > kind-3node.yaml <<'EOF'

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

- role: control-plane

image: kindest/node:v1.29.2

extraPortMappings:

- containerPort: 32080 # Istio ingress HTTP → host:80

hostPort: 80

- containerPort: 32443 # Istio ingress HTTPS → host:443

```
    hostPort: 443
- role: worker
  image: kindest/node:v1.29.2
- role: worker
  image: kindest/node:v1.29.2
EOF
```

Create the cluster:

```
kind create cluster --name demo --config kind-3node.yaml
kubectl get nodes -o wide
```

Taint the control-plane so it doesn't run workloads:

```
CP_NODE=$(kubectl get nodes -l 'node-role.kubernetes.io/control-plane' -o jsonpath='{.items[0].metadata.name}')
kubectl taint nodes "$CP_NODE" node-role.kubernetes.io/control-plane=:NoSchedule --overwrite
```

Label worker nodes:

```
for n in $(kubectl get nodes --no-headers | awk '$3!="control-plane"{print $1}'); do
  kubectl label node "$n" nodepool=worker --overwrite
done
kubectl get nodes --show-labels
```

---

3. Install Istio (demo profile)

```
istioctl install --set profile=demo -y
kubectl -n istio-system get pods
```

Patch the ingress gateway Service to use the NodePorts matching the kind mappings:

```
kubectl -n istio-system patch svc istio-ingressgateway \
  -p '{"spec":{"type":"NodePort","ports":[
    {"name":"http2","port":80,"targetPort":8080,"nodePort":32080},
```

```
  {"name":"https","port":443,"targetPort":8443,"nodePort":32443}
 ]}}'
```

At this point, http://localhost on the host should reach the Istio ingress gateway.

---

4. Deploy demo applications (Sock Shop, TeaStore, Bookinfo)

All workloads are scheduled only on worker nodes via nodeSelector: nodepool=worker.

kind_Cluster

4.1 Sock Shop

```
kubectl create ns sock-shop
kubectl label ns sock-shop istio-injection=enabled

git clone https://github.com/microservices-demo/microservices-demo.git
kubectl apply -n sock-shop -f microservices-demo/deploy/kubernetes/manifests

# Force all Deployments onto worker nodes
kubectl -n sock-shop get deploy -o name | \
  xargs -I{} kubectl -n sock-shop patch {} --type=merge \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"nodepool":"worker"}}}}}'
```

4.2 TeaStore

```
kubectl create ns teastore
kubectl label ns teastore istio-injection=enabled

git clone https://github.com/DescartesResearch/TeaStore.git
kubectl apply -n teastore -f TeaStore/kubernetes/teastore-namespace.yaml
kubectl apply -n teastore -f TeaStore/kubernetes/teastore-deployment-service.yaml
```

```
kubectl -n teastore get deploy -o name | \
  xargs -I{} kubectl -n teastore patch {} --type=merge \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"nodepool":"worker"}}}}}'
```

4.3 Bookinfo

```
kubectl create ns bookinfo
kubectl label ns bookinfo istio-injection=enabled

kubectl apply -n bookinfo -f istio-1.22.1/samples/bookinfo/platform/kube/bookinfo.yaml

kubectl -n bookinfo get deploy -o name | \
  xargs -I{} kubectl -n bookinfo patch {} --type=merge \
  -p '{"spec":{"template":{"spec":{"nodeSelector":{"nodepool":"worker"}}}}}'
```

Quick sanity check: no app pod should be scheduled on the control-plane node.

```
kubectl get pods -A -o wide | awk 'NR==1 || $8 !~ /control-plane/'
```

---

5. Configure Istio routing (single Gateway + VirtualService)

Create a routing config that exposes:

- http://localhost/sock → Sock Shop front-end
- http://localhost/tea → TeaStore web UI
- http://localhost/book → Bookinfo productpage

```
cat > istio-routes.yaml <<'EOF'
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: apps-gateway
  namespace: istio-system
```

```
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts: ["*"]


---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: route-all
  namespace: istio-system
spec:
  hosts: ["*"]
  gateways: ["istio-system/apps-gateway"]
  http:
  - match: [{ uri: { prefix: "/sock" } }]
    rewrite: { uri: "/" }
    route:
    - destination:
        host: front-end.sock-shop.svc.cluster.local
        port: { number: 80 }
```

```
  - match: [{ uri: { prefix: "/tea" } }]
    rewrite: { uri: "/" }
    route:
    - destination:
        host: teastore-webui.teastore.svc.cluster.local
        port: { number: 8080 }

  - match: [{ uri: { prefix: "/book" } }]
    rewrite: { uri: "/" }
    route:
    - destination:
        host: productpage.bookinfo.svc.cluster.local
        port: { number: 9080 }
EOF

kubectl apply -f istio-routes.yaml
```

---

6. Smoke test the endpoints

From the host:

curl -I http://localhost/sock

curl -I http://localhost/tea

curl -I http://localhost/book

You should see HTTP/1.1 200 OK (or at least a redirect followed by 200 when you do a full GET in a browser).

You can also open:

- http://localhost/sock
- http://localhost/tea
- http://localhost/book

inside a browser running on the VM.

---

7. Create the k6 spike-load script

Create s1_spike.js: