How do you scale a Deployment up/down?

To **scale a Deployment up/down** in Kubernetes means to **increase or decrease the number of replicas (pods)** managed by the Deployment.

You can do this manually using the `kubectl scale` command or by modifying the Deployment YAML and applying the change.

🧪 DEMO: Example to Scale a Deployment

Let's say we have a simple NGINX Deployment.

Step 1: Create a Deployment YAML file `nginx-deployment.yaml`

```
yaml
---------
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2  # Start with 2 pods
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Step 2: Apply the Deployment

```
bash
---------
kubectl apply -f nginx-deployment.yaml
```

Check the pods:

```
bash
---------
kubectl get pods
```

Step 3: Scale the Deployment Up (e.g., from 2 to 5 pods)

```
bash
---------
kubectl scale deployment nginx-deployment --replicas=5
```

Verify:

```
bash
---------
kubectl get deployment nginx-deployment
```

You'll see:

```
bash
---------
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    5/5     5            5           2m
```

Step 4: Scale the Deployment Down (e.g., from 5 to 1 pod)

```
bash
---------
kubectl scale deployment nginx-deployment --replicas=1
```

Check again:

```
bash
---------
kubectl get pods
```

Now only 1 pod will be running.

✅==> You can also update the YAML and re-apply

Edit YAML:

```
yaml
---------
spec:
  replicas: 3
```

Then:

```
bash
---------
kubectl apply -f nginx-deployment.yaml
```

Let’s now see how to **automatically scale** your Deployment based on CPU usage using **Horizontal Pod Autoscaler (HPA)**.

⚙️ Auto-scaling with HPA

✅ Prerequisites

Metrics Server Setup for Kind

📌 Step 1: Check if Metrics Server is already installed

```
bash
---------
kubectl get deployment metrics-server -n kube-system
```

If you get:

```
pgsql
---------
Error from server (NotFound): deployments.apps "metrics-server" not found
```

That means it's not installed.

📥 Step 2: Install Metrics Server (Kind-compatible)

Run the following to install Metrics Server with the right flags for Kind:

```
bash
---------
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Then **patch the deployment** to allow insecure TLS (necessary for local clusters like Kind):

```
bash
---------
kubectl patch deployment metrics-server -n kube-system \
  --type=json \
  -p='[{"op":"add","path":"/spec/template/spec/containers/0/args/-","value":"--kubelet-insecure-tls"}]'
```

🕒 Step 3: Wait and verify it's running

```
bash
```

```
---------
kubectl get pods -n kube-system | grep metrics-server
```

You should see something like:

```
sql
---------
metrics-server-xxxxxxx-yyyy   1/1     Running   0          30s
```

✅ Step 4: Verify metrics are available

Run:

```
bash
---------
kubectl top nodes
```

And:

```
bash
---------
kubectl top pods
```

If it shows CPU and memory usage — you're good to go! Now HPA will work perfectly on Kind.

-

📦 Step 1: Update Deployment to request CPU resources

Modify your `nginx-deployment.yaml` to add `resources`:

```
yaml
---------
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    resources:
      requests:
        cpu: "100m"
      limits:
        cpu: "200m"
```

Re-apply:

```
bash
---------
kubectl apply -f nginx-deployment.yaml
```

Let’s say:

- You have 1 pod.

- It’s using 80m CPU (80% of 100m request).
- Since **80% > 50% target**, HPA will add a pod.
- Now 2 pods handle the load.
- If load still stays high, it may go to 3, 4... until max 10.

If load goes down, HPA will **scale down** again to save resources.

📈 Step 2: Create Horizontal Pod Autoscaler (HPA)

Use this command to auto-scale between 1 and 10 pods based on 50% CPU usage:

```
bash
---------
kubectl autoscale deployment nginx-deployment --cpu-percent=50 --min=1 --max=10
```

Check the HPA status:

```
bash
---------
kubectl get hpa
```

Output (example):

```
bash
---------
NAME                REFERENCE                     TARGETS         MINPODS
MAXPODS   REPLICAS   AGE
nginx-deployment   Deployment/nginx-deployment   5% / 50%        1          10
1          10s
```

🧪 Step 3: Simulate CPU Load

To test auto-scaling, you'll need to generate CPU load on the pods.

🧪 Simulate CPU Load to Trigger HPA

We’ll use a container called `vish/stress`, which just eats CPU so we can see autoscaling in action.

✅ Step 1: Create a Deployment that stresses CPU

Create a new file `stress-deployment.yaml`:

```
yaml
---------
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stress-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: stress
  template:
    metadata:
      labels:
        app: stress
    spec:
      containers:
      - name: stress
        image: vish/stress
        resources:
          requests:
            cpu: "100m"
          limits:
            cpu: "200m"
        args:
          - -cpus
          - "1"
```

🚀 Step 2: Apply the Deployment

```
bash
---------
kubectl apply -f stress-deployment.yaml
```

📈 Step 3: Create HPA for stress deployment

```
bash
---------
kubectl autoscale deployment stress-deployment --cpu-percent=50 --min=1 --max=5
```

🔍 Step 4: Watch the HPA scale

```
bash
---------
watch kubectl get hpa
```

You'll see the CPU % rise and new pods getting added automatically.

You can also monitor:

```
bash
---------
watch kubectl get pods -l app=stress
```

Within 1–2 minutes, you'll see:

- CPU usage going over 50%
- Pods scaling from 1 → 2 → 3… until CPU usage drops below threshold

---

✅ Cleanup (after testing)

```
bash
----------
kubectl delete deployment stress-deployment
kubectl delete hpa stress-deployment
```

Q1. Which command is used to manually scale a Deployment in Kubernetes?

A) `kubectl resize deployment`
B) `kubectl scale deployment`
C) `kubectl edit deployment`
D) `kubectl expand pods`

✅ **Answer:** B) `kubectl scale deployment`

Which of the following is NOT a valid reason for HPA to increase the number of pods?

A) High CPU usage
B) High memory usage (with configuration)
C) Long-running pods
D) Custom metrics (with setup)

✅ **Answer:** C) Long-running pods