**📝 Kubernetes Hands-on Lab Guide: Exploring and Managing Pods**

**🎯 Lab Objective**

This lab is designed to give you hands-on experience with:

* Exploring all pods in a Kubernetes cluster across namespaces.
* Creating and working within a new namespace.
* Deploying a simple pod using kubectl run.
* Describing and deleting pods.
* Using kubectl output filters (like grep) for quick checks.
* Monitoring pod placement on cluster nodes.

At the end of this lab, you will understand how to:  
✅ View Kubernetes resources at cluster and namespace level.  
✅ Create, inspect, and delete pods.  
✅ Understand pod scheduling across nodes.

**🛠 Lab Prerequisites**

* Access to a Kubernetes cluster (Minikube, KIND, or cloud-hosted).
* kubectl CLI installed and configured to communicate with the cluster.
* Basic Linux shell knowledge.

**🔥 Lab Steps**

**🔹 Step 1: Explore existing pods**

Run the following to list **all pods in all namespaces**:

bash

CopyEdit

kubectl get pods -A

✅ **What happens?**

* Displays all pods running in every namespace (-A is shorthand for --all-namespaces).
* You will see system pods like kube-dns, coredns, kube-proxy, etc.

**🔹 Step 2: Get detailed pod information**

To include **node placement** and other details, use the -o wide flag:

bash

CopyEdit

kubectl get pods -A -o wide

✅ **Expected output:**

* Shows additional columns like NODE and IP to indicate on which node each pod is running.

**🔹 Step 3: Alias for kubectl**

To save typing effort, create a shell alias:

bash

CopyEdit

alias k=kubectl

✅ From now on, you can use k instead of kubectl.  
Try:

bash

CopyEdit

k get pods -A

**🔹 Step 4: Create a new namespace**

Namespaces logically isolate workloads in Kubernetes. Create a namespace called raman:

bash

CopyEdit

k create ns raman

✅ Verify the namespace:

bash

CopyEdit

k get ns

**🔹 Step 5: Check pod distribution on nodes**

Use grep to filter pods based on node names:

bash

CopyEdit

k get pods -A -o wide | grep master

k get pods -A -o wide | grep w1

k get pods -A -o wide | grep w2

✅ This lets you see which pods are running on master, w1, and w2 nodes.

**🔹 Step 6: Explore API resources**

To list all Kubernetes resource types (like Pods, Services, Deployments, etc.):

bash

CopyEdit

k api-resources

✅ This shows what you can create and manage in Kubernetes.

**🔹 Step 7: Create a pod in default namespace**

Run a simple httpd pod:

bash

CopyEdit

k run ramanapp1 --image=httpd

✅ Verify pod creation:

bash

CopyEdit

k get pods

k get pods -o wide

**🔹 Step 8: Delete the pod**

Delete the pod you created:

bash

CopyEdit

k delete pod ramanapp1

**🔹 Step 9: Create a pod in the raman namespace**

Run the pod in the newly created namespace:

bash

CopyEdit

k run ramanapp1 --image=httpd -n raman

✅ Verify:

bash

CopyEdit

k get pods -n raman

**🔹 Step 10: Monitor pod status**

To watch the pod status live:

bash

CopyEdit

k get pods -n raman -o wide -w

✅ Press Ctrl+C to exit watch mode.

**🔹 Step 11: Describe the pod**

To get detailed information (events, container specs, volumes, etc.):

bash

CopyEdit

k describe pod ramanapp1 -n raman

✅ Look at:

* Events at the bottom (for scheduling details).
* Node assignment.
* Container details.

**🔹 Step 12: Clean up**

Delete the pod in raman namespace:

bash

CopyEdit

k delete pod ramanapp1 -n raman

**📝 Key Commands Summary**

| **Command** | **Description** |
| --- | --- |
| kubectl get pods -A | List all pods in all namespaces |
| kubectl get pods -o wide | Show extra details (node, IP, etc.) |
| kubectl create ns <namespace> | Create a new namespace |
| kubectl run <podname> --image=<image> | Create a pod |
| kubectl delete pod <podname> | Delete a pod |
| kubectl describe pod <podname> | Show detailed info about a pod |
| kubectl api-resources | List all API resources |
| alias k=kubectl | Create a short alias for kubectl |

**🎓 What you learned**

✅ How to explore Kubernetes resources.  
✅ Namespace isolation for workloads.  
✅ Pod scheduling and distribution across nodes.  
✅ Debugging and describing pods.

**📝 Kubernetes Hands-on Lab Guide: Multi-container Pod with Interactive Access**

**🎯 Lab Objective**

This lab teaches you how to:  
✅ Write a YAML manifest for a pod with multiple containers.  
✅ Create and manage pods using kubectl apply/create.  
✅ Inspect pod and container details.  
✅ Execute commands inside running containers.  
✅ Redeploy pods and perform cleanup.

By the end, you’ll understand **sidecar patterns**, YAML anatomy, and working interactively with containers.

**🛠 Prerequisites**

* Access to a running Kubernetes cluster.
* kubectl installed and configured.
* A namespace raman already created (or create it in this lab).

**🗂 Lab Overview**

| **Step** | **Task** |
| --- | --- |
| 1 | Create workspace directory for YAMLs |
| 2 | Author multi-container pod YAML |
| 3 | Deploy the pod in namespace raman |
| 4 | Inspect pod status and details |
| 5 | Execute commands inside containers |
| 6 | Modify YAML and redeploy |
| 7 | Clean up resources |

**🧑‍💻 Step-by-Step Guide**

**🔹 Step 1: Prepare workspace**

1. Create a directory to store YAML manifests:

bash

CopyEdit

mkdir ~/raman

cd ~/raman

1. Confirm you're in the directory:

bash

CopyEdit

pwd

✅ **Expected output:**  
/root/raman

**🔹 Step 2: Author pod YAML**

Create a file called pod.yaml:

bash

CopyEdit

vi pod.yaml

Add the following content:

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: nginxpod

namespace: raman

spec:

containers:

- name: nginxcon

image: nginx:1.14.2

ports:

- containerPort: 80

- name: rediscon

image: redis

ports:

- containerPort: 6379

✅ **What this does:**

* Creates a **Pod** with two containers:
  + nginxcon: serves HTTP requests on port 80.
  + rediscon: runs Redis server on port 6379.
* Deploys into namespace raman.

**🔹 Step 3: Deploy the pod**

1. Apply the YAML manifest:

bash

CopyEdit

kubectl create -f pod.yaml

✅ **Expected output:**  
pod/nginxpod created

1. Check pods in namespace raman:

bash

CopyEdit

kubectl get pods -n raman

✅ Pod should be in Running state.

1. Get detailed pod info:

bash

CopyEdit

kubectl get pods -o wide -n raman

**🔹 Step 4: Inspect the pod**

1. Describe the pod to see events, container details, and node assignment:

bash

CopyEdit

kubectl describe pod nginxpod -n raman

1. List cluster-wide events:

bash

CopyEdit

kubectl get events

**🔹 Step 5: Execute commands inside containers**

You can interact with specific containers using -c <container\_name>:

1. Enter the **nginx** container:

bash

CopyEdit

kubectl exec -it nginxpod -n raman -c nginxcon -- /bin/bash

✅ Run commands like:

bash

CopyEdit

hostname

ls /usr/share/nginx/html

Exit:

bash

CopyEdit

exit

1. Enter the **redis** container:

bash

CopyEdit

kubectl exec -it nginxpod -n raman -c rediscon -- /bin/bash

✅ Create a directory inside the container:

bash

CopyEdit

mkdir /ramandir

ls /

Exit:

bash

CopyEdit

exit

1. List files inside rediscon without entering:

bash

CopyEdit

kubectl exec -it nginxpod -n raman -c rediscon -- ls /

**🔹 Step 6: Modify YAML and redeploy**

If you make changes to pod.yaml:

1. Delete the existing pod:

bash

CopyEdit

kubectl delete pod nginxpod -n raman

1. Apply the updated YAML:

bash

CopyEdit

kubectl create -f pod.yaml

**🔹 Step 7: Clean up resources**

Delete all pods in namespace raman:

bash

CopyEdit

kubectl delete pods --all -n raman

**📖 Key Concepts Covered**

✅ Multi-container pods and why they are useful (e.g., sidecars).  
✅ kubectl exec to interact with containers.  
✅ YAML manifest structure for Pod.  
✅ Pod lifecycle: create → inspect → delete → recreate.

**📝 Kubernetes Hands-on Lab Guide: From Containers to Deployments & ReplicaSets**

**🎯 Lab Objective**

This lab shows the transition from **manual container management in Docker** to **automated orchestration in Kubernetes** using **Deployments** and **ReplicaSets**.

At the end of this lab, you’ll:  
✅ Understand why Kubernetes is needed for container orchestration.  
✅ Create a Deployment that manages multiple pod replicas.  
✅ Observe **self-healing behavior** by deleting pods.  
✅ Explore how ReplicaSets maintain pod count automatically.

**🛠 Prerequisites**

* Docker and Kubernetes cluster accessible.
* kubectl installed and configured to communicate with the cluster.
* A namespace raman created (if not, create it).

**🗂 Lab Outline**

| **Step** | **Task** |
| --- | --- |
| 1 | Compare Docker and Kubernetes at container orchestration level |
| 2 | Explore API resources for pods, nodes, deployments, and ReplicaSets |
| 3 | Create a Deployment with multiple replicas |
| 4 | Inspect Deployment, ReplicaSet, and Pods |
| 5 | Test self-healing by deleting pods |
| 6 | Clean up resources |

**🧑‍💻 Step-by-Step Guide**

**🔹 Step 1: From Docker to Kubernetes**

**📝 *Theory + Demo***

**Docker limitations (managing containers manually):**

* Create containers manually:

bash

CopyEdit

docker run -d --name httpd1 httpd

docker run -d --name httpd2 httpd

* No automatic restart if container dies.
* Scaling = manual effort (e.g., repeat docker run).
* No built-in syncing between containers.

✅ Explain: “In Kubernetes, Deployments & ReplicaSets solve these problems by maintaining desired state and enabling scalability.”

**🔹 Step 2: Explore Kubernetes API resources**

1. **List API resources related to Pods, Nodes, Namespaces, Deployments, and ReplicaSets**:

bash

CopyEdit

k api-resources | grep pod

k api-resources | grep node

k api-resources | grep ns

k api-resources | grep deploy

k api-resources | grep rs

✅ **Observation:**

* Pod is a core resource.
* Deployment & ReplicaSet are managed by **Controller Manager** (Deployment controller, ReplicaSet controller).

**🔹 Step 3: Create a Deployment**

Deploy an application (httpd) with **5 replicas** in the raman namespace:

bash

CopyEdit

k create deploy ramandep1 -n raman --replicas=5 --image=httpd

✅ **Expected output:**  
deployment.apps/ramandep1 created

**🔹 Step 4: Inspect the Deployment**

1. View the Deployment:

bash

CopyEdit

k get deploy -n raman

1. Describe the Deployment to see:
   * Strategy (e.g., RollingUpdate)
   * Desired vs current replicas
   * Events and pod template

bash

CopyEdit

k describe deploy ramandep1 -n raman

✅ **Observation:**  
Deployment ensures **desired replicas** exist by creating a ReplicaSet.

**🔹 Step 5: Check associated ReplicaSet**

1. View ReplicaSets in the raman namespace:

bash

CopyEdit

k get rs -n raman

✅ **Observation:**  
ReplicaSet maintains pod count for the Deployment.

**🔹 Step 6: Inspect created pods**

1. View pods created by the ReplicaSet:

bash

CopyEdit

k get pods -n raman

1. Include details (node, IP, etc.):

bash

CopyEdit

k get pods -n raman -o wide

**🔹 Step 7: Test self-healing behavior**

1. Delete one of the pods manually:

bash

CopyEdit

k delete pod <pod-name> -n raman

For example:

bash

CopyEdit

k delete pod ramandep1-559bb87ff6-rvklw -n raman

1. Watch pods being recreated:

bash

CopyEdit

k get pods -n raman -o wide

✅ **Observation:**  
The ReplicaSet detects missing pod and spins up a replacement automatically.

**🔹 Step 8: Clean up resources**

Delete the Deployment (this also deletes ReplicaSets and Pods):

bash

CopyEdit

k delete deploy ramandep1 -n raman

Verify cleanup:

bash

CopyEdit

k get deploy,rs,pods -n raman

✅ All resources should be gone.

**📖 Key Concepts Covered**

| **Concept** | **Explanation** |
| --- | --- |
| **Deployment** | Manages ReplicaSets, enables declarative updates and scaling |
| **ReplicaSet** | Ensures desired number of pod replicas are running |
| **Self-healing** | Kubernetes recreates deleted pods to maintain desired state |
| **Controller Manager** | Handles background controllers (Deployment, ReplicaSet, etc.) |
| **Scaling** | Can increase/decrease replicas dynamically using kubectl scale |

**📝 Key Commands Summary**

| **Command** | **Description** |
| --- | --- |
| `k api-resources | grep <resource>` |
| k create deploy <name> --replicas=<n> --image=<img> | Create Deployment with replicas |
| k get deploy,rs,pods -n <namespace> | View Deployment, ReplicaSets, Pods |
| k delete pod <pod> -n <namespace> | Delete pod (triggers self-healing) |
| k delete deploy <name> -n <namespace> | Delete Deployment and related resources |

**📝 Kubernetes Lab Guide: Deployments, ReplicaSets, Labels & Selectors**

**🎯 Lab Objective**

This lab focuses on:  
✅ Understanding **Deployments** and **ReplicaSets** in Kubernetes.  
✅ Using **labels** and **selectors** to manage and target Pods.  
✅ Observing **pod creation, scaling, and replacement** behaviors.

At the end of this lab, you’ll:

* Be able to create and manage Deployments and ReplicaSets.
* Understand how **matchLabels** connects controllers to pods.
* See the **difference between Deployment and ReplicaSet** in action.

**🛠 Prerequisites**

* A Kubernetes cluster with kubectl configured.
* Namespace raman created:

bash

CopyEdit

kubectl create ns raman

**🗂 Lab Outline**

| **Step** | **Task** |
| --- | --- |
| 1 | Author and deploy a **Deployment** |
| 2 | Inspect Deployment, ReplicaSet, and Pods |
| 3 | Update Deployment and observe changes |
| 4 | Delete Deployment |
| 5 | Author and deploy a **ReplicaSet** |
| 6 | Inspect ReplicaSet and Pods |
| 7 | Compare Deployment vs ReplicaSet |
| 8 | Clean up resources |

**🧑‍💻 Step-by-Step Guide**

**🔹 Step 1: Author a Deployment YAML**

Create a file deploy.yaml:

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: rk-deployment

namespace: raman

labels:

purpose: trainingdeployment

spec:

replicas: 1

selector:

matchLabels:

rk: rep

template:

metadata:

labels:

rk: rep

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

✅ **Key parts explained:**

* **metadata.labels**: Apply a label to the Deployment itself.
* **selector.matchLabels**: Ensures Deployment manages pods with rk: rep.
* **template.metadata.labels**: Label applied to Pods created by this Deployment.

**🔹 Step 2: Deploy and verify**

1. Apply the manifest:

bash

CopyEdit

kubectl create -f deploy.yaml

✅ **Expected output:**  
deployment.apps/rk-deployment created

1. Check pods in raman:

bash

CopyEdit

kubectl get pods -n raman

1. Check Deployment details:

bash

CopyEdit

kubectl describe deploy rk-deployment -n raman

1. Check ReplicaSets created by the Deployment:

bash

CopyEdit

kubectl get rs -n raman

kubectl describe rs -n raman

✅ **Observation:**

* Deployment creates a ReplicaSet.
* ReplicaSet creates Pods matching rk: rep.

**🔹 Step 3: Update Deployment**

Change the image version in deploy.yaml to nginx:1.16.0:

yaml

CopyEdit

image: nginx:1.16.0

Apply the changes:

bash

CopyEdit

kubectl apply -f deploy.yaml

✅ **Observe rolling update:**

bash

CopyEdit

kubectl get pods -n raman

**🔹 Step 4: Delete Deployment**

Delete Deployment (ReplicaSet and Pods will also be removed):

bash

CopyEdit

kubectl delete -f deploy.yaml

✅ Verify cleanup:

bash

CopyEdit

kubectl get pods,rs,deploy -n raman

**🔹 Step 5: Author a ReplicaSet YAML**

Create a file replicaset.yaml:

yaml

CopyEdit

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: frontend

namespace: raman

labels:

app: guestbook

tier: frontend

spec:

replicas: 3

selector:

matchLabels:

tier: frontend

template:

metadata:

labels:

tier: frontend

spec:

containers:

- name: php-redis

image: us-docker.pkg.dev/google-samples/containers/gke/gb-frontend:v5

✅ **Key parts explained:**

* **ReplicaSet** directly manages Pods matching tier: frontend.
* No rolling updates (unlike Deployments).

**🔹 Step 6: Deploy and verify**

1. Apply the manifest:

bash

CopyEdit

kubectl create -f replicaset.yaml

1. Check Pods created:

bash

CopyEdit

kubectl get pods -n raman

kubectl get rs -n raman

✅ **Observation:**  
3 Pods created by ReplicaSet.

1. Delete one Pod:

bash

CopyEdit

kubectl delete pod <pod-name> -n raman

✅ **Observe:** ReplicaSet creates a new Pod automatically.

**🔹 Step 7: Compare Deployment vs ReplicaSet**

| **Feature** | **Deployment** | **ReplicaSet** |
| --- | --- | --- |
| **Manages ReplicaSets?** | ✅ Yes | ❌ No |
| **Rolling Updates?** | ✅ Yes | ❌ No |
| **Scaling** | ✅ Dynamic | ✅ Dynamic |
| **Recommended?** | ✅ For production workloads | 🔸 Rarely (used internally) |

**🔹 Step 8: Clean up**

Delete ReplicaSet:

bash

CopyEdit

kubectl delete -f replicaset.yaml

Verify cleanup:

bash

CopyEdit

kubectl get pods,rs,deploy -n raman

**📖 Key Concepts Covered**

✅ **Labels & Selectors:** Connect controllers (Deployment/ReplicaSet) to Pods.  
✅ **Deployment:** Declarative updates, manages ReplicaSets, ideal for production.  
✅ **ReplicaSet:** Ensures desired pod count, no rollout strategies.  
✅ **Self-healing:** Both controllers replace missing pods.

**📝 Key Commands Summary**

| **Command** | **Description** |
| --- | --- |
| kubectl create -f <file> | Create resource from YAML |
| kubectl apply -f <file> | Update resource declaratively |
| kubectl get pods,rs,deploy -n <namespace> | View Pods, ReplicaSets, Deployments |
| kubectl describe deploy/rs <name> | Detailed resource info |
| kubectl delete -f <file> | Delete resource from YAML |

**📝 Kubernetes Lab Guide: Deployment Rollouts, Updates & Rollbacks**

**🎯 Lab Objectives**

In this lab you will:  
✅ Deploy an application using a Deployment manifest  
✅ Perform rolling updates to upgrade container images  
✅ Observe the creation of new ReplicaSets for each revision  
✅ Roll back to a previous version of the application  
✅ Explore Deployment revision history

This lab demonstrates **Kubernetes Deployment strategies** for **zero downtime updates** and how to recover quickly from bad releases.

**🛠 Prerequisites**

* Kubernetes cluster up and running
* kubectl configured and working
* Namespace raman created:

bash

CopyEdit

kubectl create ns raman

**🗂 Lab Overview**

| **Step** | **Task** |
| --- | --- |
| 1 | Author and apply the first Deployment (v1: nginx:1.14.1) |
| 2 | Perform first rolling update (v2: nginx:1.14.2) |
| 3 | Perform second rolling update (v3: nginx:latest) |
| 4 | Explore Deployment revision history |
| 5 | Roll back to the first version (nginx:1.14.1) |
| 6 | Inspect ReplicaSets and Pods during updates and rollbacks |
| 7 | Clean up resources |

**🧑‍💻 Step-by-Step Guide**

**🔹 Step 1: Author first Deployment**

Create a file deploy.yaml:

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: rk-deployment

namespace: raman

labels:

purpose: trainingdeployment

spec:

replicas: 4

selector:

matchLabels:

rk: rep

template:

metadata:

labels:

rk: rep

spec:

containers:

- name: nginx

image: nginx:1.14.1 # first version

ports:

- containerPort: 80

✅ **Key Concepts:**

* **Deployment** manages **ReplicaSets** for each version of the pod template.
* **Revision 1** will be created for nginx:1.14.1.

Apply and record the change:

bash

CopyEdit

kubectl apply -f deploy.yaml --record=true

✅ **Expected Output:**  
deployment.apps/rk-deployment created

**🔹 Step 2: Verify Deployment**

1. Check Deployment:

bash

CopyEdit

kubectl get deploy -n raman

1. Check ReplicaSets:

bash

CopyEdit

kubectl get rs -n raman

✅ You should see one ReplicaSet like:

CopyEdit

rk-deployment-6ffc8cbb9d 4 4 4

1. Check Pods:

bash

CopyEdit

kubectl get pods -n raman

✅ All 4 pods should be running with nginx:1.14.1.

1. Check container images:

bash

CopyEdit

kubectl describe pods -n raman | grep -i image

**🔹 Step 3: Perform rolling update (v2)**

Update deploy.yaml with a new image version:

yaml

CopyEdit

image: nginx:1.14.2 # second version

Apply changes and record:

bash

CopyEdit

kubectl apply -f deploy.yaml --record=true

✅ **Observe rolling update progress:**

bash

CopyEdit

kubectl rollout status deployment rk-deployment -n raman

**🔹 Step 4: Inspect revisions**

1. Check all ReplicaSets:

bash

CopyEdit

kubectl get rs -n raman

✅ You’ll see:

* Old ReplicaSet scaled to 0/0 pods.
* New ReplicaSet scaled to 4/4 pods.

1. Check Deployment history:

bash

CopyEdit

kubectl rollout history deployment rk-deployment -n raman

✅ You should see:

csharp

CopyEdit

REVISION CHANGE-CAUSE

1 kubectl apply --record=true

2 kubectl apply --record=true

1. Inspect a specific revision:

bash

CopyEdit

kubectl rollout history deployment rk-deployment -n raman --revision=1

**🔹 Step 5: Perform rolling update (v3)**

Update deploy.yaml again:

yaml

CopyEdit

image: nginx:latest # third version

Apply and record:

bash

CopyEdit

kubectl apply -f deploy.yaml --record=true

✅ Check rollout progress:

bash

CopyEdit

kubectl rollout status deployment rk-deployment -n raman

✅ Inspect history:

bash

CopyEdit

kubectl rollout history deployment rk-deployment -n raman

**🔹 Step 6: Roll back to Revision 1**

If nginx:latest causes issues, roll back:

bash

CopyEdit

kubectl rollout undo deployment rk-deployment -n raman --to-revision=1

✅ **Expected:**

* New ReplicaSet created for nginx:1.14.1
* ReplicaSets for v2 and v3 scaled to zero.

**🔹 Step 7: Verify rollback**

1. Check Pods and their images:

bash

CopyEdit

kubectl get pods -n raman

kubectl describe pods -n raman | grep -i image

✅ All Pods should run nginx:1.14.1.

1. Inspect ReplicaSets:

bash

CopyEdit

kubectl get rs -n raman

✅ Older ReplicaSet will now be scaled up.

**🔹 Step 8: Clean up resources**

Delete the Deployment (removes ReplicaSets and Pods):

bash

CopyEdit

kubectl delete -f deploy.yaml

Verify:

bash

CopyEdit

kubectl get deploy,rs,pods -n raman

✅ All resources should be gone.

**📖 Key Concepts Covered**

| **Feature** | **Description** |
| --- | --- |
| **Deployment revisions** | Each update creates a new ReplicaSet and revision |
| **Rolling updates** | New pods created gradually while old pods are terminated |
| **Rollbacks** | Restore previous ReplicaSets with kubectl rollout undo |
| **--record=true** | Records change cause in rollout history |

**📝 Key Commands Summary**

| **Command** | **Description** |
| --- | --- |
| kubectl apply -f <file> --record=true | Apply changes and record history |
| kubectl rollout status deployment <name> | Check rollout progress |
| kubectl rollout history deployment <name> | View revision history |
| kubectl rollout undo deployment <name> | Rollback to previous revision |
| kubectl rollout undo deployment <name> --to-revision=<num> | Rollback to specific revision |

**📝 Kubernetes Lab Guide: Node Labels, Taints, Tolerations & Pod Scheduling**

**🎯 Lab Objectives**

In this lab you will:  
✅ Label Kubernetes nodes for pod selection  
✅ Apply taints to nodes to control pod scheduling  
✅ Deploy workloads and observe pod distribution across nodes  
✅ Scale deployments and test scheduler behavior  
✅ Remove taints and understand tolerations

By the end of this lab, participants will understand **how Kubernetes schedules pods based on node labels and taints**, and how **affinity and tolerations** control placement.

**🛠 Prerequisites**

* Kubernetes cluster with multiple worker nodes (w1, w2) and a master node
* Namespace raman created:

bash

CopyEdit

kubectl create ns raman

**🗂 Lab Overview**

| **Step** | **Task** |
| --- | --- |
| 1 | Verify node details |
| 2 | Apply labels to nodes |
| 3 | Taint a node to block pod scheduling |
| 4 | Deploy a workload and observe pod placement |
| 5 | Scale the workload and observe scheduler behavior |
| 6 | Remove taints and reschedule pods |
| 7 | Clean up |

**🧑‍💻 Step-by-Step Guide**

**🔹 Step 1: Verify nodes in the cluster**

List all nodes in the cluster:

bash

CopyEdit

kubectl get nodes -o wide

✅ Check the names of worker nodes (e.g., w1, w2).

**🔹 Step 2: Apply labels to nodes**

Add labels to nodes for grouping:

bash

CopyEdit

kubectl label node w1 app=first

kubectl label node w2 app=second

✅ Verify labels:

bash

CopyEdit

kubectl describe node w1 | grep -i labels

kubectl describe node w2 | grep -i labels

**🔹 Step 3: Apply taints to control pod scheduling**

Taint w1 so no new pods can be scheduled on it:

bash

CopyEdit

kubectl taint nodes w1 app=first:NoSchedule

✅ Verify taint:

bash

CopyEdit

kubectl describe node w1 | grep -i taint

✅ Check master node taints:

bash

CopyEdit

kubectl describe node master | grep -i taint

**🔹 Step 4: Deploy a workload and observe**

Deploy a new Deployment in the raman namespace:

bash

CopyEdit

kubectl create deploy testdeploy --replicas=5 --image=httpd -n raman

✅ Check pods and their node assignments:

bash

CopyEdit

kubectl get pods -n raman -o wide

✅ **Expected behavior:**

* Pods will not be scheduled on w1 because of the NoSchedule taint.
* Scheduler will prefer w2 or other available nodes.

**🔹 Step 5: Scale the deployment**

Scale up to increase pod count:

bash

CopyEdit

kubectl scale deploy testdeploy --replicas=10 -n raman

✅ Check pods again:

bash

CopyEdit

kubectl get pods -n raman -o wide

✅ Observe if any pods are stuck in Pending state because there are insufficient taint-free nodes.

**🔹 Step 6: Remove taints to allow scheduling**

Remove the taint from w1:

bash

CopyEdit

kubectl taint nodes w1 app-

✅ Verify taints are gone:

bash

CopyEdit

kubectl describe node w1 | grep -i taint

Scale down the deployment to zero:

bash

CopyEdit

kubectl scale deploy testdeploy --replicas=0 -n raman

✅ Scale back up:

bash

CopyEdit

kubectl scale deploy testdeploy --replicas=10 -n raman

✅ Check pod distribution:

bash

CopyEdit

kubectl get pods -n raman -o wide

✅ Now pods should schedule on w1 as well.

**🔹 Step 7: Apply a stronger taint**

Apply a NoExecute taint to force eviction:

bash

CopyEdit

kubectl taint nodes w1 app=first:NoExecute

✅ Observe:

* Any pods running on w1 without tolerations will be evicted.

✅ Check pods running on w1:

bash

CopyEdit

kubectl get pods -n raman -o wide | grep w1

**🔹 Step 8: Clean up**

Delete the deployment:

bash

CopyEdit

kubectl delete deploy testdeploy -n raman

Remove taints from w1:

bash

CopyEdit

kubectl taint nodes w1 app-

✅ Verify:

bash

CopyEdit

kubectl describe node w1 | grep -i taint

**📖 Key Concepts Covered**

| **Concept** | **Description** |
| --- | --- |
| **Node Labels** | Tags on nodes used for affinity/anti-affinity pod placement |
| **Taints** | Prevent pods from being scheduled unless they tolerate the taint |
| **NoSchedule** | New pods will not be scheduled on tainted nodes |
| **NoExecute** | Existing pods on tainted node will be evicted unless tolerations exist |
| **Pod Scheduling** | Kubernetes scheduler places pods on nodes based on resources and policies |

**📝 Key Commands Summary**

| **Command** | **Description** |
| --- | --- |
| kubectl label node <node> <key>=<value> | Add label to node |
| kubectl taint nodes <node> <key>=<value>:<effect> | Taint node |
| kubectl taint nodes <node> <key>- | Remove taint |
| kubectl get pods -n <ns> -o wide | Check pod-to-node assignments |
| kubectl scale deploy <name> --replicas=<count> -n <ns> | Scale deployment |