**🔬 Lab Guide: Creating and Managing a Pod in a Custom Namespace**

**🧪 Lab Objective**

By the end of this lab, you will:

* Create a custom namespace.
* Deploy a Pod using a YAML manifest into the custom namespace.
* Verify and troubleshoot the Pod.
* Expose and access the Pod internally.

**🧰 Lab Prerequisites**

Ensure the following before starting:

* A running Kubernetes cluster (in this case, set up via kubeadm).
* Kubectl configured and accessible (alias k=kubectl already set).
* Internet access from nodes to pull container images (e.g., from Docker Hub).
* A Linux terminal environment (like Ubuntu).

**🧭 Step-by-Step Instructions**

**✅ Step 1: Create a Custom Namespace**

Namespaces in Kubernetes are used to logically isolate resources. This is especially useful in multi-tenant environments or for organizing environments (dev/test/prod).

bash

CopyEdit

k create ns raman

🧠 **Concept**: A namespace is an abstraction that allows separation of resources within a cluster. Any object (Pod, Service, etc.) can be scoped to a namespace. Resources not assigned a namespace go to the default namespace.

**🔍 Step 2: Verify Namespace**

bash

CopyEdit

k get ns

Expected output:

mathematica

CopyEdit

NAME STATUS AGE

default Active ...

kube-system Active ...

kube-public Active ...

raman Active ...

**✍️ Step 3: Create a Pod Manifest**

Use any text editor to create a YAML manifest file (pod.yml):

bash

CopyEdit

vi pod.yml

Paste the following content:

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: ramanapp2

namespace: raman # The Pod will be deployed into 'raman' namespace

spec:

containers:

- name: ramancon

image: nginx:1.14.2

ports:

- containerPort: 80

💡 **Explanation**:

* apiVersion: v1 — The Pod resource is part of the core API group.
* kind: Pod — We're defining a Pod directly (not via a higher-level controller like Deployment).
* metadata.namespace — Ensures the pod is created inside the raman namespace.
* spec.containers — Defines one or more containers to run inside the Pod.
* image: nginx:1.14.2 — Pulls this version of the nginx image from Docker Hub.
* containerPort: 80 — Tells Kubernetes that the container listens on port 80 (not for exposure; used by Services or probes).

**🚀 Step 4: Apply the Manifest**

Create the Pod using the YAML file:

bash

CopyEdit

k create -f pod.yml

To verify:

bash

CopyEdit

k get pods -n raman

Expected output:

sql

CopyEdit

NAME READY STATUS RESTARTS AGE

ramanapp2 1/1 Running 0 10s

**🛠 Step 5: Describe and Debug the Pod**

Check detailed info:

bash

CopyEdit

k describe pod ramanapp2 -n raman

This will show:

* Events (pulling image, started container, etc.)
* IP address
* Container details (image, ports, restart count)
* Any error if image pull fails or container crashes.

**🌐 Step 6: Access the Pod (From Master Node)**

To test Nginx is running, curl the Pod IP from another Pod or node (assuming no network policies block it):

bash

CopyEdit

k get pods -o wide -n raman

Sample output:

nginx

CopyEdit

NAME READY STATUS IP NODE ...

ramanapp2 1/1 Running 10.244.1.8 worker1 ...

Then from the **master node or another Pod on the same network**:

bash

CopyEdit

curl 10.244.1.8

Expected result: The default nginx welcome page HTML.

**🧽 Step 7: Clean Up**

Once done, you can delete the Pod and namespace:

bash

CopyEdit

k delete pod ramanapp2 -n raman

k delete ns raman

**📌 Additional Notes**

**🔄 Alternative: Imperative Pod Creation**

You had run this earlier:

bash

CopyEdit

k run ramanapp --image httpd -n raman

💡 kubectl run is a quick way to create a Pod but does not persist config in a YAML file (not recommended for production use). YAML manifests give better control, documentation, and versioning.

**🧠 Bonus Learning**

**🔗 How to Expose Pod via Service (Optional)**

If you want to access the Nginx app from outside or cluster-internal network reliably:

bash

CopyEdit

k expose pod ramanapp2 --type=NodePort --port=80 -n raman

Then:

bash

CopyEdit

k get svc -n raman

Use the NodePort (e.g., 30080) to access via curl <NodeIP>:NodePort.

**🧪 Validation Checklist**

| **Task** | **Command** | **Verified** |
| --- | --- | --- |
| Namespace created | k get ns | ✅ |
| Pod YAML defined correctly | cat pod.yml | ✅ |
| Pod created | k get pods -n raman | ✅ |
| Pod running and accessible | curl <Pod IP> | ✅ |

**📘 Summary**

In this lab, you:

* Learned about Kubernetes namespaces.
* Created a Pod using YAML.
* Deployed and verified the Pod.
* Accessed it via internal IP.

This foundational knowledge is essential before working with Deployments, Services, and other abstractions.

**🧪 Lab Guide: Deploying and Scaling Applications Using Kubernetes Deployments**

**🎯 Lab Objectives**

You will learn how to:

* Create a Deployment to manage multiple Pod replicas.
* Understand the relationship between Deployments, ReplicaSets, and Pods.
* Scale Deployments dynamically.
* Observe and verify how Kubernetes manages Pods via ReplicaSets.

**🧰 Prerequisites**

Ensure the following:

* Kubernetes cluster is running (kubeadm based setup is fine).
* kubectl is configured and accessible (alias k=kubectl is assumed).
* A namespace named raman already exists (you created it earlier).

**🧭 Step-by-Step Instructions**

**✅ Step 1: Check Resources Related to Deployments**

To understand which resources exist for Deployments:

bash

CopyEdit

k api-resources | grep -i deploy

Expected output:

bash

CopyEdit

deployments deploy apps/v1 true Deployment

To list ReplicaSets (used by Deployments):

bash

CopyEdit

k api-resources | grep -i rs

Expected output:

bash

CopyEdit

replicasets rs apps/v1 true ReplicaSet

🧠 **Concepts**:

* **Deployment**: A controller that manages ReplicaSets.
* **ReplicaSet**: Ensures the desired number of Pod replicas are always running.
* **Pod**: The smallest deployable unit.

**🛠 Step 2: Create a Deployment with Multiple Replicas**

We’ll use the httpd (Apache HTTP Server) image to demonstrate.

bash

CopyEdit

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

💡 This is an **imperative command** that:

* Creates a Deployment named ramandep1.
* Uses the httpd container image.
* Starts with 5 Pod replicas.
* Deploys it in the raman namespace.

**🔍 Step 3: Inspect the Deployment**

bash

CopyEdit

k get deploy -n raman

Sample output:

pgsql

CopyEdit

NAME READY UP-TO-DATE AVAILABLE AGE

ramandep1 5/5 5 5 30s

Check the Pods:

bash

CopyEdit

k get pods -n raman

Sample output:

python-repl

CopyEdit

NAME READY STATUS RESTARTS AGE

ramandep1-559bb87ff6-xxxxx 1/1 Running 0 45s

... (4 more similar entries)

Check the underlying ReplicaSet:

bash

CopyEdit

k get rs -n raman

Sample output:

pgsql

CopyEdit

NAME DESIRED CURRENT READY AGE

ramandep1-559bb87ff6 5 5 5 1m

🧠 **How It Works**:

* The Deployment created a ReplicaSet.
* The ReplicaSet maintains the desired number of Pods (5 in this case).

**🧹 Step 4: Delete a Pod Manually and Observe**

bash

CopyEdit

k delete pod ramandep1-559bb87ff6-8nst5 -n raman

Then check Pods again:

bash

CopyEdit

k get pods -n raman

✅ **Observation**: A new Pod with a different name will be automatically created. This is because the ReplicaSet ensures that the **desired number of replicas** is maintained.

**🔄 Step 5: Scale the Deployment (Up and Down)**

**Scale Down to 1 Replica:**

bash

CopyEdit

k scale deploy ramandep1 --replicas=1 -n raman

Check Pods:

bash

CopyEdit

k get pods -n raman

✅ **Result**: Only one Pod remains; the others are terminated by the ReplicaSet under Deployment control.

**Scale Up to 5 Replicas Again:**

bash

CopyEdit

k scale deploy ramandep1 --replicas=5 -n raman

**Scale Up to 10:**

bash

CopyEdit

k scale deploy ramandep1 --replicas=10 -n raman

**Scale Up to 11:**

bash

CopyEdit

k scale deploy ramandep1 --replicas=11 -n raman

🧠 **Concept**:

* Scaling affects the ReplicaSet.
* Deployment automatically adjusts Pods via its associated ReplicaSet.

**🧠 Additional Notes**

**🔁 Rolling Updates (optional)**

You can update the Deployment’s image with:

bash

CopyEdit

k set image deploy/ramandep1 httpd=httpd:2.4 -n raman

Then monitor:

bash

CopyEdit

k rollout status deploy ramandep1 -n raman

**🧯 Rollback (optional)**

If the update causes issues:

bash

CopyEdit

k rollout undo deploy ramandep1 -n raman

**📌 Summary of Key Concepts**

| **Concept** | **Description** |
| --- | --- |
| **Deployment** | Manages ReplicaSets; supports scaling and rolling updates |
| **ReplicaSet** | Maintains a stable set of replica Pods |
| **Pod** | The actual container workload |
| **Scaling** | Changes the number of replicas |
| **Self-Healing** | K8s restarts failed Pods via RS |

**🧪 Validation Checklist**

| **Task** | **Command** | **Verified** |
| --- | --- | --- |
| Deployment created | k get deploy -n raman | ✅ |
| Pods created | k get pods -n raman | ✅ |
| ReplicaSet active | k get rs -n raman | ✅ |
| Scaling worked | k scale deploy ... and k get pods | ✅ |
| Self-healing | Deleted Pod re-created | ✅ |

**🧼 Clean-up (Optional)**

bash

CopyEdit

k delete deploy ramandep1 -n raman

k delete ns raman

**🧪 Lab Guide: Kubernetes Deployments, Labels, Selectors, and Object Inspection**

**🎯 Lab Objectives**

In this lab, you will:

* Create Deployments using imperative commands.
* Explore Pod metadata, including labels.
* Use kubectl selectors to filter resources based on labels.
* Understand how Deployments, ReplicaSets, and Pods are related via labels.
* Practice debugging and visibility with describe, get, and selectors.

**🧰 Prerequisites**

* Kubernetes cluster and kubectl configured.
* Namespace raman and sunit should already exist.
* You should already be using the alias k=kubectl.

**🧭 Step-by-Step Lab Instructions**

**🔁 Step 1: Delete Old Deployment**

First, clean up any existing Deployment in the raman namespace:

bash

CopyEdit

k delete deploy -n raman ramandep1

Verify Pods and ReplicaSets are removed:

bash

CopyEdit

k get pods -n raman

k get rs -n raman

**📄 Step 2: Create a Deployment Using Imperative Command**

Let’s create a new Deployment named ramanadep1:

bash

CopyEdit

k create deploy ramanadep1 -n raman --image=httpd --replicas=3

This will:

* Create a Deployment with 3 replicas.
* Use the httpd image.
* Deploy all objects in the raman namespace.

Verify Pods:

bash

CopyEdit

k get pods -n raman

You should see 3 Pods like:

python-repl

CopyEdit

ramanadep1-7b6858cbc5-xxxxx

...

Check ReplicaSet:

bash

CopyEdit

k get rs -n raman

**🔍 Step 3: Inspect Labels and Metadata**

**Describe Pods**

bash

CopyEdit

k describe pod -n raman | grep -i label

This shows the labels attached to the Pod, typically something like:

makefile

CopyEdit

Labels: app=ramanadep1

**Describe ReplicaSet**

bash

CopyEdit

k describe rs ramanadep1-7b6858cbc5 -n raman | grep -i label

Expected:

makefile

CopyEdit

Labels: app=ramanadep1

Selector: app=ramanadep1

**Describe Deployment**

bash

CopyEdit

k describe deploy -n raman | grep -i label

Expected:

makefile

CopyEdit

Labels: app=ramanadep1

Selector: matchLabels: app=ramanadep1

🧠 **Concept**:

* Deployments define **selectors** to manage Pods.
* ReplicaSets and Pods get created with the **same label** (app=ramanadep1).
* This label is what links them together.

**🎯 Step 4: Use Label Selectors to Filter Resources**

You can use --selector (or -l) to query objects based on labels.

**Get All Pods Across Namespaces with app=ramanadep1:**

bash

CopyEdit

k get pods -A --selector "app=ramanadep1"

This will return Pods from any namespace matching this label. Output:

python-repl

CopyEdit

NAMESPACE NAME READY STATUS RESTARTS AGE

raman ramanadep1-7b6858cbc5-xxxxx 1/1 Running 0 1m

...

🧠 **Why This Matters**:

* **Selectors** are key for querying, managing, and grouping objects.
* Services, NetworkPolicies, and many other K8s resources depend on label selectors.

**🔍 Step 5: Investigate Another Namespace (sunit)**

You explored a Pod named sunitapp in the sunit namespace:

bash

CopyEdit

k describe pod sunitapp -n sunit | grep -i label

Sample output:

arduino

CopyEdit

Labels: run=sunitapp

Check if a Deployment exists:

bash

CopyEdit

k describe deploy -n sunit | grep -i label

To list Pods matching a label across namespaces:

bash

CopyEdit

k get pods -A --selector run=sunitapp

**🧹 Optional: YAML-Based Deployment**

You opened vi deploy.yml earlier. Here’s how to write a Deployment in YAML that matches what you did imperatively.

**📄 Sample deploy.yml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: ramanadep1

namespace: raman

spec:

replicas: 3

selector:

matchLabels:

app: ramanadep1

template:

metadata:

labels:

app: ramanadep1

spec:

containers:

- name: httpd

image: httpd:latest

ports:

- containerPort: 80

Apply it:

bash

CopyEdit

k apply -f deploy.yml

**🧠 Summary of Core Concepts**

| **Concept** | **Explanation** |
| --- | --- |
| **Label** | Key-value metadata assigned to objects like Pods and Deployments |
| **Selector** | Filters resources based on matching labels |
| **Deployment** | Controller that manages ReplicaSets and provides declarative updates |
| **ReplicaSet** | Ensures a stable set of identical Pods |
| **Imperative vs Declarative** | kubectl create is imperative, kubectl apply -f is declarative |

**✅ Validation Checklist**

| **Task** | **Command** | **Verified** |
| --- | --- | --- |
| Deployment created | k get deploy -n raman | ✅ |
| Pods created | k get pods -n raman | ✅ |
| Labels applied | k describe pod ... | ✅ |
| Filtered with selectors | k get pods -A --selector app=ramanadep1 | ✅ |
| ReplicaSet linked with Deployment | k get rs -n raman | ✅ |

**🧼 Clean-Up (Optional)**

bash

CopyEdit

k delete deploy ramanadep1 -n raman

k delete ns raman