**Lab Guide: Kubernetes Deployment with Rollout and Version Control**

**Objective**

* Deploy a multi-replica nginx application.
* Manage different versions (image tags).
* Track rollout history and rollback to a specific version.

**Pre-requisites**

* Kubernetes cluster up and running (e.g., Minikube, kind, or cloud-managed K8s).
* kubectl CLI configured.
* A namespace raman created.

bash

CopyEdit

kubectl create namespace raman

**Step 1: Create Initial Deployment (v1 - nginx:1.14.2)**

yaml

CopyEdit

# deploy.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-yml

labels:

purpose: training

namespace: raman

spec:

replicas: 5

selector:

matchLabels:

name: myapp

template:

metadata:

labels:

name: myapp

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

Apply the deployment with rollout recording:

bash

CopyEdit

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

Check resources:

bash

CopyEdit

kubectl get all -n raman

**Step 2: Upgrade to v2 (nginx:latest)**

Edit the deployment file:

yaml

CopyEdit

image: nginx:latest

Reapply:

bash

CopyEdit

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

kubectl get all -n raman

Notice:

* A new ReplicaSet is created.
* Old pods are terminated, new ones launched.

Check rollout history:

bash

CopyEdit

kubectl rollout history deployment raman-dep-yml -n raman

Optional: Inspect a revision:

bash

CopyEdit

kubectl rollout history deployment raman-dep-yml -n raman --revision=1

kubectl rollout history deployment raman-dep-yml -n raman --revision=2

**Step 3: Downgrade to v3 (nginx:1.14.1)**

Edit again:

yaml

CopyEdit

image: nginx:1.14.1

Apply the change:

bash

CopyEdit

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

kubectl get all -n raman

Check rollout status:

bash

CopyEdit

kubectl rollout status deployment raman-dep-yml -n raman

Describe new pod to verify the image:

bash

CopyEdit

kubectl describe pod <pod-name> -n raman | grep -i image

**Step 4: Rollback Operations**

**List all revisions:**

bash

CopyEdit

kubectl rollout history deployment raman-dep-yml -n raman

**Rollback to revision 1:**

bash

CopyEdit

kubectl rollout undo deployment raman-dep-yml -n raman --to-revision=1

**Rollback to revision 2 (if needed):**

bash

CopyEdit

kubectl rollout undo deployment raman-dep-yml -n raman --to-revision=2

Check the deployment and pod image:

bash

CopyEdit

kubectl get pods -n raman

kubectl describe pod <pod-name> -n raman | grep -i image

**Step 5: Clean Up (Optional)**

bash

CopyEdit

kubectl delete namespace raman

**Explanation of Key Concepts**

**1. Deployment Strategy**

* Default: RollingUpdate – old pods are gradually replaced by new ones.
* Ensures zero downtime (if app supports it).

**2. ReplicaSet**

* Behind every Deployment revision is a ReplicaSet.
* Manages the desired number of pod replicas for that version.

**3. --record=true**

* Adds the command executed in .metadata.annotations["kubectl.kubernetes.io/change-cause"].
* Useful for rollback visibility.

**4. Rollout History**

* Kubernetes maintains revision history of deployments.
* You can audit changes and roll back to previous states.

**5. Rollback**

* Reverts to a previous ReplicaSet managed by the deployment.
* Helps in quick recovery from broken deployments.

**Lab Guide: Kubernetes Services – ClusterIP, NodePort, LoadBalancer**

**Objective**

To understand and demonstrate how the three main types of Kubernetes Services expose and route traffic to applications:

* ClusterIP: Internal cluster-only access.
* NodePort: External access via any node’s IP and a fixed port.
* LoadBalancer: External access via a public cloud load balancer.

**Environment Setup**

1. Kubernetes cluster with at least 2 worker nodes.
2. kubectl CLI configured.
3. Namespace: raman

bash

CopyEdit

kubectl create namespace raman

**Step 1: Deploy a Demo Application**

We'll use nginxdemos/hello for visual output when accessed.

bash

CopyEdit

kubectl create deployment ramandep --image=nginxdemos/hello --replicas=5 -n raman

Verify pods:

bash

CopyEdit

kubectl get pods -n raman -o wide

Check internal IPs (useful for manual testing later):

bash

CopyEdit

kubectl get pods -n raman -o wide

Example output:

plaintext

CopyEdit

ramandep-xxxxxxx 192.168.x.x w1

ramandep-yyyyyyy 192.168.x.x w2

**Step 2: Create Services**

**🔹 1. ClusterIP (default)**

* **Internal access only** within the cluster.
* Use case: communication between services (e.g., frontend → backend).

bash

CopyEdit

kubectl expose deployment ramandep \

--name=raman-clusterip \

--port=80 \

--target-port=80 \

--type=ClusterIP \

-n raman

Verify:

bash

CopyEdit

kubectl get svc -n raman

kubectl get endpoints -n raman

Test from within cluster (if you have access to a pod/shell):

bash

CopyEdit

kubectl run testpod --rm -it --image=busybox --restart=Never -n raman -- sh

wget -qO- http://raman-clusterip

**🔸 2. NodePort**

* **Exposes service on a static port (30000-32767)** on each node.
* Can be accessed via <NodeIP>:<NodePort> from outside the cluster.
* Use case: development, testing.

bash

CopyEdit

kubectl expose deployment ramandep \

--name=raman-nodeport \

--port=80 \

--target-port=80 \

--type=NodePort \

-n raman

Check:

bash

CopyEdit

kubectl get svc -n raman

Sample output:

plaintext

CopyEdit

NAME TYPE CLUSTER-IP PORT(S) NODEPORT

raman-nodeport NodePort 10.108.112.65 80:30128/TCP 30128

Access externally:

bash

CopyEdit

curl http://<AnyNodeIP>:30128

Example:

bash

CopyEdit

curl http://35.87.236.3:30128

**🔶 3. LoadBalancer**

* **Allocates a cloud provider load balancer** and assigns a public IP.
* Use case: production-grade external exposure.
* Works only if your cluster is on a cloud platform (e.g., EKS, GKE, AKS).

bash

CopyEdit

kubectl expose deployment ramandep \

--name=raman-lb \

--port=80 \

--target-port=80 \

--type=LoadBalancer \

-n raman

Check:

bash

CopyEdit

kubectl get svc -n raman

Wait a few seconds for external IP to appear:

plaintext

CopyEdit

raman-lb LoadBalancer 10.0.42.21 <external-ip> 80:xxxx/TCP

Test it:

bash

CopyEdit

curl http://<external-ip>

**Service Comparison Table**

| **Service Type** | **Cluster Access** | **External Access** | **Typical Use Case** |
| --- | --- | --- | --- |
| ClusterIP | ✅ | ❌ | Microservice communication |
| NodePort | ✅ | ✅ (via NodeIP:Port) | Dev/testing environments |
| LoadBalancer | ✅ | ✅ (via LB IP) | Production & public apps |

**Step 3: Clean Up**

To remove everything after testing:

bash

CopyEdit

kubectl delete svc --all -n raman

kubectl delete deploy --all -n raman

**Bonus: Validate Endpoints**

Check if service endpoints are populated correctly:

bash

CopyEdit

kubectl get endpoints -n raman

Check which pod IPs are backing the service.

**Real-Time Troubleshooting Tips**

* Use kubectl describe svc <svc-name> -n raman to inspect service config.
* Check logs: kubectl logs <pod-name> -n raman
* Run a busybox test pod and use wget, nslookup, or telnet to test DNS and connectivity.

**🧪 Lab Guide: Taints and Tolerations in Kubernetes**

**✅ Objective**

Understand and demonstrate the use of **Taints** and **Tolerations** to control pod scheduling on Kubernetes nodes.

**📘 Conceptual Overview**

**🔹 Taint**

A *taint* is applied to a node to **repel** pods from being scheduled onto it unless those pods explicitly “tolerate” the taint.

**Syntax:**

bash

CopyEdit

kubectl taint nodes <node-name> <key>=<value>:<effect>

**Taint Effects:**

| **Effect** | **Description** |
| --- | --- |
| NoSchedule | Do not schedule pods that don’t tolerate the taint. |
| PreferNoSchedule | Avoid scheduling if possible, but not mandatory. |
| NoExecute | Existing pods will be evicted unless they tolerate. |

**🔸 Toleration**

A **toleration** is applied to a pod and allows it to be scheduled on nodes **with matching taints**.

**In Pod Spec YAML:**

yaml

CopyEdit

tolerations:

- key: "key2"

operator: "Equal"

value: "value2"

effect: "NoSchedule"

**🔧 Lab Setup**

**1️⃣ Taint Nodes**

bash

CopyEdit

kubectl taint nodes w1 key1=value1:NoSchedule

kubectl taint nodes w2 key2=value2:NoSchedule

**Verify Taints:**

bash

CopyEdit

kubectl describe nodes | grep -i taints

Expected output:

makefile

CopyEdit

Taints: key1=value1:NoSchedule

key2=value2:NoSchedule

**2️⃣ Deploy Without Toleration (For Failure Case)**

bash

CopyEdit

kubectl create deployment ramandep --image=httpd --replicas=5 -n raman

**Observe:**

bash

CopyEdit

kubectl get pods -n raman -o wide

You should see Pending status since the pods can’t tolerate any taints.

**3️⃣ Apply Toleration via YAML (Success Case)**

**deploy2.yml — Your Provided File**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-yml2

labels:

purpose: training

namespace: raman

spec:

replicas: 5

selector:

matchLabels:

name: myapp

template:

metadata:

labels:

name: myapp

spec:

containers:

- name: nginx

image: httpd

ports:

- containerPort: 80

tolerations:

- key: "key2"

operator: "Equal"

value: "value2"

effect: "NoSchedule"

**Apply:**

bash

CopyEdit

kubectl apply -f deploy2.yml

**Verify:**

bash

CopyEdit

kubectl get pods -n raman -o wide

Expected:

* Pods will now be scheduled on **node w2**, which has the key2=value2:NoSchedule taint.
* They **will not** get scheduled on w1 because toleration only matches key2.

**🧪 Test Case: Scaling Replicas**

bash

CopyEdit

kubectl scale deployment raman-dep-yml2 --replicas=15 -n raman

kubectl get pods -n raman -o wide

✅ This will test the **scheduling pressure** and ensure excess pods are only scheduled on tolerable nodes.

**🧹 Clean-Up**

To remove the taints:

bash

CopyEdit

kubectl taint nodes w1 key1=value1:NoSchedule-

kubectl taint nodes w2 key2=value2:NoSchedule-

To delete deployments:

bash

CopyEdit

kubectl delete deploy --all -n raman

**📋 Summary Table**

| **Node** | **Taint Applied** | **Pods Scheduled (Without Toleration)** | **With Toleration Matching** |
| --- | --- | --- | --- |
| w1 | key1=value1:NoSchedule | ❌ | ✅ if matching toleration |
| w2 | key2=value2:NoSchedule | ❌ | ✅ (as in deploy2.yml) |

**🧪 Lab Guide: Deploying Fluentd as a DaemonSet with Tolerations**

**✅ Objective**

* Deploy fluentd using a **DaemonSet** in the raman namespace.
* Ensure it runs on all nodes, including control-plane/master nodes (which are usually tainted).
* Demonstrate volume mounts, tolerations, and node targeting.

**📘 Conceptual Overview**

**🔹 DaemonSet**

A DaemonSet ensures **one pod runs per node** in the cluster. Useful for:

* Log collectors (like Fluentd, Logstash)
* Node-level monitoring agents (like Prometheus Node Exporter)
* Security scanners

**🔸 Tolerations in DaemonSet**

Control-plane/master nodes are tainted to avoid regular workloads:

bash

CopyEdit

node-role.kubernetes.io/master:NoSchedule

node-role.kubernetes.io/control-plane:NoSchedule

To allow your DaemonSet to run on these nodes, tolerations must be explicitly defined.

**📄 DaemonSet YAML (daemon.yml)**

yaml

CopyEdit

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd-elasticsearch

namespace: raman

labels:

k8s-app: fluentd-logging

spec:

selector:

matchLabels:

name: fluentd-elasticsearch

template:

metadata:

labels:

name: fluentd-elasticsearch

spec:

containers:

- name: fluentd-elasticsearch

image: fluent/fluentd

resources:

limits:

memory: 200Mi

requests:

cpu: 100m

memory: 200Mi

volumeMounts:

- name: rk

mountPath: /opt

volumes:

- name: rk

hostPath:

path: /home/ubuntu

tolerations:

- key: "node-role.kubernetes.io/control-plane"

operator: "Exists"

effect: "NoSchedule"

- key: "node-role.kubernetes.io/master"

operator: "Exists"

effect: "NoSchedule"

terminationGracePeriodSeconds: 30

**🛠️ Step-by-Step Lab Instructions**

**1️⃣ Verify Node Taints**

bash

CopyEdit

kubectl describe nodes | grep -i taints

Ensure control-plane/master nodes have the appropriate taints like:

text

CopyEdit

Taints: node-role.kubernetes.io/master:NoSchedule

**2️⃣ Apply the DaemonSet**

bash

CopyEdit

kubectl create ns raman

kubectl apply -f daemon.yml

**3️⃣ Verify Pods**

bash

CopyEdit

kubectl get pods -n raman -o wide

You should see:

* One pod per node (including control-plane).
* Pod names prefixed like: fluentd-elasticsearch-xxxxx

**4️⃣ Inspect Pod Details**

bash

CopyEdit

kubectl describe pod <pod-name> -n raman

Check:

* Node where the pod is running
* Mounted volume /home/ubuntu is mapped to /opt

**5️⃣ Shell into a Pod**

bash

CopyEdit

kubectl exec -it <fluentd-pod-name> -n raman -- /bin/bash

Inside the container, check:

bash

CopyEdit

ls /opt

Expected: Files from the host path /home/ubuntu should be visible.

**6️⃣ Modify & Reapply DaemonSet**

Edit and reapply if changes are needed:

bash

CopyEdit

vi daemon.yml

kubectl apply -f daemon.yml

To delete and recreate:

bash

CopyEdit

kubectl delete -f daemon.yml

kubectl apply -f daemon.yml

**🧹 Cleanup (Optional)**

bash

CopyEdit

kubectl delete -f daemon.yml

kubectl delete ns raman

**📋 Summary Table**

| **Feature** | **Explanation** |
| --- | --- |
| **DaemonSet** | Ensures one pod per node |
| **Fluentd Image** | Collects logs |
| **Volume Mount** | /home/ubuntu on host mounted to /opt |
| **Tolerations** | Allows scheduling on master/control-plane |
| **Resources** | CPU/Memory limits/requests defined |

**🧠 Bonus Tips**

* You can add nodeSelector to limit DaemonSet only to certain node pools.
* Consider integrating this with **Elasticsearch + Kibana** for a full EFK stack.
* You can also use hostPID: true and hostNetwork: true if collecting system-level logs.

**🎯 Objective**

* Deploy **two versions** of an application: "blue" (current) and "green" (new).
* Use a **Kubernetes Service** to shift traffic between them with **zero downtime**.
* Implement manual cutover between versions by updating the service selector.

**💡 What is Blue-Green Deployment?**

Blue-Green Deployment is a release technique that reduces downtime and risk by having **two identical production environments**:

* **Blue** is the current live deployment.
* **Green** is the new version to be deployed and tested.

Once verified, you **switch traffic** from Blue to Green by changing the service’s selector.

**📁 Lab Directory Structure**

Make sure your YAML files are in place:

lua

CopyEdit

.

├── deploy.yml --> Blue deployment (nginx)

├── deploy-green.yml --> Green deployment (httpd)

├── service.yml --> NodePort service pointing to 'app: new'

**🛠️ Step-by-Step Lab Guide**

**1️⃣ Create Namespace**

bash

CopyEdit

kubectl create ns raman

**2️⃣ Deploy Blue (Current Version)**

yaml

CopyEdit

# deploy.yml (Blue)

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-v1-nginx

namespace: raman

spec:

replicas: 5

selector:

matchLabels:

app: old

template:

metadata:

labels:

app: old

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

bash

CopyEdit

kubectl apply -f deploy.yml

kubectl get pods -n raman -l app=old

Verify:

bash

CopyEdit

kubectl get pods -n raman -o wide

**3️⃣ Deploy Green (New Version)**

yaml

CopyEdit

# deploy-green.yml (Green)

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-v2-httpd

namespace: raman

spec:

replicas: 5

selector:

matchLabels:

app: new

template:

metadata:

labels:

app: new

spec:

containers:

- name: httpd

image: httpd

ports:

- containerPort: 80

bash

CopyEdit

kubectl apply -f deploy-green.yml

kubectl get pods -n raman -l app=new

Check:

bash

CopyEdit

kubectl get pods -n raman -o wide

**4️⃣ Create Service Pointing to Green**

yaml

CopyEdit

# service.yml

apiVersion: v1

kind: Service

metadata:

name: blue-green-svc

namespace: raman

spec:

type: NodePort

selector:

app: new # Pointing to GREEN version

ports:

- port: 80

targetPort: 80

nodePort: 30007

bash

CopyEdit

kubectl apply -f service.yml

kubectl get svc -n raman

Access the app using:

bash

CopyEdit

curl <NodeIP>:30007

**5️⃣ Blue-Green Switch (Manual Cutover)**

Once Green is tested successfully, if you want to switch back to Blue, **change the selector** in the service:

yaml

CopyEdit

# Update service.yml

spec:

selector:

app: old # Now pointing to BLUE

Then reapply:

bash

CopyEdit

kubectl apply -f service.yml

This **re-routes all external traffic** to the old version (nginx).

**6️⃣ Clean-up (Optional)**

bash

CopyEdit

kubectl delete -f deploy.yml

kubectl delete -f deploy-green.yml

kubectl delete -f service.yml

kubectl delete ns raman

**📊 Summary Table**

| **Component** | **Version** | **Label** | **Selector (for Service)** |
| --- | --- | --- | --- |
| Deployment v1 | Blue | app: old | Initially unused |
| Deployment v2 | Green | app: new | app: new (in Service) |
| Service | Traffic router | N/A | app: new initially |

**🔁 Rolling vs Blue-Green Comparison**

| **Feature** | **Rolling Update** | **Blue-Green** |
| --- | --- | --- |
| Downtime | Minimal | Zero |
| Resource Usage | Optimized | High (both versions run) |
| Rollback Strategy | Slow (rolling back) | Fast (flip service selector) |
| Complexity | Low | Medium |

**💡 What is Canary Deployment?**

Canary deployment is a progressive release strategy where a small subset of users receives the new version of the application first (called a *canary*), while the majority continues using the current version.

Advantages:

* Safer than blue-green: detect bugs early before full rollout.
* Easily rollback by scaling down the canary.
* Fine-grained traffic control via pod replicas or service meshes (e.g., Istio).

**🏗️ Architecture Overview**

text

CopyEdit

+--------------------------+

| NodePort Service |

| app: same (all pods) |

+------------+-------------+

|

+----------------------+-----------------------+

| |

+------------+ +---------------+

| Deployment | (nginx, 3 pods) | Deployment | (httpd, 2 pods)

| v1 | app: same | v2 | app: same

+------------+ +---------------+

**📁 Files Required**

lua

CopyEdit

.

├── deploy.yml --> Base version (nginx)

├── deploy-green.yml --> Canary version (httpd)

├── service.yml --> Service with selector `app: same`

**🛠️ Step-by-Step Lab Guide**

**1️⃣ Create Namespace**

bash

CopyEdit

kubectl create ns raman

**2️⃣ Deploy Version 1 (Base - nginx)**

yaml

CopyEdit

# deploy.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-v1-nginx

namespace: raman

spec:

replicas: 5

selector:

matchLabels:

app: same

template:

metadata:

labels:

app: same

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

bash

CopyEdit

kubectl apply -f deploy.yml

kubectl get pods -n raman -l app=same

**3️⃣ Create Service (Common to both)**

yaml

CopyEdit

# service.yml

apiVersion: v1

kind: Service

metadata:

name: blue-green-svc

namespace: raman

spec:

type: NodePort

selector:

app: same

ports:

- port: 80

targetPort: 80

nodePort: 30007

bash

CopyEdit

kubectl apply -f service.yml

kubectl get svc -n raman

Check access:

bash

CopyEdit

curl <NodeIP>:30007

**4️⃣ Deploy Version 2 (Canary - httpd)**

yaml

CopyEdit

# deploy-green.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-v2-httpd

namespace: raman

spec:

replicas: 0

selector:

matchLabels:

app: same

template:

metadata:

labels:

app: same

spec:

containers:

- name: httpd

image: httpd

ports:

- containerPort: 80

Apply and verify:

bash

CopyEdit

kubectl apply -f deploy-green.yml

kubectl get deployments -n raman

Initially, this deployment is scaled to 0 replicas.

**5️⃣ Canary Release Rollout**

Gradually increase the number of replicas of version 2 (httpd) while monitoring the system:

bash

CopyEdit

kubectl scale deploy raman-dep-v2-httpd --replicas=1 -n raman

Monitor traffic distribution via logs:

bash

CopyEdit

kubectl get pods -n raman -o wide

kubectl logs <pod-name> -n raman

Gradually increase:

bash

CopyEdit

kubectl scale deploy raman-dep-v2-httpd --replicas=2 -n raman

kubectl scale deploy raman-dep-v1-nginx --replicas=3 -n raman

Repeat until you’ve transitioned fully to the new version.

**6️⃣ Full Rollout or Rollback**

* ✅ **Full rollout (upgrade):**

bash

CopyEdit

kubectl scale deploy raman-dep-v1-nginx --replicas=0 -n raman

kubectl scale deploy raman-dep-v2-httpd --replicas=5 -n raman

* ❌ **Rollback (revert):**

bash

CopyEdit

kubectl scale deploy raman-dep-v2-httpd --replicas=0 -n raman

kubectl scale deploy raman-dep-v1-nginx --replicas=5 -n raman

**7️⃣ Clean-up (Optional)**

bash

CopyEdit

kubectl delete -f deploy.yml

kubectl delete -f deploy-green.yml

kubectl delete -f service.yml

kubectl delete ns raman

**📊 Summary Table**

| **Component** | **Version** | **Label** | **Replicas** | **Purpose** |
| --- | --- | --- | --- | --- |
| Deployment V1 | nginx | app: same | 5 | Base (current) |
| Deployment V2 | httpd | app: same | 0 → 5 | Canary/Upgrade |
| Service | NodePort | Selects app: same | — | Routes traffic |

**📌 Tips & Best Practices**

* Use readinessProbes to avoid routing traffic to unhealthy canary pods.
* Use kubectl rollout status to monitor progress.
* Use HPA or service meshes (e.g., Istio) for automated traffic shifting.
* Add annotations or labels for versioning/debugging:

yaml

CopyEdit

labels:

version: v1

app: same