**🧪 Kubernetes Lab Guide: Pod and Node Inspection Using kubectl**

**🎯 Objective**

To understand how to inspect the state of Kubernetes pods and nodes across namespaces using kubectl, and interpret the output to verify cluster initialization and workload distribution.

**📘 Prerequisites**

* A working Kubernetes cluster (e.g., local minikube, kind, or cloud-managed cluster).
* kubectl installed and configured (kubectl config view should show the correct context).
* Basic understanding of Kubernetes concepts: **pods**, **nodes**, **namespaces**, **control plane**, and **system components**.

**🔧 Step-by-Step Breakdown of Commands**

**1. List All Pods in All Namespaces**

bash

CopyEdit

kubectl get pods -A

* get pods: Lists running pods.
* -A or --all-namespaces: Retrieves pods across **all namespaces** (not just default).
* 🔍 Use this to verify that **system components** (like kube-dns, coredns, kube-proxy, etc.) are running correctly.

**2. Clear the Terminal**

bash

CopyEdit

clear

* Clears terminal output to reduce clutter between commands.

**3. List Pods in Current Namespace**

bash

CopyEdit

kubectl get pods

* Lists pods **only in the current namespace** (typically default unless changed).
* Use kubectl config view --minify | grep namespace to view current namespace.

**4. Detailed Pod Listing Across All Namespaces**

bash

CopyEdit

kubectl get pods -A -o wide

* -o wide: Adds extra fields to the output like NODE, IP, RESTARTS, etc.
* ✅ Useful for debugging **pod-to-node placement**, **IP allocation**, and network diagnostics.

**5. Check Cluster Initialization File (If Any)**

bash

CopyEdit

cat cluster\_initialized.txt

* This step suggests there may be a script or output file from a cluster setup process.
* 📁 This file can be used to store flags or logs after kubeadm init, kind create cluster, or custom provisioning scripts.

**6. List Cluster Nodes**

bash

CopyEdit

kubectl get nodes

* Shows the status of nodes in the cluster.
* Columns:
  + NAME: Node hostname.
  + STATUS: Ready, NotReady, etc.
  + ROLES: master or control-plane, worker.
  + AGE, VERSION: Useful for debugging version mismatches or provisioning age.
* 🔎 If a node is in NotReady, investigate kubelet, container runtime, and network issues.

**7. Attempt to Get "npods" (Typo)**

bash

CopyEdit

kubectl get npods -A

* ❌ npods is not a valid resource type.
* Possibly a **typo** — user likely meant pods.

**✅ Correct Command:**

bash

CopyEdit

kubectl get pods -A

**8. List Pods in kube-system Namespace with Node Information**

bash

CopyEdit

kubectl get pods -n kube-system -o wide

* Targets kube-system, the namespace where core components run.
* These include:
  + kube-apiserver
  + etcd
  + kube-controller-manager
  + kube-scheduler
  + coredns
  + kube-proxy
* -o wide: Shows which node the system pods are running on.
* 💡 This is useful for validating **control plane health**.

**🧠 Summary of Key Concepts**

| **Command** | **Purpose** |
| --- | --- |
| kubectl get pods -A | Lists all pods in the cluster |
| kubectl get pods -n kube-system | Checks system-level pods |
| kubectl get pods -A -o wide | Adds node and IP info for pod placement inspection |
| kubectl get nodes | Shows all nodes and their readiness |
| clear | Clears terminal |
| cat cluster\_initialized.txt | View file containing initialization status/logs |
| Typos like kubectl get npods -A | Will result in "unknown resource" errors |

**🔍 Additional Tips**

**Set default namespace:**

bash

CopyEdit

kubectl config set-context --current --namespace=<namespace-name>

**Describe a pod for more detailed info:**

bash

CopyEdit

kubectl describe pod <pod-name> -n <namespace>

**View logs for a specific pod:**

bash

CopyEdit

kubectl logs <pod-name> -n <namespace>

**Get cluster info:**

bash

CopyEdit

kubectl cluster-info

**✅ Lab Checklist**

* List system pods in all namespaces.
* Inspect node readiness.
* Verify cluster control plane components.
* Understand pod placement across nodes.
* Identify and fix typos in commands.
* Understand purpose of -o wide and -A flags.

**🧪 Full Lab Guide: Docker to Kubernetes Deployment (Namespace + Pod Lifecycle)**

**🏗️ Lab Structure**

vbnet

CopyEdit

Code → Dockerfile → docker build → Custom Image → Deploy on Kubernetes → Pod Management → Namespace Isolation

**📋 Prerequisites**

* Working Kubernetes Cluster (minikube, kind, or cloud-hosted).
* Docker installed on your node/worker machine.
* kubectl CLI configured.
* Basic Dockerfile (can use sample httpd if not custom-built).
* Your shell is configured with the alias k=kubectl.

**1️⃣ Understanding the kubectl and Docker Setup**

**Alias Setup:**

bash

CopyEdit

alias k=kubectl

* This shortens commands (e.g., k get pods).

**Checking Nodes and Cluster Resources**

bash

CopyEdit

k get nodes

* Lists cluster nodes (e.g., master/worker).

bash

CopyEdit

k get pods -A -o wide

* All pods, in all namespaces, with node assignment, pod IP, and more.

**2️⃣ Create a Kubernetes Namespace**

**Create Namespace named raman**

bash

CopyEdit

k create ns raman

**Verify Namespace Creation:**

bash

CopyEdit

k get ns

Namespaces help isolate workloads logically.

**3️⃣ Pod Deployment Testing**

**✅ Deploy httpd Pod in Default Namespace (initial test)**

bash

CopyEdit

k run ramanpod --image=httpd

* Creates a pod named ramanpod using the httpd (Apache server) image.
* **Default namespace** is used.

**View the Pod**

bash

CopyEdit

k get pods

k get pods -o wide

**❌ Delete the Pod (Cleanup)**

bash

CopyEdit

k delete pod ramanpod

**4️⃣ Deploy in Custom Namespace raman**

bash

CopyEdit

k run ramanapp --image=httpd -n raman

* Same pod as before, but isolated in raman namespace.

**Check the Pod**

bash

CopyEdit

k get pods -n raman

k get pods -n raman -o wide

* Use -n raman to scope the command to the correct namespace.

**❌ Delete Pod from Namespace raman**

bash

CopyEdit

k delete pod ramanapp -n raman

**5️⃣ Docker Side: Inspect on Worker Node**

These commands are executed **on the worker node** (e.g., via SSH into the node):

**View Running Containers**

bash

CopyEdit

docker ps

* Helps map which containers are running under the hood.
* Kubernetes will use containers for each pod (often 1:1 mapping, unless multiple containers per pod).

**Force Remove a Container (e.g., by ID)**

bash

CopyEdit

docker rm -f 6820f266e008

* Forcefully stops and removes the container.
* Note: Be cautious, Kubernetes will recreate pods based on controllers like Deployments/ReplicaSets if they exist.

**6️⃣ Bonus: Custom Image (Optional Advanced Step)**

If you're going from **source code** to Docker to Kubernetes, the flow looks like:

**📁 Sample Dockerfile**

Dockerfile

CopyEdit

FROM httpd:2.4

COPY ./index.html /usr/local/apache2/htdocs/

**Build Docker Image:**

bash

CopyEdit

docker build -t custom-httpd:v1 .

**Load to Kubernetes Cluster:**

* If using minikube:

bash

CopyEdit

minikube image load custom-httpd:v1

* If remote nodes, push to registry like DockerHub:

bash

CopyEdit

docker tag custom-httpd:v1 your-dockerhub-username/custom-httpd:v1

docker push your-dockerhub-username/custom-httpd:v1

Then use in Kubernetes:

bash

CopyEdit

k run mycustompod --image=your-dockerhub-username/custom-httpd:v1 -n raman

**✅ Summary**

| **Step** | **Command** | **Description** |
| --- | --- | --- |
| Alias | alias k=kubectl | Set shorthand alias |
| Create Namespace | k create ns raman | Logical isolation |
| Run Pod | k run <name> --image <image> | Create pod from image |
| Delete Pod | k delete pod <name> | Cleanup resource |
| View Resources | k get pods -A -o wide | Deep visibility |
| Docker Check | docker ps, docker rm -f | Worker-level container control |

**🧠 Final Notes**

* Pods not managed by controllers (e.g., k run) are ephemeral — if deleted or the node crashes, they don’t auto-recover.
* For production, use **Deployments**.
* Namespaces are great for multitenancy or environment isolation (dev/stage/prod).
* Container management on worker nodes should be used **only for debugging**, not for direct app lifecycle.

**🧪 Kubernetes Lab Guide: Deploying a Pod using YAML Manifest in a Custom Namespace**

**🧰 Lab Objectives**

* Create and edit a Pod YAML file.
* Deploy the pod into a custom namespace (raman).
* Explore Docker image storage on the host machine.
* Understand pod lifecycle management using declarative (.yml) configuration.

**🔧 Prerequisites**

* Running Kubernetes cluster (master + worker setup).
* Docker installed on worker/master.
* Shell access to master node.
* kubectl CLI set up and alias k=kubectl.

**🔹 Step 1: Check System Status**

bash

CopyEdit

df -h

* Check disk usage before deploying resources.

**🔹 Step 2: Explore/Create Filesystem & Prepare Workspace**

bash

CopyEdit

ls

vi pod.yml

* Edit the pod.yml manifest using vi.

**✅ Sample pod.yml**

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: raman-nginx

namespace: raman # 👈 Ensures pod is deployed in the right namespace

spec:

containers:

- name: nginx-con

image: nginx:1.14.2

ports:

- containerPort: 80

* This YAML defines:
  + A pod named raman-nginx
  + Running in the raman namespace
  + Using official NGINX image (nginx:1.14.2)
  + Exposing container port 80

**🔹 Step 3: Check Namespace and Create Pod**

**Confirm Namespace:**

bash

CopyEdit

k get ns

**If not present, create namespace:**

bash

CopyEdit

k create ns raman

**Apply the Manifest:**

bash

CopyEdit

k create -f pod.yml

* Deploys pod in raman namespace as specified in the YAML.

**🔹 Step 4: Inspect the Pod**

bash

CopyEdit

k get pods -n raman

k describe pod raman-nginx -n raman

k get pods -n raman -o wide

* describe provides events, IP, node, container status, etc.
* -o wide shows the assigned node, IPs, and images.

**🔹 Step 5: Pod Lifecycle Management**

**Delete Using Manifest**

bash

CopyEdit

k delete -f pod.yml

**Reapply After Modification**

1. Move file to a new directory:

bash

CopyEdit

mkdir raman

mv pod.yml raman/

cd raman/

1. Recreate:

bash

CopyEdit

k create -f pod.yml

1. Confirm:

bash

CopyEdit

k get pods -n raman -o wide

**🔹 Step 6: Inspect Docker Images and Storage (On Node)**

**View Images:**

bash

CopyEdit

docker images

**Explore Docker Image Layers:**

bash

CopyEdit

cd /var/lib/docker/image/overlay2/

ls

cat repositories.json

* repositories.json maps image tags to SHA digests.
* You’ll see entries like:

json

CopyEdit

{

"nginx": {

"nginx:1.14.2": "sha256:..."

}

}

* Helps trace which images are cached locally.

**🧠 Deep Dive: Why overlay2?**

* **overlay2** is the default Docker **storage driver** on most modern Linux distributions.
* It provides **layered filesystem** support (copy-on-write), making images lightweight and composable.

**✅ Summary of Commands Used**

| **Command** | **Purpose** |
| --- | --- |
| k get pods -n raman | View pods in the raman namespace |
| k describe pod <name> -n raman | View detailed pod info |
| k create -f pod.yml | Create resources defined in manifest |
| k delete -f pod.yml | Delete pod defined in manifest |
| docker images | Check locally available Docker images |
| cat repositories.json | Inspect how Docker stores image references |
| cd /var/lib/docker/overlay2 | Navigate Docker's image layer storage |

**🧪 Kubernetes Lab Guide: Multi-Container Pod in a Custom Namespace (raman)**

**🎯 Lab Objectives**

* Deploy a multi-container pod in Kubernetes.
* Interact with individual containers inside the pod.
* Observe logs and perform runtime actions inside containers.
* Understand multi-container patterns like sidecars and init-containers (briefly touched).

**🧰 Pre-requisites**

* Kubernetes cluster running (minikube, kind, kubeadm cluster, etc.)
* Docker installed on worker/master nodes.
* Access to terminal with kubectl installed.
* Namespace raman already created.

**📁 Project File Structure**

bash

CopyEdit

~/raman/

└── multicon.yml # Multi-container pod spec

**🔹 Step 1: Understand the Pod YAML**

**File: multicon.yml**

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: raman-multicon-pod

namespace: raman # Pod will be deployed in this custom namespace

spec:

containers:

- name: con1

image: nginx:1.14.2

ports:

- containerPort: 80

- name: con2

image: redis

ports:

- containerPort: 6379

# Optional third container was commented out

# - name: con3

# image: httpd

# ports:

# - containerPort: 80

**🔍 What this YAML defines:**

* A pod named raman-multicon-pod.
* Deployed in the raman namespace.
* It contains two containers:
  + con1 running nginx web server on port 80.
  + con2 running redis in-memory database on port 6379.
* Containers **share the same network namespace** and can talk via localhost.

**🔹 Step 2: Deploy the Pod**

bash

CopyEdit

k apply -f multicon.yml

**Verify Deployment**

bash

CopyEdit

k get pods -n raman -o wide

Check logs to confirm both containers started properly:

bash

CopyEdit

k logs raman-multicon-pod -c con1 -n raman

k logs raman-multicon-pod -c con2 -n raman

You can also describe the pod to see container lifecycle events:

bash

CopyEdit

k describe pod raman-multicon-pod -n raman

**🔹 Step 3: Interact with Containers**

**Enter con1 (nginx)**

bash

CopyEdit

k exec -it raman-multicon-pod -c con1 -n raman -- /bin/bash

Check nginx default page:

bash

CopyEdit

curl localhost

**Enter con2 (redis)**

bash

CopyEdit

k exec -it raman-multicon-pod -c con2 -n raman -- /bin/bash

Redis doesn’t have bash in slim images, fallback to sh if bash fails:

bash

CopyEdit

k exec -it raman-multicon-pod -c con2 -n raman -- sh

Run Redis CLI:

bash

CopyEdit

redis-cli

127.0.0.1:6379> set foo bar

OK

127.0.0.1:6379> get foo

"bar"

**Test container inter-communication**

From con1, try reaching Redis (con2) via localhost:6379:

bash

CopyEdit

apt update && apt install -y netcat

nc -vz localhost 6379

**🔹 Step 4: Update or Modify Pod**

If you modify multicon.yml, use:

bash

CopyEdit

k apply -f multicon.yml # Non-destructive

Or for hard replace:

bash

CopyEdit

k replace -f multicon.yml # Deletes and recreates

To delete all pods in the namespace (cleanup):

bash

CopyEdit

k delete pod --all -n raman

**🔹 Step 5: Logs & Debugging**

Get logs from individual containers:

bash

CopyEdit

k logs raman-multicon-pod -c con1 -n raman

k logs raman-multicon-pod -c con2 -n raman

You can inspect logs to trace request handling (nginx) or connection info (redis).

**📌 Additional Concepts (Optional)**

**📦 Why Use Multi-Container Pods?**

* **Sidecar pattern**: Add helper container (e.g., logging, reverse proxy).
* **Ambassador pattern**: Network proxy container.
* **Adapter pattern**: Data transformation container.

Each container shares:

* Pod IP.
* Volume (if defined).
* Lifecycle (if one fails, pod restarts).

**📋 Summary of Useful Commands**

| **Command** | **Description** |
| --- | --- |
| k apply -f multicon.yml | Create/update the pod |
| k get pods -n raman -o wide | Check status with more info |
| k logs <pod> -c <container> | Get logs of a container |
| k exec -it <pod> -c <container> -- /bin/bash | Exec into container |
| k delete pod --all -n raman | Cleanup all pods |
| k describe pod <name> -n raman | Detailed pod info |

**🧪 Kubernetes Lab Guide: Node Labeling & Pod Scheduling via nodeSelector**

**🎯 Lab Objectives**

* Understand and manage node labels.
* Use nodeSelector in pod specifications.
* Control where Kubernetes schedules your pods.
* Observe how scheduling reacts to label changes.

**🛠 Pre-requisites**

* Kubernetes cluster with at least two worker nodes (e.g., w1 and w2)
* kubectl configured and working.
* Namespace raman already created.

**🔹 Step 1: Understand the pod.yml**

**File: pod.yml**

yaml

CopyEdit

apiVersion: v1

kind: Pod

metadata:

name: raman-nginx

namespace: raman

spec:

containers:

- name: nginx-con

image: nginx:1.14.2

ports:

- containerPort: 80

nodeSelector:

env: dev

**🔍 What This YAML Defines:**

* A pod named raman-nginx.
* Placed in namespace raman.
* Runs an NGINX container.
* Scheduled **only on nodes labeled env=dev** due to the nodeSelector.

**🔹 Step 2: Label Nodes**

To control which node runs the pod, we assign custom labels.

**✅ Label nodes:**

bash

CopyEdit

k label node w1 env=prod

k label node w2 env=dev

💡 nodeSelector is **key-value exact match**, so spelling/case must match exactly.

**🔍 Verify labels:**

bash

CopyEdit

kubectl get nodes --show-labels

You should see:

bash

CopyEdit

NAME STATUS ... LABELS

w1 Ready ... env=prod,...

w2 Ready ... env=dev,...

**🔹 Step 3: Create the Pod Using nodeSelector**

bash

CopyEdit

k apply -f pod.yml

⏳ Then verify:

bash

CopyEdit

k get pods -n raman -o wide

You should see the pod scheduled on node w2 (because of env=dev).

**🔹 Step 4: Confirm the Pod's Node Placement**

bash

CopyEdit

k describe pod raman-nginx -n raman

Look for this section in output:

makefile

CopyEdit

Node: w2/10.x.x.x

...

Node-Selectors: env=dev

If the node does **not** have the env=dev label, the pod will stay in Pending state.

**🔹 Step 5: Troubleshooting (if pod is Pending)**

**🛑 Check pod status:**

bash

CopyEdit

k get pods -n raman

If it's stuck in Pending:

**✅ Describe the pod:**

bash

CopyEdit

k describe pod raman-nginx -n raman

Check for:

bash

CopyEdit

Events:

Warning FailedScheduling ... 0/2 nodes are available: 2 node(s) didn't match node selector.

🔁 Solution: Make sure **at least one node has the required label**:

bash

CopyEdit

k label node w1 env=dev --overwrite

Then re-create or update the pod:

bash

CopyEdit

k delete pod raman-nginx -n raman

k apply -f pod.yml

**🔹 Step 6: Update/Change Labels Dynamically**

If you want to move the pod to a different node:

1. Change the nodeSelector in pod.yml to env: prod.
2. Label a different node accordingly:

bash

CopyEdit

k label node w1 env=prod --overwrite

k label node w2 env=dev --overwrite

1. Recreate the pod:

bash

CopyEdit

k delete pod raman-nginx -n raman

k apply -f pod.yml

Now the pod should run on w1.

**🔹 Step 7: Clean Up**

bash

CopyEdit

k delete pod raman-nginx -n raman

Remove node labels if needed:

bash

CopyEdit

k label node w1 env-

k label node w2 env-

**📌 Summary of Key Concepts**

| **Feature** | **Description** |
| --- | --- |
| nodeSelector | Simple key-value scheduler constraint |
| Node Labeling | Allows selective scheduling of workloads |
| kubectl get nodes --show-labels | Shows all node labels |
| kubectl label node <node> key=value | Add a label |
| key- (e.g. env-) | Removes a label |
| --overwrite | Needed to update existing labels |

**🧠 Bonus: When to Use nodeSelector**

| **Use Case** | **Why?** |
| --- | --- |
| Dev/Test/Prod separation | Run pods on appropriate env-specific nodes |
| GPU / High-memory workloads | Direct pods to capability-specific nodes |
| Node-specific configurations | Example: storage-heavy apps on storage-optimized nodes |

**🧪 Kubernetes Lab Guide: Working with Deployments**

**🎯 Lab Objectives**

* Understand Kubernetes Deployments
* Create deployments using kubectl and YAML
* Observe ReplicaSets and Pods
* Scale deployments up and down
* Describe relationships between Deployment → ReplicaSet → Pod

**📦 What is a Deployment?**

A **Deployment** in Kubernetes is a higher-level controller that manages **ReplicaSets**, which in turn manage **Pods**. Deployments provide:

* Declarative updates for pods and ReplicaSets
* Rollback capability
* Scaling
* Self-healing via pod rescheduling

**🛠 Prerequisites**

* Kubernetes cluster
* kubectl configured
* Namespace raman already exists

**🔹 Step 1: Create Deployment Using CLI**

bash

CopyEdit

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

This will:

* Create a Deployment named ramandep in the raman namespace.
* Deploy 5 replicas of the httpd image (Apache HTTP server).

**🔍 Verify:**

bash

CopyEdit

kubectl get deploy -n raman

kubectl get rs -n raman

kubectl get pods -n raman

**🔹 Step 2: Delete a Pod & See Self-Healing**

bash

CopyEdit

kubectl delete pod <pod-name> -n raman

Kubernetes (via ReplicaSet) will recreate the pod automatically to maintain the desired replica count.

**🔹 Step 3: Create Deployment Using YAML**

**File: deploy.yml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-dep-yml

namespace: raman

labels:

purpose: training

spec:

replicas: 3

selector:

matchLabels:

name: myapp

template:

metadata:

labels:

name: myapp

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

This YAML:

* Creates a deployment named raman-dep-yml
* Manages 3 replicas of an NGINX container
* Selects pods based on the label name: myapp

**🚀 Apply it:**

bash

CopyEdit

kubectl create -f deploy.yml

**🔍 Check resources:**

bash

CopyEdit

kubectl get deploy -n raman

kubectl get rs -n raman

kubectl get pods -n raman

**🔹 Step 4: Explore Resource Relationships**

bash

CopyEdit

kubectl describe deploy raman-dep-yml -n raman

kubectl describe rs -n raman

kubectl describe pod <pod-name> -n raman

Look for:

* Controlled By: in pod and ReplicaSet to confirm ownership
* Selector: to see label-based selection

**🔹 Step 5: Scale Deployment**

**🧍‍♂️ Scale to 0 (delete all pods)**

bash

CopyEdit

kubectl scale deployment raman-dep-yml --replicas=0 -n raman

✅ Result: All pods are terminated, but Deployment and ReplicaSet persist.

bash

CopyEdit

kubectl get pods -n raman

Output: No resources found.

**👯‍♂️ Scale up again**

bash

CopyEdit

kubectl scale deployment raman-dep-yml --replicas=4 -n raman

bash

CopyEdit

kubectl get pods -n raman

You should now see 4 new pods created.

**🔹 Step 6: Update the Deployment (Optional)**

Modify the image or replicas in deploy.yml:

yaml

CopyEdit

...

spec:

replicas: 5

template:

spec:

containers:

- name: nginx

image: nginx:1.25.0

...

Apply changes:

bash

CopyEdit

kubectl apply -f deploy.yml

Kubernetes will **create a new ReplicaSet** and gradually roll out new pods.

**🔍 Observe rollout:**

bash

CopyEdit

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

**🔹 Step 7: Clean Up**

bash

CopyEdit

kubectl delete deploy ramandep -n raman

kubectl delete deploy raman-dep-yml -n raman

**📌 Summary of Concepts**

| **Concept** | **Description** |
| --- | --- |
| **Deployment** | Manages ReplicaSets and enables declarative updates |
| **ReplicaSet** | Ensures a stable set of running pods |
| **Pod** | Smallest deployable unit |
| kubectl scale | Adjust number of replicas dynamically |
| kubectl apply | Update deployments declaratively |
| kubectl describe | Shows detailed info about resource relationships |
| kubectl rollout | Monitor or undo rolling updates |