**Kubernetes with KOPS: Cluster, Pods, RC & RS**

📘 **What is kOps?**

kOps (short for Kubernetes Operations) is an open-source tool created by the Kubernetes community for managing Kubernetes clusters. It helps you create, maintain, and destroy Kubernetes clusters in a reproducible, declarative, and automated manner.

kOps is sometimes described as "kubectl for clusters" because it provides similar convenience, but for infrastructure-level operations.

**🧱 Why Use kOps?**

kOps is ideal for:

* Production-grade Kubernetes clusters
* Full control over cloud infrastructure (mostly AWS, GCE, etc.)
* Infrastructure as Code (IaC) with YAML
* Easy upgrades and rolling updates of clusters
* Built-in support for High Availability (HA)

**⚙️ Supported Platforms**

As of now, kOps officially supports:

* AWS (Amazon Web Services) – Primary supported platform
* GCE (Google Compute Engine)
* DigitalOcean
* OpenStack
* Bare Metal (via Terraform integration)

**🚀 Core Features of kOps**

* Easy cluster creation using command-line
* Infrastructure-as-code (YAML-based cluster definitions)
* Support for multiple cloud providers
* Integration with Terraform
* Secure cluster setup (including SSH, TLS, secrets)
* Automatic updates and rolling upgrades
* Highly configurable and extendable

**🛠️ How Does kOps Work?**

kOps works in a few steps:

1. Define your cluster configuration in YAML or using the kops create cluster command.

2. Store the configuration in a state store (usually an S3 bucket for AWS).

3. kOps generates the infrastructure using cloud APIs (e.g., EC2 instances, Auto Scaling Groups).

4. kOps installs and configures Kubernetes components on the instances.

5. Manage your cluster (rolling updates, upgrades, delete, etc.) using kops commands.

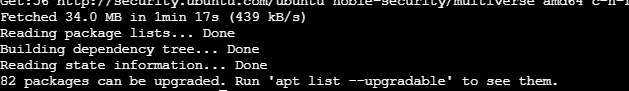
**kOps setup in aws ec2**

**✅ STEP 1: Launch EC2 Instance**

• Launch an EC2 instance of type t2.micro with 20GB SSD.

**✅ STEP 2: Install AWS CLI**

sudo apt update -y



sudo apt install unzip curl -y

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

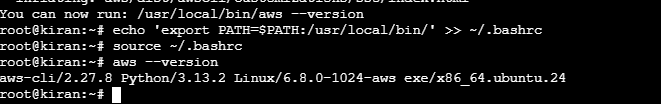
/usr/local/bin/aws --version

# Set PATH

echo 'export PATH=$PATH:/usr/local/bin/' >> ~/.bashrc

source ~/.bashrc

aws –version



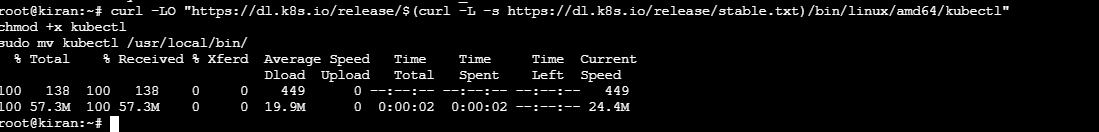
**✅ STEP 3: Install kubectl and kops**

**# Install kubectl**

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

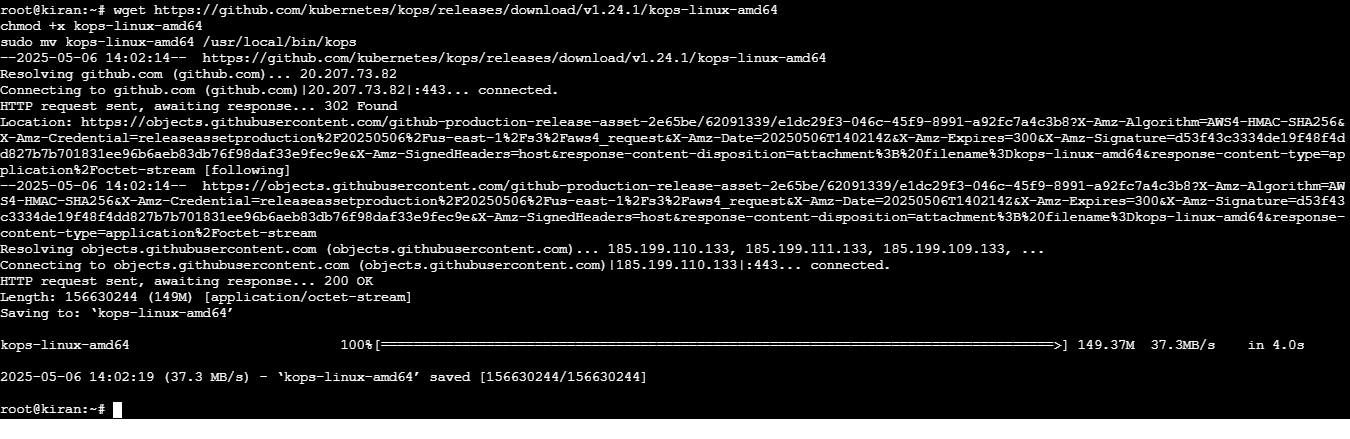


**# Install kops**

wget https://github.com/kubernetes/kops/releases/download/v1.24.1/kops-linux-amd64

chmod +x kops-linux-amd64

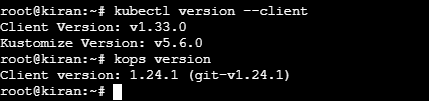
sudo mv kops-linux-amd64 /usr/local/bin/kops



**# Check versions**

kubectl version --client

kops version



**✅ STEP 4: Create IAM User with Admin Access**

1. Go to AWS Console → IAM → Users → Add user
2. Username: kops-admin
   1. Access type: Programmatic Access
3. Attach existing policies: AdministratorAccess
4. Save Access Key ID and Secret Access Key

**Configure credentials:**

**Cmd: aws configure**

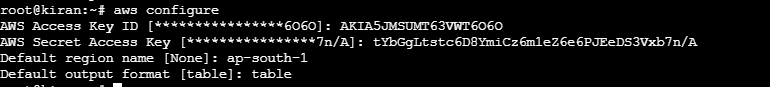
# Provide:

# AWS Access Key ID: <your-access-key-id>

# AWS Secret Access Key: <your-secret-access-key>

# Default region name: us-east-1

# Default output format: table



✅ **STEP 5: Set Up Infrastructure for KOPS**

**# Create S3 bucket**

aws s3api create-bucket \

--bucket kirannk9.k8s.local \

--region ap-south-1 \

--create-bucket-configuration LocationConstraint=ap-south-1

**# Enable versioning**

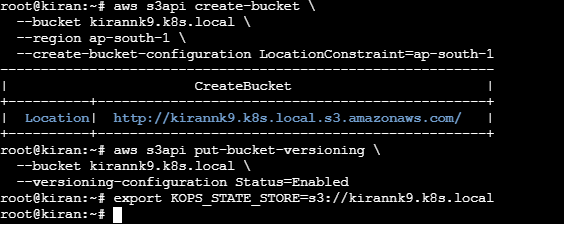
aws s3api put-bucket-versioning \

--bucket kirannk9.k8s.local \

--versioning-configuration Status=Enabled

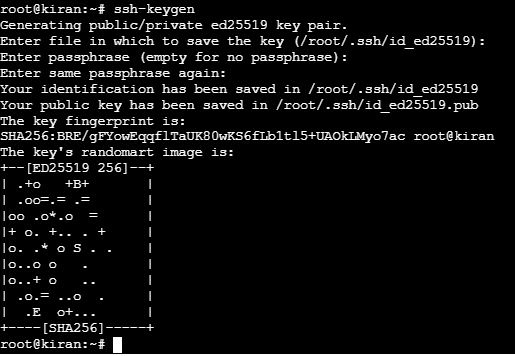
**# Export cluster state store**

export KOPS\_STATE\_STORE=s3://kirannk9.k8s.local



# Generate SSH key (optional if you already have one)

ssh-keygen



**Create kubernetes cluster**

To create a Kubernetes cluster on AWS using **kOps**, the following command was used:

kops create cluster \

--name=kiran.k8s.local \

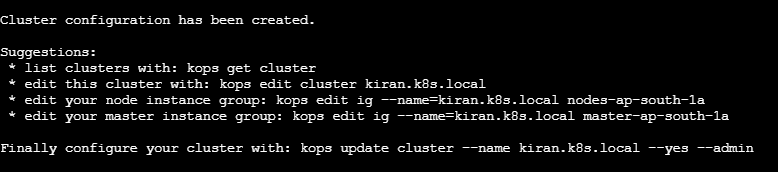
--cloud=aws \

--zones=ap-south-1a \

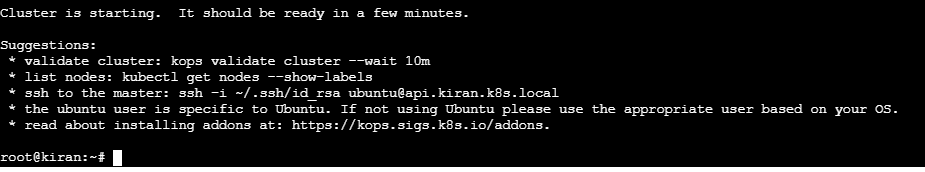
--node-count=2 \

--node-size=t2.micro \

--master-size=t2.medium

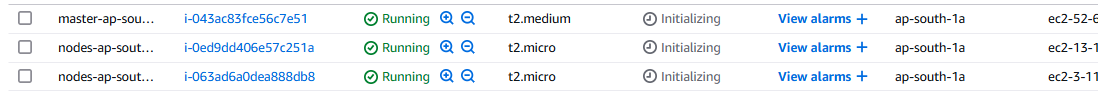


kops update cluster --name kiran.k8s.local --yes –admin

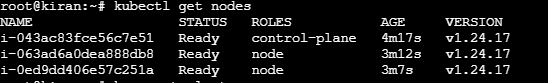


This sets up a Kubernetes cluster with:

* 2 nodes of size t2.micro
* 1 master node of size t2.medium
* Zone ap-south-1a on AWS.







**Creating a Basic Pod**

To create a simple pod with an **nginx** container, the YAML configuration is:

apiVersion: v1

kind: Pod

metadata:

name: my-first-pod

spec:

containers:

- name: my-container

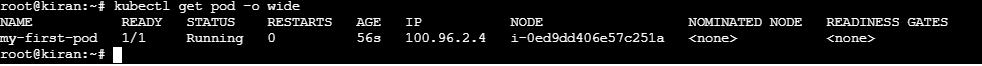
image: nginx:latest

ports:

- containerPort: 80

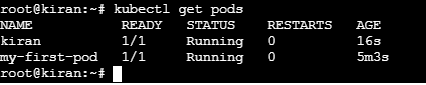
This configuration creates a pod named my-first-pod with an **nginx** container running on port 80.

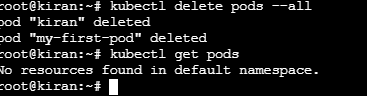




**Pod Deletion and Issues**

If a pod is deleted or encounters issues (such as network problems or heavy load), the container within the pod will also be deleted. When the container is deleted, the application stops. To address this, we need a **replication mechanism** to ensure continuous availability and scaling.



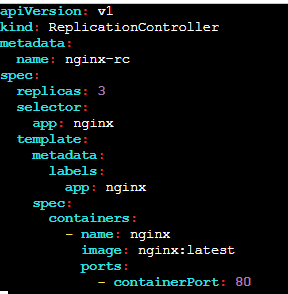


**Replication controller:**

* A Replication Controller (RC) ensures a specific number of pods are running at any given time.
* It is responsible for managing the pod lifecycle.
* It ensures that pods are always up and running.
* If there are too many pods, the RC will terminate the extra pods.
* If there are too few pods, the RC will create new ones.
* The Replication Controller has self-healing capabilities — it automatically recreates failed or deleted pods.
* If a pod fails, is terminated, or deleted, the RC will automatically create a replacement pod.
* It uses labels to identify the pods it manages.

**Using Replication Controller**

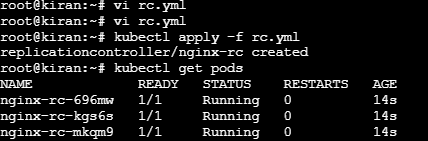
A **ReplicationController** ensures that the desired number of pod replicas are always running, even if a pod is deleted. The **ReplicationController** uses labels to match the pods it should manage.

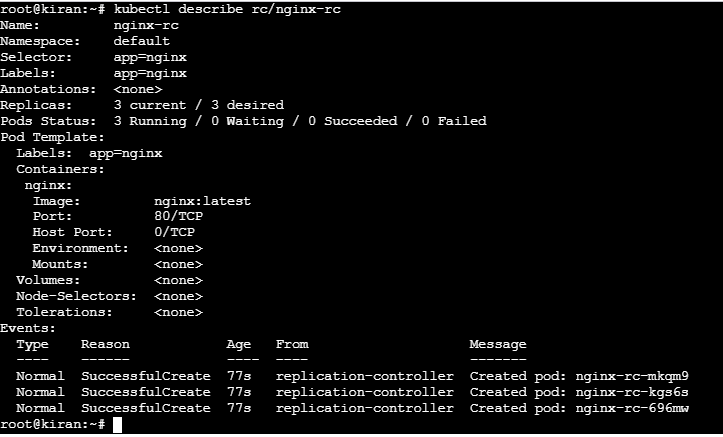
Here’s a simple YAML file for creating a **ReplicationController**:

 **replicas**: Specifies the number of pods to maintain (3 pods in this case).

 **selector**: The label selector ensures that the replication controller manages pods with the label app: nginx.

 **template**: This defines the pod template, which includes the container specifications like the image nginx:latest and the container port.

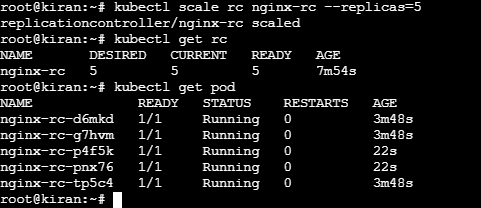




#### ****Scaling the Replication Controller****

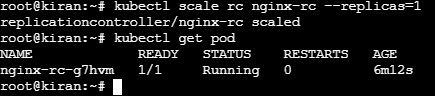
To scale up the **ReplicationController** to 5 pods, use:

kubectl scale rc nginx-rc --replicas=5



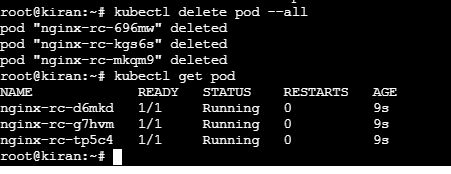
**To scale down the ReplicationController to 1 pod, use:**

kubectl scale rc nginx-rc --replicas=1



**Auto-Healing of pods**

One of the most important features of Kubernetes is its **auto-healing** capability. If a pod crashes or is deleted, Kubernetes will automatically replace the pod to ensure that the desired number of replicas is maintained.

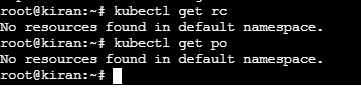


**Delete the Replication Controller**

To delete the **ReplicationController** and optionally all the pods it manages:

🔹 If you want to delete the RC and all its pods together, use:

kubectl delete rc nginx-rc --cascade=true



This ensures both the **ReplicationController and its managed pods** are cleaned up

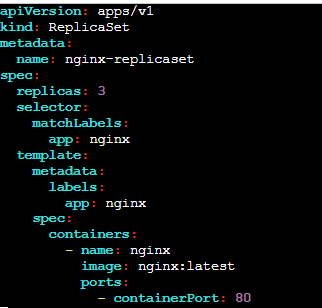
Key points:

* The selector must match the labels in the pod template.
* **ReplicationController** is now mostly replaced by **ReplicaSet** in newer versions of Kubernetes, but it is still valid for older versions or learning the basics.

**ReplicaSet**

A **ReplicaSet** is the next-generation controller that ensures a specified number of **identical pods** are always running. It is more flexible and powerful than the ReplicationController and supports **set-based selectors**. ReplicaSets are commonly used with **Deployments**, but they can also be used independently for learning purposes.

Here’s a simple YAML file for creating a ReplicaSet:



**Explanation of key fields:**

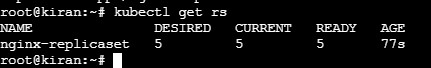
* **replicas**: Defines the number of pod replicas to maintain (3 in this case).
* **selector**: Matches pods with label app: nginx to manage.
* **template**: Defines the pod configuration with container image, name, and port.



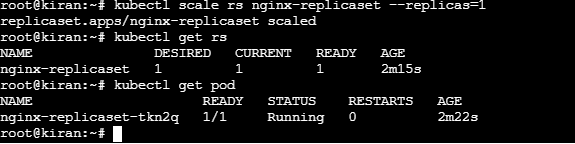
**Scaling the ReplicaSet**

To scale up the ReplicaSet to 5 pods, use:

kubectl scale rs nginx-replicaset --replicas=5

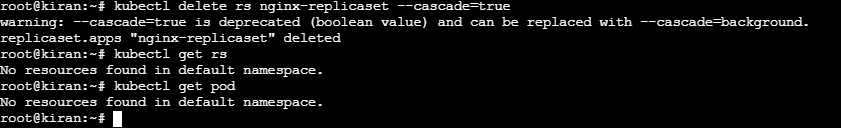


To scale down the ReplicaSet to 1 pod, use:



**Delete the ReplicaSet**

To delete the ReplicaSet along with all its managed pods:



**Key Points:**

• ReplicaSet is the successor to ReplicationController and supports set-based label selectors.  
• It ensures that the specified number of pod replicas are always running.

• ReplicaSet is often used by Deployments but can be used directly for learning and simple scenarios.  
• Like RC, it provides auto-healing by replacing failed or deleted pods automatically.

**Difference Between ReplicationController and ReplicaSet**

| **Feature** | **ReplicationController (RC)** | **ReplicaSet (RS)** |
| --- | --- | --- |
| **API Version** | v1 | apps/v1 |
| **Label Selector Type** | Only **equality-based selectors** (e.g., key = value) | Supports both **equality-based** and **set-based selectors** (e.g., in, notin) |
| **Usage** | Legacy (older Kubernetes versions) | Newer and recommended for use |
| **Template Field** | Uses spec.template to define pod | Same, uses spec.template |
| **Integration with Deployment** | Not used with Deployments | Commonly used with Deployments |
| **Auto-Healing** | Yes – replaces failed or deleted pods | Yes – same behavior |
| **Flexibility** | Limited due to basic selector support | More flexible and powerful |
| **Kubernetes Recommendation** | Deprecated in favor of ReplicaSet | Preferred controller |

**Summary:**

* Use **ReplicationController** only for legacy systems or basic learning.
* Use **ReplicaSet** for modern Kubernetes workloads—especially when working with **Deployments**, which manage ReplicaSets under the hood.
* The main technical upgrade is **support for set-based label selectors**, which allows more control over pod matching.