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.

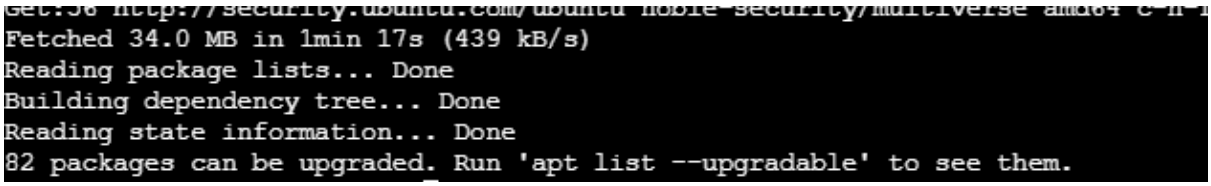
kOps setup in aws ec2

✅ 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

```
You can now run: /usr/local/bin/aws --version
root@kiran:~# echo 'export PATH=$PATH:/usr/local/bin/' >> ~/.bashrc
root@kiran:~# source ~/.bashrc
root@kiran:~# aws --version
aws-cli/2.27.8 Python/3.13.2 Linux/6.8.0-1024-aws exe/x86_64.ubuntu.24
root@kiran:~#
```

✅ 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/

```
root@kiran:~# 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/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0    449      0 --:--:-- --:--:-- --:--:--   449
100 57.3M  100 57.3M    0     0  19.9M      0  0:00:02  0:00:02 --:--:-- 24.4M
root@kiran:~#
```

 # 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

```
root@kiran:~# 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
--2025-05-06 14:02:14--  https://github.com/kubernetes/kops/releases/download/v1.24.1/kops-linux-amd64
Resolving github.com (github.com)... 20.207.73.82
Connecting to github.com (github.com)|20.207.73.82|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/62091339/e1dc29f3-046c-45f9-8991-a92fc7a4c3b8?X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Credential=releaseassetproduction%2F20250506%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20250506T140214Z&X-Amz-Expires=300&X-Amz-Signature=d53f43c3334de19f48f4d
d827b7b701831ee96b6aeb83db76f98daf33e9fec9e&X-Amz-SignedHeaders=host&response-content-disposition=attachment%3B%20filename%3Dkops-linux-amd64&response-content-type=ap
plication%2Foctet-stream [following]
--2025-05-06 14:02:14--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/62091339/e1dc29f3-046c-45f9-8991-a92fc7a4c3b8?X-Amz-Algorithm=AW
S4-HMAC-SHA256&X-Amz-Credential=releaseassetproduction%2F20250506%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20250506T140214Z&X-Amz-Expires=300&X-Amz-Signature=d53f43
c3334de19f48f4dd827b7b701831ee96b6aeb83db76f98daf33e9fec9e&X-Amz-SignedHeaders=host&response-content-disposition=attachment%3B%20filename%3Dkops-linux-amd64&response-
content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.110.133, 185.199.111.133, 185.199.109.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 156630244 (149M) [application/octet-stream]
Saving to: 'kops-linux-amd64'

kops-linux-amd64                    100%[===================================================================>] 149.37M  37.3MB/s    in 4.0s

2025-05-06 14:02:19 (37.3 MB/s) - 'kops-linux-amd64' saved [156630244/156630244]

root@kiran:~#
```

 # Check versions

kubectl version --client

kops version

```
root@kiran:~# kubectl version --client
Client Version: v1.33.0
Kustomize Version: v5.6.0
root@kiran:~# kops version
Client version: 1.24.1 (git-v1.24.1)
root@kiran:~#
```

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

1. Go to AWS Console → IAM → Users → Add user
2. Username: kops-admin
    a. 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

```
root@kiran:~# aws configure
AWS Access Key ID [****************6O6O]: AKIA5JMSUMT63VWT6O6O
AWS Secret Access Key [****************7n/A]: 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
Default region name [None]: ap-south-1
Default output format [table]: 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

```
root@kiran:~# aws s3api create-bucket \
  --bucket kirannk9.k8s.local \
  --region ap-south-1 \
  --create-bucket-configuration LocationConstraint=ap-south-1
-------------------------------------------------------------
|                        CreateBucket                       |
+----------+--------------------------------------------------+
|  Location|  http://kirannk9.k8s.local.s3.amazonaws.com/     |
+----------+--------------------------------------------------+
root@kiran:~# aws s3api put-bucket-versioning \
  --bucket kirannk9.k8s.local \
  --versioning-configuration Status=Enabled
root@kiran:~# export KOPS_STATE_STORE=s3://kirannk9.k8s.local
root@kiran:~#
```

Generate SSH key (optional if you already have one)

ssh-keygen

```
root@kiran:~# ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:BRE/gFYowEqqflTaUK80wKS6fLb1tl5+UAOkLMyo7ac root@kiran
The key's randomart image is:
+--[ED25519 256]--+
| .+o    +B+      |
| .oo=.= .=       |
|oo .o*.o  =      |
|+ o. +.. . +     |
|o. .* o S . .    |
|o..o o   .       |
|o..+ o   ..      |
| .o.= ..o  .     |
|  .E  o+...      |
+----[SHA256]-----+
root@kiran:~#
```

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
```

```
Cluster configuration has been created.

Suggestions:
 * list clusters with: kops get cluster
 * edit this cluster with: kops edit cluster kiran.k8s.local
 * edit your node instance group: kops edit ig --name=kiran.k8s.local nodes-ap-south-1a
 * edit your master instance group: kops edit ig --name=kiran.k8s.local master-ap-south-1a

Finally configure your cluster with: kops update cluster --name kiran.k8s.local --yes --admin
```

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

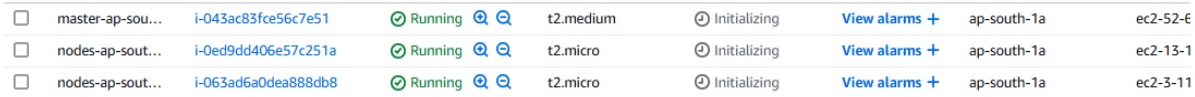
```
Cluster is starting.  It should be ready in a few minutes.

Suggestions:
 * validate cluster: kops validate cluster --wait 10m
 * list nodes: kubectl get nodes --show-labels
 * ssh to the master: ssh -i ~/.ssh/id_rsa ubuntu@api.kiran.k8s.local
 * the ubuntu user is specific to Ubuntu. If not using Ubuntu please use the appropriate user based on your OS.
 * read about installing addons at: https://kops.sigs.k8s.io/addons.

root@kiran:~#
```

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.




```
root@kiran:~# kops get cluster
NAME            CLOUD   ZONES
kiran.k8s.local aws     ap-south-1a
```

```
root@kiran:~# kubectl get nodes
NAME                  STATUS   ROLES           AGE     VERSION
i-043ac83fce56c7e51   Ready    control-plane   4m17s   v1.24.17
i-063ad6a0dea888db8   Ready    node            3m12s   v1.24.17
i-0ed9dd406e57c251a   Ready    node            3m7s    v1.24.17
```

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.

```
root@kiran:~# vi my-first-pod.yml
root@kiran:~# kubectl apply -f my-first-pod.yml
pod/my-first-pod created
```

```
root@kiran:~# kubectl get pod -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE                  NOMINATED NODE   READINESS GATES
my-first-pod  1/1     Running   0          56s   100.96.2.4   i-0ed9dd406e57c251a   <none>           <none>
root@kiran:~#
```

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.

```
root@kiran:~# kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
kiran          1/1     Running   0          16s
my-first-pod   1/1     Running   0          5m3s
root@kiran:~#
```

```
root@kiran:~# kubectl delete pods --all
pod "kiran" deleted
pod "my-first-pod" deleted
root@kiran:~# kubectl get pods
No resources found in default namespace.
root@kiran:~#
```

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**:

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

 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.

```
root@kiran:~# vi rc.yml
root@kiran:~# vi rc.yml
root@kiran:~# kubectl apply -f rc.yml
replicationcontroller/nginx-rc created
root@kiran:~# kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
nginx-rc-696mw   1/1     Running   0          14s
nginx-rc-kgs6s   1/1     Running   0          14s
nginx-rc-mkqm9   1/1     Running   0          14s
```

```
root@kiran:~# kubectl describe rc/nginx-rc
Name:         nginx-rc
Namespace:    default
Selector:     app=nginx
Labels:       app=nginx
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:         nginx:latest
    Port:          80/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Events:
  Type    Reason            Age   From                    Message
  ----    ------            ----  ----                    -------
  Normal  SuccessfulCreate  77s   replication-controller  Created pod: nginx-rc-mkqm9
  Normal  SuccessfulCreate  77s   replication-controller  Created pod: nginx-rc-kgs6s
  Normal  SuccessfulCreate  77s   replication-controller  Created pod: nginx-rc-696mw
root@kiran:~#
```

Scaling the Replication Controller

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

kubectl scale rc nginx-rc --replicas=5

```
root@kiran:~# kubectl scale rc nginx-rc --replicas=5
replicationcontroller/nginx-rc scaled
root@kiran:~# kubectl get rc
NAME       DESIRED   CURRENT   READY   AGE
nginx-rc   5         5         5       7m54s
root@kiran:~# kubectl get pod
NAME             READY   STATUS    RESTARTS   AGE
nginx-rc-d6mkd   1/1     Running   0          3m48s
nginx-rc-g7hvm   1/1     Running   0          3m48s
nginx-rc-p4f5k   1/1     Running   0          22s
nginx-rc-pnx76   1/1     Running   0          22s
nginx-rc-tp5c4   1/1     Running   0          3m48s
root@kiran:~#
```

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

kubectl scale rc nginx-rc --replicas=1

```
root@kiran:~# kubectl scale rc nginx-rc --replicas=1
replicationcontroller/nginx-rc scaled
root@kiran:~# kubectl get pod
NAME             READY   STATUS    RESTARTS   AGE
nginx-rc-g7hvm   1/1     Running   0          6m12s
root@kiran:~#
```

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.

```
root@kiran:~# kubectl delete pod --all
pod "nginx-rc-696mw" deleted
pod "nginx-rc-kgs6s" deleted
pod "nginx-rc-mkqm9" deleted
root@kiran:~# kubectl get pod
NAME             READY   STATUS    RESTARTS   AGE
nginx-rc-d6mkd   1/1     Running   0          9s
nginx-rc-g7hvm   1/1     Running   0          9s
nginx-rc-tp5c4   1/1     Running   0          9s
root@kiran:~#
```

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

```
root@kiran:~# kubectl get rc
No resources found in default namespace.
root@kiran:~# kubectl get po
No resources found in default namespace.
root@kiran:~#
```

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:

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

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.

```
root@kiran:~# kubectl apply -f rs.yml
replicaset.apps/nginx-replicaset created
root@kiran:~#
```

Scaling the ReplicaSet

To scale up the ReplicaSet to 5 pods, use:

kubectl scale rs nginx-replicaset --replicas=5

```
root@kiran:~# kubectl get rs
NAME               DESIRED   CURRENT   READY   AGE
nginx-replicaset   5         5         5       77s
root@kiran:~#
```

To scale down the ReplicaSet to 1 pod, use:

```
root@kiran:~# kubectl scale rs nginx-replicaset --replicas=1
replicaset.apps/nginx-replicaset scaled
root@kiran:~# kubectl get rs
NAME               DESIRED   CURRENT   READY   AGE
nginx-replicaset   1         1         1       2m15s
root@kiran:~# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-replicaset-tkn2q   1/1     Running   0          2m22s
root@kiran:~#
```

Delete the ReplicaSet

To delete the ReplicaSet along with all its managed pods:

```
root@kiran:~# kubectl delete rs nginx-replicaset --cascade=true
warning: --cascade=true is deprecated (boolean value) and can be replaced with --cascade=background.
replicaset.apps "nginx-replicaset" deleted
root@kiran:~# kubectl get rs
No resources found in default namespace.
root@kiran:~# kubectl get pod
No resources found in default namespace.
root@kiran:~#
```

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.