Kubernetes (K3S) on AWS EC2

You need

- 3 EC2 instances (1 master, 2 workers)
- Configured with permissions for the necessary ports (e.g. SSH, K3s API, NodePort).
- AWS CLI configured for your account.

Inbound Rules need to allow for K3s Nodes

Protocol	Port	Source	Destination	Description
TCP	2379-2380	Servers	Servers	Required only for HA with embedded etcd
TCP	6443	Agents	Servers	K3s supervisor and Kubernetes API Server
UDP	8472	All nodes	All nodes	Required only for Flannel VXLAN
TCP	10250	All nodes	All nodes	Kubelet metrics
UDP	51820	All nodes	All nodes	Required only for Flannel Wireguard with IPv4
UDP	51821	All nodes	All nodes	Required only for Flannel Wireguard with IPv6
TCP	5001	All nodes	All nodes	Required only for embedded distributed registry (Spegel)
TCP	6443	All nodes	All nodes	Required only for embedded distributed registry (Spegel)

- The K3s server needs port 6443 to be accessible by all nodes.
- Typically, all outbound traffic is allowed.

Step 1: Install K3S on Master Node

update

```
sudo dnf update
```

Install k3s on master node.

```
curl -sfL https://get.k3s.io | sh -
```

Check Status

```
sudo systemctl status k3s
```

Now Get the node token (needed to join worker nodes):
You need this token to securely join worker nodes to the master node (also known as the control plane).

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

Step 2: Prepare Worker Node

Install Docker in the both worker machine and set appropriate hostnames

```
sudo dnf update
sudo dnf install docker -y
sudo systemctl status docker
sudo hostnamectl set-hostname workernode1
sudo hostnamectl set-hostname workernode2
```

Step 3: Join Worker Nodes to the Cluster (Master Node)

First get the token form master node. We already collect the token earlier this tutorial. Now to join the nodes in master nodes run this command from worker nodes.

```
curl -sfL https://get.k3s.io | K3S_URL=https://<MasterNodePublicIP>:6443 K3S_TOKEN=<NodeToken> sh -
```

*Here change the <MasterNodePublicIP> with your EC2 master node Public IP & <NodeToken> with your token from master node.

Now Check the nodes list from master node

```
sudo kubectl get nodes
```

Step 4: Now Deploy a JS Game on k3s:

To run an application on K3s, we need to create Kubernetes deployment and service files.

First Create a Namespace

Kubernetes namespace is like a separate section within a Kubernetes cluster, where you can keep your resources organized and isolated from others. Think of it as different folders on your computer. Each namespace (or folder) can have its own set of applications, settings, and permissions, and they won't interfere with each other.

```
sudo kubectl create namespace <name>
```

Now Create Kubernetes Deployment File:

```
nano deployment.yml
```

*you can place this file anywhere

Example 1:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: flask-app
spec:
 replicas: 2
 selector:
  matchLabels:
   app: flask-app
 template:
  metadata:
   labels:
    app: flask-app
  spec:
   containers:
   - name: flask-app
    image: your-dockerhub-username/flask-app:latest
    ports:
    - containerPort: 5000
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  selector:
    app: flask-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

Example 2:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: js-game-app
  labels:
    app: js-game-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: js-game-app
  template:
    metadata:
      labels:
        app: js-game-app
    spec:
      containers:
        - name: js-game-app
          image: internaldev/jsgame:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: js-game-app-service
  name: js-game-app-service
spec:
```

```
  ports:
    - name: "3000-80"
      port: 3000
      protocol: TCP
      targetPort: 80
  selector:
    app: js-game-app
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: js-game-app-ingress
  annotations:
    ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: js-game-app-service
                port:
                  number: 3000
```

* in the Image line you can update your dockerhub or any image link.

Applying the Manifest

```
sudo kubectl apply -f deployment.yml -n <namespace>
```

Verify the Deployment

To check if the deployment, service, and ingress are created successfully, you can use the following commands:

```
sudo kubectl get deployments -n <namespace>
sudo kubectl get services -n <namespace>
sudo kubectl get ingress -n <namespace>
```

Now check the pods running....

```
sudo kubectl get pods -n <namespace>
```

Now open the browser and visit with your Master Node Public IP:port

Managing a k3s Cluster on Amazon Linux 2023

k3s is a lightweight Kubernetes distribution optimized for resource efficiency. Below are the key tasks for managing a k3s cluster.

1. Check k3s Status

```
sudo systemctl status k3s
```

If k3s is not running, start it:

```
sudo systemctl start k3s
```

Enable k3s to start on boot:

```
sudo systemctl enable k3s
```

2. Check Cluster Nodes and Pods

List Nodes

```
kubectl get nodes
```

Expected output:

NAME	STATUS	ROLES	AGE	VERSION
my-node	Ready	control-plane,master	10m	v1.26.0

List All Running Pods

```
kubectl get pods -A
```

3. Configure kubectl Access

If kubectl is not working, set the correct kubeconfig file:

```
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
```

Make it permanent:

```
echo 'export KUBECONFIG=/etc/rancher/k3s/k3s.yaml' >> ~/.bashrc
source ~/.bashrc
```

Now, try:

```
kubectl get nodes
```

4. Deploy Applications

Example: Deploy a Simple Nginx Pod (deployment.yml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
```

```
  matchLabels:
    app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 selector:
  app: nginx
 ports:
  - protocol: TCP
   port: 80
   targetPort: 80
 type: LoadBalancer
```

Apply the YAML file:

```
kubectl apply -f nginx.yaml
```

Check the deployment:

```
kubectl get pods
kubectl get services
```

5. Scale an Application

To scale up/down a deployment:

```
kubectl scale deployment nginx-deployment --replicas=4
```

Check if new pods are created:

```
kubectl get pods
```

6. Restart k3s

Restart the k3s service:

```
sudo systemctl restart k3s
```

7. Stop & Disable k3s

Stop k3s:

```
sudo systemctl stop k3s
```

Disable auto-start:

```
sudo systemctl disable k3s
```

8. Uninstall k3s

If you need to remove k3s completely:

```
sudo /usr/local/bin/k3s-uninstall.sh
```

9. Troubleshooting

Check k3s Logs

```
sudo journalctl -u k3s -f
```

Check Pod Events

```
kubectl get events
```

Describe a Pod for Debugging

```
kubectl describe pod pod-name
```

Access a Running Container

```
kubectl exec -it pod-name -- /bin/sh
```

Deleting Nodes and Pods in Kubernetes (k3s or EKS)

1. Delete a Pod

   First, list all pods:

   ```
   kubectl get pods -A
   ```

   Then delete a specific pod:

   ```
   kubectl delete pod pod-name -n namespace
   ```

   Example:

   ```
   kubectl delete pod myapp-12345 -n default
   ```

   To delete all pods in a namespace:

   ```
   kubectl delete pods --all -n namespace
   ```

   Example:

   ```
   kubectl delete pods --all -n default
   ```

2. Delete a Deployment (Removes All Related Pods)

```
kubectl delete deployment deployment-name
```

Example:

```
kubectl delete deployment myapp-deployment
```

3. Delete a Node from the Cluster

List Nodes

```
kubectl get nodes
```

Drain the Node (Remove Running Pods)

```
kubectl drain node-name --ignore-daemonsets --delete-emptydir-data
```

Example:

```
kubectl drain my-node --ignore-daemonsets --delete-emptydir-data
```

Delete the Node from the Cluster

```
kubectl delete node node-name
```

Example:

```
kubectl delete node my-node
```

4. Remove a Worker Node (k3s)

If you're using k3s with multiple nodes, remove a worker node by running this on the worker node:

```
sudo k3s-agent-uninstall.sh
```

On the master node, remove the node entry:

```
kubectl delete node worker-node-name
```

5. Force Delete Stuck Pods

If a pod is stuck in Terminating or Evicted state, force delete it:

```
kubectl delete pod pod-name --grace-period=0 --force -n namespace
```

Deleting a k3s Cluster on Amazon Linux 2023

If you installed k3s, you can remove the cluster completely using the following steps.

Check if k3s is Running

Before deleting the cluster, verify if k3s is active:

```
kubectl get nodes
```

If k3s is running, stop it first.

```
sudo systemctl stop k3s
```

Prevent k3s from starting on boot:

```
sudo systemctl disable k3s
```

Uninstall k3s
Run the official uninstall script provided by k3s:

```
sudo /usr/local/bin/k3s-uninstall.sh
```

This script removes:

- ✓ The k3s binary
- ✓ The systemd service
- ✓ The Kubernetes configuration files (/etc/rancher/k3s/)

To confirm the removal, check:

```
kubectl get nodes
```

Remove Kubernetes Configuration (Optional)
If you want to remove all Kubernetes-related files, run:

```
rm -rf ~/.kube /etc/rancher/k3s
```