# **Setup Kubernetes on Amazon EKS**

You can follow same procedure in the official AWS document [Getting started with Amazon EKS – eksctl](https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html)

Setup Kubectl -

a. Download kubectl version 1.20  
b. Grant execution permissions to kubectl executable  
c. Move kubectl onto /usr/local/bin  
d. Test that your kubectl installation was successful

curl -o kubectl <https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl>

chmod +x ./kubectl

mv ./kubectl /usr/local/bin

kubectl version --short --client

Setup eksctl

1. Download

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

1. sudo mv /tmp/eksctl /usr/local/bin
2. eksctl version

IAM user should have access to

IAM

EC2

VPC

CloudFormation

eksctl create cluster --name cluster-name \

--region region-name \

--node-type instance-type \

--nodes-min 2 \

--nodes-max 10 \

--zones <AZ-1>,<AZ-2>

example:

eksctl create cluster --name guvi \

--region ap-south-1 \

--node-type t2.small

aws eks update-kubeconfig --name guvi --region ap-south-1

1. To delete the EKS clsuter  
   eksctl delete cluster guvi --region ap-south-1

Validate your cluster using by creating by checking nodes and by creating a pod  
kubectl get nodes

**# nginx-pod.yaml**

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

tier: dev

spec:

containers:

- name: nginx-container

image: nginx

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Create and display Pods

# Create and display PODs

kubectl create -f nginx-pod.yaml

kubectl get pod

kubectl get pod -o wide

kubectl get pod nginx-pod -o yaml

kubectl describe pod nginx-pod

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Test & Delete

# To get inside the pod

kubectl exec -it nginx-pod -- /bin/sh

# Create test HTML page

cat <<EOF > /usr/share/nginx/html/test.html

<!DOCTYPE html>

<html>

<head>

<title>Testing..</title>

</head>

<body>

<h1 style="color:rgb(90,70,250);">Hello, Kubernetes...!</h1>

<h2>Congratulations, you passed :-) </h2>

</body>

</html>

EOF

exit

# Expose PODS using NodePort service

kubectl expose pod nginx-pod --type=NodePort --port=80

# Display Service and find NodePort

kubectl describe svc nginx-pod

# Open Web-browser and access webapge using

http://nodeip:nodeport/test.html

# Delete pod & svc

kubectl delete svc nginx-pod

kubectl delete pod nginx-pod

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**# 1. Deployment YAML file**

# nginx-deploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deploy

labels:

app: nginx-app

spec:

replicas: 3

template:

metadata:

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx:1.7.9

ports:

- containerPort: 80

selector:

matchLabels:

app: nginx-app

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 2. Create and Display Deployment

kubectl create -f nginx-deploy.yaml

kubectl get deploy -l app=nginx-app

kubectl get rs -l app=nginx-app

kubectl get po -l app=nginx-app

kubectl describe deploy nginx-deploy

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 3. Testing: Rollback update

kubectl set image deploy nginx-deploy nginx-container=nginx:1.91 --record

kubectl rollout status deployment/nginx-deploy

kubectl rollout history deployment/nginx-deploy

kubectl rollout undo deployment/nginx-deploy

kubectl rollout status deployment/nginx-deploy

kubectl describe deploy nginx-deploy | grep -i image

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 4. Testing: Update Version of "nginx:1.7.9" to "nginx:1.9.1"

kubectl set image deploy nginx-deploy nginx-container=nginx:1.9.1

kubectl edit deploy nginx-deploy

kubectl rollout status deployment/nginx-deploy

kubectl get deploy

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 5. Testing: Scale UP

kubectl scale deployment nginx-deploy --replicas=5

kubectl get deploy

kubectl get po -o wide

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 6. Testing: Scale DOWN

kubectl scale deployment nginx-deploy --replicas=3

kubectl get deploy

kubectl get po -o wide

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 7. Cleanup

kubectl delete -f nginx-deploy.yaml

kubectl get deploy

kubectl get rs

kubectl get po

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

React Application

# Stage 1: Build stage

FROM node:14 AS mybuild

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm ci

COPY . .

RUN npm run build

# Stage 2: Production stage

FROM nginx:1.21

COPY --from=mybuild /app/build /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

→docker build -t reactapp .

Docker image – reactapp

Docker run command or DockerCompose

docker run -d -p 7000:80 reactapp

Docker-compose.yml

“””

version: '3'

service​​s:​​

app:

build:

context: .

dockerfile: reactapp

ports:

- 80:80

“”””

→docker-compose up -d

‘’’’

version: '3'

services:

web:

image: reactapp

ports:

- 80:80

“””

Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-deployment

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-container

image: reactapp

ports:

- containerPort: 80

Service

Service.yml

appVersion: v1

kind : Service

metadata:

name: my-service

spec:

selector:

app: my-app

ports:

* port: 80

targetPort: 4000

type: LoadBalancer

kubectl apply -f deployment.yml

kubectl apply -f service.yml

Kubectl get service my-service

External ip or hostname

Ip:4000

====================================

Python

Deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-python-app

spec:

replicas: 5

selector:

matchLabels:

app: my-python-app

template:

metadata:

labels:

app: my-python-app

spec:

containers:

- name: my-python-app

image: my-python-app-image:latest

ports:

- containerPort: 5000

##Service.yml

apiVersion: v1

kind: Service

metadata:

name: my-python-app-service

spec:

selector:

app: my-python-app

ports:

- protocol: TCP

port: 5000

targetPort: 5000

type: LoadBalancer

—----------------------------

Dockerfile—-

# Base image

FROM python:3.9

# Set working directory

WORKDIR /app

# Copy requirements.txt to the container

COPY requirements.txt .

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code to the container

COPY . .

# Expose the application port

EXPOSE 5000

CMD ["python", "app.py"]

**Kubernetes Objects**

Kubernetes provides various types of objects that you can use to define and manage the desired state of your applications and resources within the cluster. Here are some of the most commonly used Kubernetes objects:

**Pod**:

The smallest deployable unit in Kubernetes.

Represents a single instance of a running process in the cluster.

Can contain one or more containers.

Usually not directly used for managing applications, but other resources like Deployments manage Pods.

**Deployment**:

Used to manage the deployment and scaling of Pods.

Provides features like replica management, rolling updates, and rollbacks.

Ensures a specified number of Pod replicas are running and available.

**Service**:

Defines a set of Pods and a policy to access them.

Provides a stable endpoint for accessing one or more Pods, load balancing, and service discovery.

Types include ClusterIP, NodePort, LoadBalancer, and ExternalName services.

**ConfigMap**:

Stores configuration data in key-value pairs.

Can be used to configure applications separately from the Pods that run them.

Changes to ConfigMaps can trigger updates in Pods.

**Secret**:

Similar to ConfigMaps but used for sensitive data like passwords, tokens, and certificates.

Provides a way to store and manage sensitive information securely.

**Namespace**:

Logical partition of a Kubernetes cluster.

Allows for isolation and separation of resources within the cluster.

Used to organize and control access to resources.

**StatefulSet:**

Used to manage stateful applications (e.g., databases) where each Pod has a unique identity.

Provides stable network identities and ordered deployment and scaling.

**DaemonSet:**

Ensures that a copy of a Pod is running on all or a subset of nodes in the cluster.

Useful for running monitoring agents, log collectors, and other node-level tasks.

**Job and CronJob:**

Jobs run a task to completion, such as a batch job or a one-time operation.

CronJobs schedule Jobs to run at specified times or intervals.

**HorizontalPodAutoscaler (HPA):**

Automatically adjusts the number of Pod replicas in a Deployment or ReplicaSet based on CPU or custom metrics.

Allows for automatic scaling of your applications.

**Ingress**:

Manages external access to services within a cluster.

Provides rules for routing external HTTP and HTTPS traffic to services and Pods.

These are just some of the core Kubernetes objects. Kubernetes is highly extensible and allows for the creation of custom resources (Custom Resource Definitions or CRDs) to represent complex application-specific objects. The choice of which Kubernetes objects to use depends on the requirements and architecture of your applications and services within the cluster.

**PVC**

**Create a PersistentVolumeClaim (PVC):**

First, create a PersistentVolumeClaim (PVC) that defines the desired storage and access mode.

Pvc.yml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

storageClassName: your-storage-class

Deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-deployment

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-container

image: nginx

volumeMounts:

- name: my-volume

mountPath: /path/in/container

volumes:

- name: my-volume

persistentVolumeClaim:

claimName: my-pvc

kubectl apply -f pvc.yaml

kubectl apply -f deployment.yaml