A) How to create an EKS Cluster from AWS Console?

**Step-1: Configure Cluster**

**Name**: << Name of the Cluster >>

**Kubernetes version**: 1.21 or 1.22

**Cluster Service Role**: << IAM Role to allow the Kubernetes control plane to manage AWS resources on your behalf >>: Follow below steps to create an IAM Role if it is not present already.

Steps to create a Cluster Service Role

(1) Go to IAM Console -> Roles -> Create Role.

(2) Step-1:

(2.1) Trusted Entity Type: AWS Service

(2.2) Use case: EKS (Choose EKS from Use cases for other AWS services dropdown)

(2.3) Choose ‘EKS – Cluster` from list of EKS use case.

Step-2: Add Permissions

(+) Make sure ‘**AmazonEKSClusterPolicy**’, ` **AmazonEKSVPCResourceController**` AWS Managed policies are present. If not, attach these policies

**AmazonEKSVPCResourceController**: Policy used by VPC Resource Controller to manage ENI and IPs for worker nodes.

**AmazonEKSClusterPolicy:** This policy provides Kubernetes the permissions it requires to manage resources on your behalf.

Step-3: Name, review, and create

(3.1) Enter Role name – Eg: eks-cluster-role

(3.2) Review configuration and create role.

**Tags**: << Tags >>

**Step-2: Networking**

**Networking:**

**VPC**: << VPC ID>>

**Subnets**: << Choose Public / Private Subnets in different AZ, Minimum: 2 >>

**Security groups**: Leave it empty by default.

Note: Amazon EKS clusters, starting with Kubernetes version 1.14 and platform version eks.3, create a cluster security group when they are created. A cluster security group is designed to allow all traffic from the control plane and managed node groups to flow freely between each other. However, we can attach additional security groups if required.

The cluster security group is applied by default to the Kubernetes control plane managed by Amazon EKS as well as any managed compute resources created through the Amazon EKS API.

Reference: <https://docs.aws.amazon.com/eks/latest/userguide/sec-group-reqs.html>

**cluster IP address family**: ipv4

**Cluster endpoint access:** Public / Private / Public and private (based on requirement)

**Networking add-ons**: Go with default values (Amazon VPC CNI, CoreDNS, kube-proxy)

**Step-3: Logging**

Enable Logging for below components (if required)

a) API Server

b) Audit

c) Authenticator

d) Controller Manager

e) Scheduler

**Step-4: Review and Create an EKS Cluster**

Review Configuration and Create an EKS Cluster

B) How to create an EKS Managed Node Group from AWS Console?

Intro on AWS EKS Managed Node Group <https://docs.aws.amazon.com/eks/latest/userguide/managed-node-groups.html>

Go to ‘Compute’ section of your eks cluster -> Add Node Group.

**Step-1: Configure Node Group**

**Name:** << Name of EKS Managed Node Group >>

**Node IAM Role**: IAM Role that will be used by the nodes. Follow below steps to create an IAM Role if it is not present already.

(1) Go to IAM Console -> Roles -> Create Role.

(2) Step-1:

(2.1) Trusted Entity Type: AWS Service

(2.2) Use case: EC2

Step-2: Add below AWS Managed Permissions

a) **AmazonEKSWorkerNodePolicy**: Allows Amazon EKS worker nodes to connect to Amazon EKS Clusters.

b) **AmazonEC2ContainerRegistryReadOnly**: Provides read-only access to Amazon EC2 Container Registry repositories.

c) **AmazonEKS\_CNI\_Policy**: This policy provides the Amazon VPC CNI Plugin (amazon-vpc-cni-k8s) the permissions it requires to modify the IP address configuration on your EKS worker nodes. This permission set allows the CNI to list, describe, and modify Elastic Network Interfaces on your behalf.

(3) Role name: eks-cluster-ng-role

**Tags**: << Tags >>

**Step-2: Set compute and scaling configuration**

**AMI type**: Amazon Linux 2 (AL2\_X86\_64)

**Capacity type**: On-Demand

**Instance Types**: t3.medium

**Disk Size**: 50 GiB

**Scaling Configuration**: Min: 1, Max: 2, Desired: 2

**Node Group Update Configuration**: Maximum unavailable: 1

**Step-3: Networking**

Choose Subnets – Public / Private Subnets (Minimum: 2 for High Availability)

**Step-4: Review and Create an EKS Managed Node Group**

Review Configuration and Create an EKS Managed Node Group

Observation: When we create an EKS Managed Node Group from Console, there is no option to specify node group related security group. By default, EKS will attach cluster security group which allows all traffic from the control plane and managed node groups to flow freely between each other. If we want an additional security group, we can create and attach externally or create a launch template (recommended one) and use it in node group creation.

EKS Node Group – Additional Security Group

sg-name: eks-nodegroup-security-group

Inbound Rules

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Protocol** | **Port** | **Source** |
| Minimum inbound traffic | SSH | 22 | 0.0.0.0/0 or Custom CIDR / IP |
| Custom TCP | TCP | 31231 | 0.0.0.0/0 or Custom CIDR / IP |
|  | All | All | Self |

Outbound Rules

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Protocol** | **Port** | **Source** |
| Recommended Outbound traffic | All | All | 0.0.0.0/0 |

C) How to access an EKS Cluster?

When you create an Amazon EKS cluster, the AWS Identity and Access Management (IAM) entity user or role, such as a federated user that creates the cluster, is automatically granted **system: masters** permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the aws-auth ConfigMap within Kubernetes and create a Kubernetes rolebinding or clusterrolebinding with the name of a group that you specify in the aws-auth ConfigMap.

Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as kubectl). If an endpoint is public, we can directly use it to access cluster (our identity should be authorized by RBAC). If it’s private, we can either use a jumpbox or load balancer to access it (jumpbox or load balancer should be authorized by K8s RBAC).

Access Cluster which has an API server endpoint is public to the internet using Kubectl utility

1) To install kubectl: <https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>

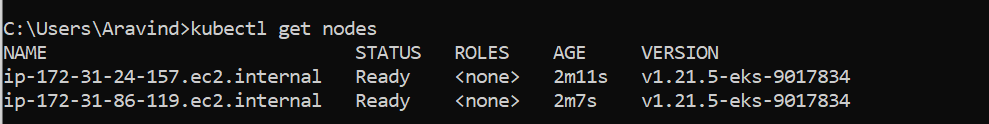
2) Configure kubectl so that you can connect to an Amazon EKS cluster.

**aws eks update-kubeconfig --name << NAME OF CLUSTER >> --region << REGION >>**

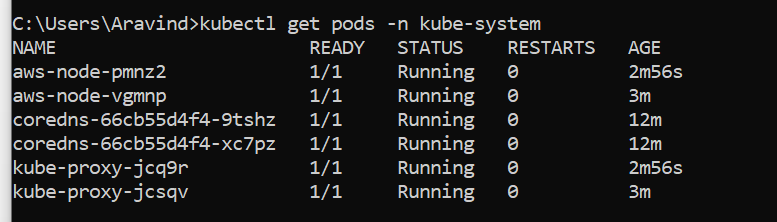
Note: It will update kube config file present in ~/.kube/ directory.

Basic Verification

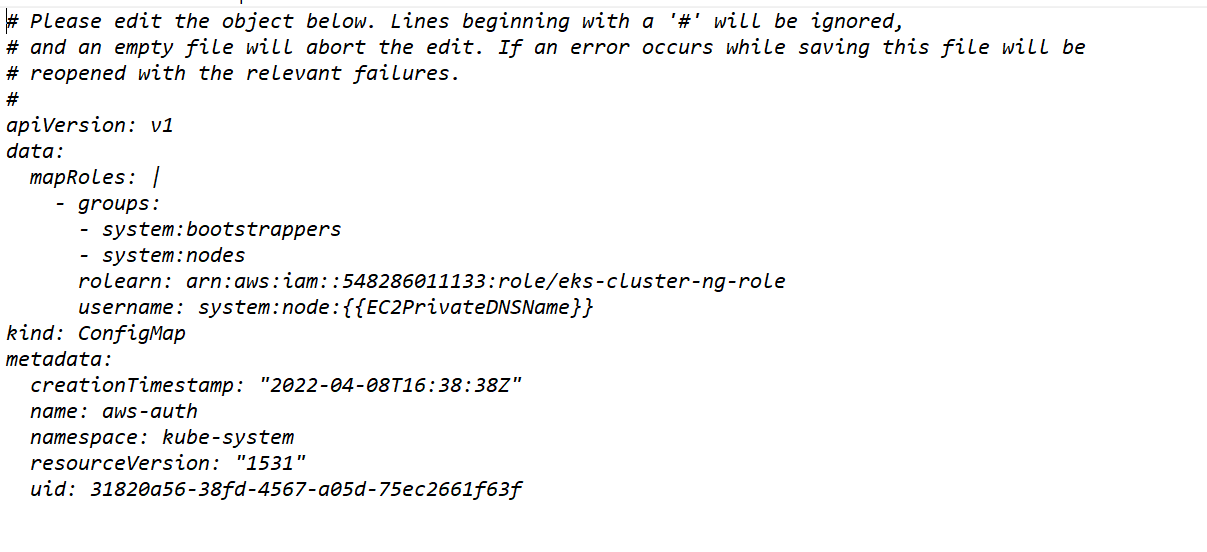
1) To list nodes: **kubectl get nodes**



2) To list pods running in kube-system namespace: **kubectl get pods –n kube-system**



3) To see eks cluster aws-auth map: **kubectl edit configmap aws-auth –n kube-system**



Note: rolearn - Worker node IAM Role.

D) Deploy an Nginx Application in K8s and Expose it through internet.

**1) Deploy Nginx Application in Pod – K8s Manifest**

nginx\_pod.yml: kubectl apply –f nginx\_pod.yml

apiVersion: v1                      # String , api version of k8s objects

kind: Pod                           # String, Type of k8s objects

metadata:                           # map, metadata about objects

  name: nginx-pod

  labels:               # map, Key value pairs (like a tag)

    product: nginx

spec:                               # Real configuration of k8s object

  containers:           # List

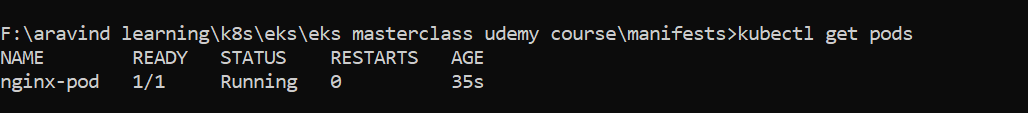
    - name: nginx-container

      image: stacksimplify/kubenginx:1.0.0

      ports:

        - containerPort: 80

Verification: kubectl get pods



kubectl describe pod nginx-pod: To describe pod properties and events.

**2) Expose Nginx Application – K8s Manifest**

nginx\_node\_port.yml : kubectl apply –f nginx\_node\_port.yml

apiVersion: v1

kind: Service

metadata:

  name: nginx-nodeport-service

  labels:

    product: nginx

spec:

  selector:

     product: nginx #Route service traffic to pods with label keys and values matching this selector

  ports:

    - port: 80 # The port that will be exposed by this service.

      targetPort: 80 # Container Port

      nodePort: 31231 # Worker Node Port

  type: NodePort

Verification: kubectl get svc

If worker nodes are deployed in public subnet,

Get Public IP of EC2 Worker Nodes, Access Nginx Application in browser.

http://<<WORKER\_NODE\_ID>>: 31231

