

# How to Create EKS Cluster Using Terraform?

How to Create EKS Cluster Using Terraform + IAM Roles for Ser...



- You can find the source code for this video in my [GitHub Repo](#)

## Create AWS VPC using Terraform

- Let's start with terraform. First, we need to create an AWS provider. It allows to interact with the many resources supported by AWS, such as VPC, EC2, EKS, and many others. You must configure the provider with the proper credentials before using it. The most common authentications methods:
  - AWS shared credentials/configuration files
  - Environment variables
  - Static credentials
  - EC2 instance metadata
- Create AWS provider and give it a name `terraform/0-provider.tf`.

**0-provider.tf**

```

1 provider "aws" {
2   region = "us-east-1"
3 }
4
5 terraform {
6   required_providers {
7     aws = {
8       source  = "hashicorp/aws"
9       version = "~> 3.0"
10    }
11  }
12}

```

- The next step is to create a virtual private cloud in AWS using the `aws_vpc` resource. There is one required field that you need to provide, which is the size of your network. `10.0.0.0/16` will give you approximately 65 thousand IP addresses. For your convenience, you can also give it a tag, for example, `main`. Let's name it `terraform/1-vpc.tf`.

**1-vpc.tf**

```

1 resource "aws_vpc" "main" {
2   cidr_block = "10.0.0.0/16"
3
4   tags = {
5     Name = "main"
6   }
7 }

```

## Create Internet Gateway AWS using Terraform

- To provide internet access for your services, we need to have an internet gateway in our VPC. You need to attach it to the VPC that we just created. It will be used as a default route in public subnets. Give it a name `terraform/2-igw.tf`.

**2-igw.tf**

```

1 resource "aws_internet_gateway" "igw" {
2   vpc_id = aws_vpc.main.id
3
4   tags = {
5     Name = "igw"
6   }
7 }

```

```

6    }
7 }
```

## Create private and public subnets in AWS using Terraform

- Now, we need to create four subnets. To meet EKS requirements, we need to have two public and two private subnets in different availability zones. File name is `terraform/3-subnets.tf`.

### 3-subnets.tf

```

1 resource "aws_subnet" "private-us-east-1a" {
2   vpc_id          = aws_vpc.main.id
3   cidr_block      = "10.0.0.0/19"
4   availability_zone = "us-east-1a"
5
6   tags = {
7     "Name"           = "private-us-east-1a"
8     "kubernetes.io/role/internal-elb" = "1"
9     "kubernetes.io/cluster/demo"       = "owned"
10  }
11 }
12
13 resource "aws_subnet" "private-us-east-1b" {
14   vpc_id          = aws_vpc.main.id
15   cidr_block      = "10.0.32.0/19"
16   availability_zone = "us-east-1b"
17
18   tags = {
19     "Name"           = "private-us-east-1b"
20     "kubernetes.io/role/internal-elb" = "1"
21     "kubernetes.io/cluster/demo"       = "owned"
22  }
23 }
24
25 resource "aws_subnet" "public-us-east-1a" {
26   vpc_id          = aws_vpc.main.id
27   cidr_block      = "10.0.64.0/19"
28   availability_zone = "us-east-1a"
29   map_public_ip_on_launch = true
30
31   tags = {
32     "Name"           = "public-us-east-1a"
33     "kubernetes.io/role/elb" = "1"
34     "kubernetes.io/cluster/demo" = "owned"
35  }
36 }
37
38 resource "aws_subnet" "public-us-east-1b" {
39   vpc_id          = aws_vpc.main.id
```

```

40  cidr_block          = "10.0.96.0/19"
41  availability_zone   = "us-east-1b"
42  map_public_ip_on_launch = true
43
44  tags = {
45      "Name"           = "public-us-east-1b"
46      "kubernetes.io/role/elb" = "1"
47      "kubernetes.io/cluster/demo" = "owned"
48  }
49 }
```

## Create NAT Gateway in AWS using Terraform

- It's time to create a NAT gateway. It is used in private subnets to allow services to connect to the internet. For NAT, we need to allocate public IP address first. Then we can use it in the `aws_nat_gateway` resource. The important part here, you need to place it in the public subnet. That subnet must have an internet gateway as a default route. Give it a name `terraform/4-nat.tf`.

### 4-nat.tf

```

1  resource "aws_eip" "nat" {
2      vpc = true
3
4      tags = {
5          Name = "nat"
6      }
7  }
8
9  resource "aws_nat_gateway" "nat" {
10     allocation_id = aws_eip.nat.id
11     subnet_id     = aws_subnet.public-us-east-1a.id
12
13    tags = {
14        Name = "nat"
15    }
16
17    depends_on = [aws_internet_gateway.igw]
18 }
```

- By now, we have created subnets, internet gateway, and nat gateway. It's time to create routing tables and associate subnets with them. File name is `terraform/5-routes.tf`.

### 5-routes.tf

```

1 resource "aws_route_table" "private" {
2   vpc_id = aws_vpc.main.id
3
4   route = [
5     {
6       cidr_block          = "0.0.0.0/0"
7       nat_gateway_id      = aws_nat_gateway.nat.id
8       carrier_gateway_id  = ""
9       destination_prefix_list_id = ""
10      egress_only_gateway_id = ""
11      gateway_id          = ""
12      instance_id         = ""
13      ipv6_cidr_block     = ""
14      local_gateway_id    = ""
15      network_interface_id = ""
16      transit_gateway_id  = ""
17      vpc_endpoint_id     = ""
18      vpc_peering_connection_id = ""
19     },
20   ]
21
22   tags = {
23     Name = "private"
24   }
25 }
26
27 resource "aws_route_table" "public" {
28   vpc_id = aws_vpc.main.id
29
30   route = [
31     {
32       cidr_block          = "0.0.0.0/0"
33       gateway_id          = aws_internet_gateway.igw.id
34       nat_gateway_id      = ""
35       carrier_gateway_id  = ""
36       destination_prefix_list_id = ""
37       egress_only_gateway_id = ""
38       instance_id         = ""
39       ipv6_cidr_block     = ""
40       local_gateway_id    = ""
41       network_interface_id = ""
42       transit_gateway_id  = ""
43       vpc_endpoint_id     = ""
44       vpc_peering_connection_id = ""
45     },
46   ]
47
48   tags = {
49     Name = "public"
50   }
51 }
52
53 resource "aws_route_table_association" "private-us-east-1a" {

```

```

54   subnet_id      = aws_subnet.private-us-east-1a.id
55   route_table_id = aws_route_table.private.id
56 }
57
58 resource "aws_route_table_association" "private-us-east-1b" {
59   subnet_id      = aws_subnet.private-us-east-1b.id
60   route_table_id = aws_route_table.private.id
61 }
62
63 resource "aws_route_table_association" "public-us-east-1a" {
64   subnet_id      = aws_subnet.public-us-east-1a.id
65   route_table_id = aws_route_table.public.id
66 }
67
68 resource "aws_route_table_association" "public-us-east-1b" {
69   subnet_id      = aws_subnet.public-us-east-1b.id
70   route_table_id = aws_route_table.public.id
71 }
```

## Create EKS cluster using Terraform

- Finally, we got to the EKS cluster. Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service. For example, EKS will create an autoscaling group for each instance group if you use managed nodes. Before you can create Amazon EKS clusters, you must create an IAM role with the `AmazonEKSClusterPolicy`. Let's name it `terraform/6-eks.tf`.

### 6-eks.tf

```

1 resource "aws_iam_role" "demo" {
2   name = "eks-cluster-demo"
3
4   assume_role_policy = <<POLICY
5 {
6     "Version": "2012-10-17",
7     "Statement": [
8       {
9         "Effect": "Allow",
10        "Principal": {
11          "Service": "eks.amazonaws.com"
12        },
13        "Action": "sts:AssumeRole"
14      }
15    ]
16  }
17 POLICY
18 }
19 }
```

```

20 resource "aws_iam_role_policy_attachment" "demo-AmazonEKSClusterPolicy" {
21   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
22   role       = aws_iam_role.demo.name
23 }
24
25 resource "aws_eks_cluster" "demo" {
26   name      = "demo"
27   role_arn  = aws_iam_role.demo.arn
28
29   vpc_config {
30     subnet_ids = [
31       aws_subnet.private-us-east-1a.id,
32       aws_subnet.private-us-east-1b.id,
33       aws_subnet.public-us-east-1a.id,
34       aws_subnet.public-us-east-1b.id
35     ]
36   }
37
38   depends_on = [aws_iam_role_policy_attachment.demo-AmazonEKSClusterPolicy]
39 }
```

- Next, we are going to create a single instance group for Kubernetes. Similar to the EKS cluster, it requires an IAM role as well.

### 7-nodes.tf

```

1 resource "aws_iam_role" "nodes" {
2   name = "eks-node-group-nodes"
3
4   assume_role_policy = jsonencode({
5     Statement = [
6       Action = "sts:AssumeRole"
7       Effect = "Allow"
8       Principal = {
9         Service = "ec2.amazonaws.com"
10      }
11    }]
12   Version = "2012-10-17"
13 })
14 }
15
16 resource "aws_iam_role_policy_attachment" "nodes-AmazonEKSWorkerNodePolicy" {
17   policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
18   role       = aws_iam_role.nodes.name
19 }
20
21 resource "aws_iam_role_policy_attachment" "nodes-AmazonEKS_CNI_Policy" {
22   policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
23   role       = aws_iam_role.nodes.name
24 }
25 }
```

```

26 resource "aws_iam_role_policy_attachment" "nodes-
27 AmazonEC2ContainerRegistryReadOnly" {
28   policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
29   role        = aws_iam_role.nodes.name
30 }
31
32 resource "aws_eks_node_group" "private-nodes" {
33   cluster_name      = aws_eks_cluster.demo.name
34   node_group_name  = "private-nodes"
35   node_role_arn    = aws_iam_role.nodes.arn
36
37   subnet_ids = [
38     aws_subnet.private-us-east-1a.id,
39     aws_subnet.private-us-east-1b.id
40   ]
41
42   capacity_type  = "ON_DEMAND"
43   instance_types = ["t3.small"]
44
45   scaling_config {
46     desired_size = 1
47     max_size     = 5
48     min_size     = 0
49   }
50
51   update_config {
52     max_unavailable = 1
53   }
54
55   labels = {
56     role = "general"
57   }
58
59   # taint {
60   #   key   = "team"
61   #   value = "devops"
62   #   effect = "NO_SCHEDULE"
63   # }
64
65   # launch_template {
66   #   name   = aws_launch_template.eks-with-disks.name
67   #   version = aws_launch_template.eks-with-disks.latest_version
68   # }
69
70   depends_on = [
71     aws_iam_role_policy_attachment.nodes-AmazonEKSWorkerNodePolicy,
72     aws_iam_role_policy_attachment.nodes-AmazonEKS_CNI_Policy,
73     aws_iam_role_policy_attachment.nodes-AmazonEC2ContainerRegistryReadOnly,
74   ]
75 }
76
77 # resource "aws_launch_template" "eks-with-disks" {
78 #   name = "eks-with-disks"

```

```

79  # ... key_name = "local-provisioner"
80
81  # ... block_device_mappings {
82  #     device_name = "/dev/xvdb"
83
84  #     ebs {
85  #         volume_size = 50
86  #         volume_type = "gp2"
87  #     }
88  # }
89
# }
```

## Create IAM OIDC provider EKS using Terraform

- To manage permissions for your applications that you deploy in Kubernetes. You can either attach policies to Kubernetes nodes directly. In that case, every pod will get the same access to AWS resources. Or you can create OpenID connect provider, which will allow granting IAM permissions based on the service account used by the pod. File name is `terraform/8-iam-oidc.tf`.

### 8-iam-oidc.tf

```

1  data "tls_certificate" "eks" {
2      url = aws_eks_cluster.demo.identity[0].oidc[0].issuer
3  }
4
5  resource "aws_iam_openid_connect_provider" "eks" {
6      client_id_list  = ["sts.amazonaws.com"]
7      thumbprint_list =
8      [data.tls_certificate.eks.certificates[0].sha1_fingerprint]
9      url            = aws_eks_cluster.demo.identity[0].oidc[0].issuer
}
```

- I highly recommend testing the provider first before deploying the autoscaler. It can save you a lot of time. File name is `terraform/9-iam-test.tf`.

### 9-iam-test.tf

```

1  data "aws_iam_policy_document" "test_oidc_assume_role_policy" {
2      statement {
3          actions = ["sts:AssumeRoleWithWebIdentity"]
4          effect  = "Allow"
5
6          condition {
7              test      = "StringEquals"
```

```

8     variable = "${replace(aws_iam_openid_connect_provider.eks.url,
9     "https://", "")}:sub"
10    values    = ["system:serviceaccount:default:aws-test"]
11  }
12
13 principals {
14   identifiers = [aws_iam_openid_connect_provider.eks.arn]
15   type        = "Federated"
16 }
17 }
18 }
19
20 resource "aws_iam_role" "test_oidc" {
21   assume_role_policy =
22   data.aws_iam_policy_document.test_oidc_assume_role_policy.json
23   name          = "test-oidc"
24 }
25
26 resource "aws_iam_policy" "test-policy" {
27   name = "test-policy"
28
29   policy = jsonencode({
30     Statement = [
31       Action = [
32         "s3:ListAllMyBuckets",
33         "s3:GetBucketLocation"
34       ]
35       Effect  = "Allow"
36       Resource = "arn:aws:s3:::/*"
37     ]
38     Version = "2012-10-17"
39   })
40 }
41
42 resource "aws_iam_role_policy_attachment" "test_attach" {
43   role      = aws_iam_role.test_oidc.name
44   policy_arn = aws_iam_policy.test-policy.arn
45 }
46
47 output "test_policy_arn" {
48   value = aws_iam_role.test_oidc.arn
49 }
```

- Now we can run terraform.

```
terraform apply
```

- To export Kubernetes context you can use `aws eks ...` command; just replace region and name of the cluster.

```
aws eks --region us-east-1 update-kubeconfig --name demo
```

- To check connection to EKS cluster run the following command:

```
kubectl get svc
```

- Next is to create a pod to test IAM roles for service accounts. First, we are going to omit annotations to bind the service account with the role. The way it works, you create a service account and use it in your pod spec. It can be anything, deployment, statefulset, or some jobs. Give it a name `k8s/aws-test.yaml`.

#### aws-test.yaml

```

1  ---
2  apiVersion: v1
3  kind: ServiceAccount
4  metadata:
5    name: aws-test
6    namespace: default
7  ---
8  apiVersion: v1
9  kind: Pod
10 metadata:
11   name: aws-cli
12   namespace: default
13 spec:
14   serviceAccountName: aws-test
15   containers:
16     - name: aws-cli
17       image: amazon/aws-cli
18       command: [ "/bin/bash", "-c", "--" ]
19       args: [ "while true; do sleep 30; done;" ]
20   tolerations:
21     - operator: Exists
22       effect: NoSchedule

```

- Then you need to apply it using `kubectl apply -f <folder/file>` command.

```
kubectl apply -f k8s/aws-test.yaml
```

- Now, let's check if can list S3 buckets in our account.

```
kubectl exec aws-cli -- aws s3api list-buckets
```

- Let's add missing annotation to the service account and redeploy the pod. Don't forget to replace `424432388155` with your AWS account number.

**aws-test.yaml**

```

1  ---
2  ...
3  annotations:
4    eks.amazonaws.com/role-arn: arn:aws:iam::424432388155:role/test-oidc
5  ...

```

```
kubectl delete -f k8s/aws-test.yaml
kubectl apply -f k8s/aws-test.yaml
```

- Try to list buckets again.

```
kubectl exec aws-cli -- aws s3api list-buckets
```

## Create public load balancer on EKS

- Next, let's deploy the sample application and expose it using public and private load balancers. The first is a deployment object with a base nginx image. File name is `k8s/deployment.yaml`.

**deployment.yaml**

```

1  ---
2  apiVersion: apps/v1
3  kind: Deployment
4  metadata:
5    name: nginx
6  spec:
7    replicas: 1
8    selector:
9      matchLabels:
10        app: nginx
11    template:
12      metadata:
13        labels:
14          app: nginx
15    spec:
16      containers:
17        - name: nginx
18          image: nginx:1.14.2
19        ports:

```

```

20      - name: web
21        containerPort: 80
22      resources:
23        requests:
24          memory: 256Mi
25          cpu: 250m
26        limits:
27          memory: 256Mi
28          cpu: 250m
29      affinity:
30        nodeAffinity:
31          requiredDuringSchedulingIgnoredDuringExecution:
32            nodeSelectorTerms:
33              - matchExpressions:
34                - key: role
35                  operator: In
36                  values:
37                    - general
38      # tolerations:
39      # - key: team
40      #   operator: Equal
41      #   value: devops
42      #   effect: NoSchedule

```

- To expose the application to the internet, you can create a Kubernetes service of a type load balancer and use annotations to configure load balancer properties. By default, Kubernetes will create a load balancer in public subnets, so you don't need to provide any additional configurations. Also, if you want a new network load balancer instead of the old classic load balancer, you can add `aws-load-balancer-type` equal to `nlb`. Call it `k8s/public-lb.yaml`.

### public-lb.yaml

```

1  ---
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: public-lb
6    annotations:
7      service.beta.kubernetes.io/aws-load-balancer-type: nlb
8  spec:
9    type: LoadBalancer
10   selector:
11     app: nginx
12   ports:
13     - protocol: TCP
14       port: 80
15       targetPort: web

```

- Create both deployment and the service objects.

```
kubectl apply -f k8s/deployment.yaml
kubectl apply -f k8s/public-lb.yaml
```

- Find load balancer in AWS console by name. Verify that LB was created in public subnets

## Create private load balancer on EKS

- Sometimes if you have a large infrastructure with many different services, you have a requirement to expose the application only within your VPC. For that, you can create a private load balancer. To make it private, you need additional annotation: aws-load-balancer-internal and then provide the CIDR range. Usually, you use 0.0.0.0/0 to allow any services within your VPC to access it. Give it a name `k8s/private-lb.yaml`.

### `private-lb.yaml`

```
1  ---
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: private-lb
6  annotations:
7    service.beta.kubernetes.io/aws-load-balancer-type: nlb
8    service.beta.kubernetes.io/aws-load-balancer-internal: 0.0.0.0/0
9  spec:
10    type: LoadBalancer
11    selector:
12      app: nginx
13    ports:
14      - protocol: TCP
15        port: 80
16        targetPort: web
```

- Let's go back to the terminal and apply it. You can grab the load balancer name and find it in the AWS console as well.

```
kubectl apply -f k8s/private-lb.yaml
```

- Find load balancer in AWS console by name. Verify that LB was created in private subnets

## Deploy EKS cluster autoscaler

- Finally, we got to the EKS autoscaler. We will be using OpenID connect provider to create an IAM role and bind it with the autoscaler. Let's create an IAM policy and role first. It's similar to

the previous one, but autoscaler will be deployed in the kube-system namespace. File name is `terraform/10-iam-autoscaler.tf`.

### 10-iam-autoscaler.tf

```

1  data "aws_iam_policy_document" "eks_cluster_autoscaler_assume_role_policy" {
2    statement {
3      actions = ["sts:AssumeRoleWithWebIdentity"]
4      effect  = "Allow"
5
6      condition {
7        test    = "StringEquals"
8        variable = "${replace(aws_iam_openid_connect_provider.eks.url,
9          "https://", "")}:sub"
10       values   = ["system:serviceaccount:kube-system:cluster-autoscaler"]
11     }
12
13     principals {
14       identifiers = [aws_iam_openid_connect_provider.eks.arn]
15       type        = "Federated"
16     }
17   }
18 }
19
20 resource "aws_iam_role" "eks_cluster_autoscaler" {
21   assume_role_policy =
22   data.aws_iam_policy_document.eks_cluster_autoscaler_assume_role_policy.json
23   name           = "eks-cluster-autoscaler"
24 }
25
26 resource "aws_iam_policy" "eks_cluster_autoscaler" {
27   name = "eks-cluster-autoscaler"
28
29   policy = jsonencode({
30     Statement = [
31       Action = [
32         "autoscaling:DescribeAutoScalingGroups",
33         "autoscaling:DescribeAutoScalingInstances",
34         "autoscaling:DescribeLaunchConfigurations",
35         "autoscaling:DescribeTags",
36         "autoscaling:SetDesiredCapacity",
37         "autoscaling:TerminateInstanceInAutoScalingGroup",
38         "ec2:DescribeLaunchTemplateVersions"
39       ]
40       Effect  = "Allow"
41       Resource = "*"
42     ]
43     Version = "2012-10-17"
44   })
45 }
46

```

```

47 resource "aws_iam_role_policy_attachment" "eks_cluster_autoscaler_attach" {
48   role        = aws_iam_role.eks_cluster_autoscaler.name
49   policy_arn = aws_iam_policy.eks_cluster_autoscaler.arn
50 }
51
52 output "eks_cluster_autoscaler_arn" {
53   value = aws_iam_role.eks_cluster_autoscaler.arn
54 }
```

- Let's apply the terraform again to create those objects.

```
terraform apply
```

- Let's create autoscaller itself. You can find the source code for autoscaller [here](#).
- Go back to the terminal and apply.

```
kubectl apply -f k8s/cluster-autoscaler.yaml
```

- You can verify that the autoscaler pod is up and running with the following command.

```
kubectl get pods -n kube-system
```

- It's a good practice to check logs for any errors.

```
kubectl logs -l app=cluster-autoscaler -n kube-system -f
```

## EKS cluster auto scaling demo

- Verify that AG (aws autoscaling group) has required tags:
  - `k8s.io/cluster-autoscaler/<cluster-name>` : owned
  - `k8s.io/cluster-autoscaler/enabled` : TRUE
- Split the terminal screen. In the first window run:

```
watch -n 1 -t kubectl get pods
```

- In the second window run:

```
watch -n 1 -t kubectl get nodes
```

- Now, to trigger autoscaling, increase replica for nginx deployment from 1 to 5.

```
kubectl apply -f k8s/deployment.yaml
```

 **Clean**



```
terraform destroy
```