Terraform: EKS Cluster Provision on AWS [10 Steps]

Terraform: EKS Cluster Provision on AWS [10 Steps]

Table of Contents

Related Searches: terraform aws kubernetes cluster, aws terraform kubernetes, terraform eks tutorial, terraform kubernetes cluster aws, terraform kubernetes example, terraform eks module

Elastic Kubernetes Service, EKS, is a managed Kubernetes service on AWS. This service provides Cloud Native Computing functionalities on AWS such as scalable and flexible application deployments on AWS.

Terraform is a tool used to automate infrastructure deployments on cloud environments. This tool implements the Infrastructure as Code (IaC) principles and Functionalities. Terraform is free and open source. There is an enterprise version of Terraform which contains more advanced features.

This article shall cover **how to deploy an EKS cluster on AWS using Terraform**. There are many ways to deploy an EKS cluster on AWS such as using the web console, AWS CLI, and CloudFormation. Below are some of the reasons why you would consider using Terraform over the other options.

You can manage the whole lifecycle of the EKS cluster through Terraform. You can do version updates, scale your cluster and resources under one place, Terraform. This is done seamlessly without the need for one to inspect the API to identify the resources.
Terraform will easily integrate with your AWS services, i.e, if there are some existing dependencies such as VPCs, Terraform will use them seamlessly without creating other parallel resources.
There is a streamlined workflow, if you are already using Terraform in your cloud environment, you can easily add the Terraform modules and proceed with the deployment for both the cluster and applications.

Deploy EKS Cluster Using Terraform

For us to successfully deploy our EKS Cluster on AWS using Terraform, we need to have set our environment with the following:

1. AWS account
2. AWS user account with IAM permissions necessary for EKS deployment
3. AWS CLI
4. kubectl

Step-1: Prepare Environment for EKS Deployment with Terraform

Follow the steps below to prepare the environment for EKS deployment on AWS using Terraform.

1.1: Setup AWS Account

Sign up for an AWS account to get a free AWS package for 12 months. This package, however, doesn't provide EKS service for free. You will be charged for this service.

Advertisement

When you have the account, you will be required to create a user with IAM permissions that allows the user to create and manage the EKS cluster. Ideally, this would be a System Admin account.

1.2: Create AWS Policy

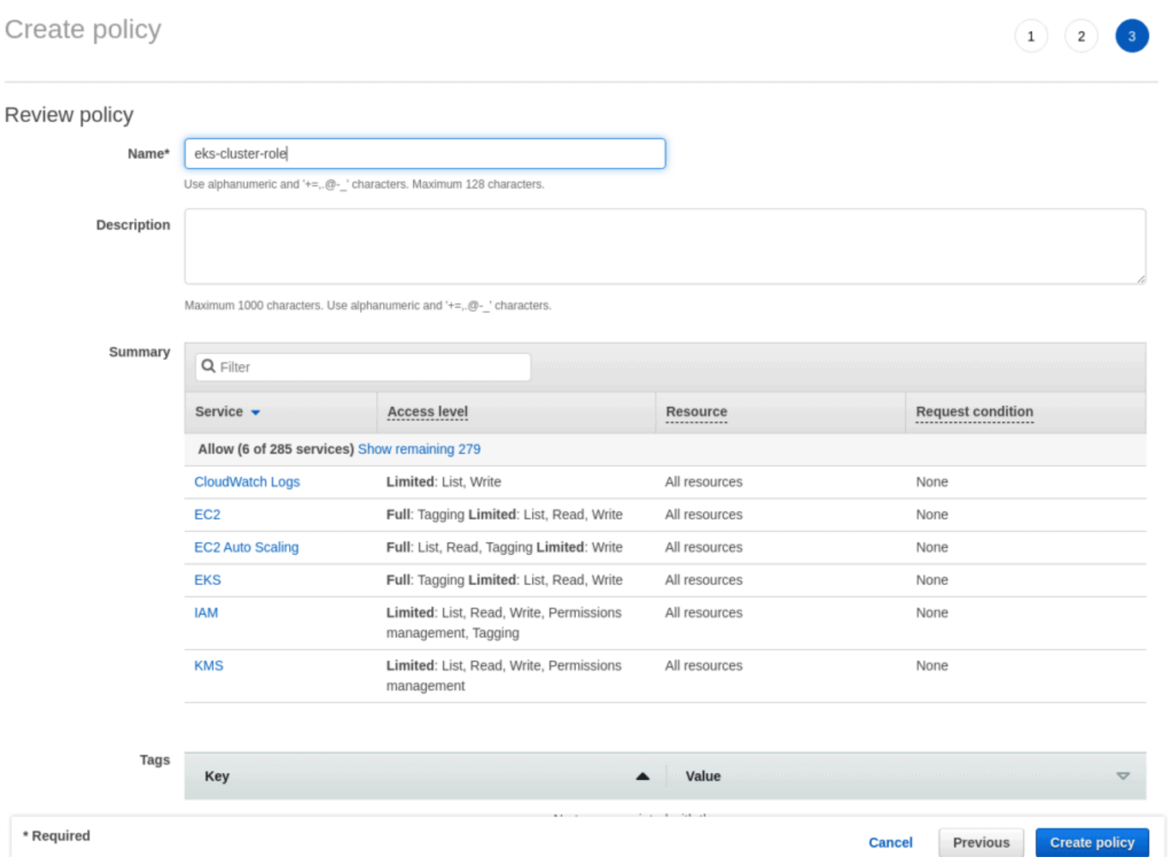
On the AWS console, go to **IAM** > **Policies** > **Create Policy**. Choose the JSON option then add the details below:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "autoscaling:AttachInstances",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:CreateOrUpdateTags",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:DeleteTags",
                "autoscaling:Describe*",
                "autoscaling:DetachInstances",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:SuspendProcesses",
                "ec2:AllocateAddress",
                "ec2:AssignPrivateIpAddresses",
                "ec2:Associate*",
                "ec2:AttachInternetGateway",
                "ec2:AttachNetworkInterface",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateDefaultSubnet",
                "ec2:CreateDhcpOptions",
                "ec2:CreateEgressOnlyInternetGateway",
                "ec2:CreateInternetGateway",
                "ec2:CreateNatGateway",
                "ec2:CreateNetworkInterface",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateVpc",
                "ec2:CreateVpcEndpoint",
                "ec2:DeleteDhcpOptions",
                "ec2:DeleteEgressOnlyInternetGateway",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteNatGateway",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteRoute",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteSubnet",
                "ec2:DeleteTags",
                "ec2:DeleteVolume",
                "ec2:DeleteVpc",
                "ec2:DeleteVpnGateway",
                "ec2:Describe*",
                "ec2:DetachInternetGateway",
                "ec2:DetachNetworkInterface",
                "ec2:DetachVolume",
                "ec2:Disassociate*",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:ModifyVpcEndpoint",
                "ec2:ReleaseAddress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:UpdateSecurityGroupRuleDescriptionsEgress",
                "ec2:UpdateSecurityGroupRuleDescriptionsIngress",
                "ec2:CreateLaunchTemplate",
                "ec2:CreateLaunchTemplateVersion",
                "ec2:DeleteLaunchTemplate",
                "ec2:DeleteLaunchTemplateVersions",
                "ec2:DescribeLaunchTemplates",
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:GetLaunchTemplateData",
                "ec2:ModifyLaunchTemplate",
                "ec2:RunInstances",
                "eks:CreateCluster",
                "eks:DeleteCluster",
                "eks:DescribeCluster",
                "eks:ListClusters",
                "eks:UpdateClusterConfig",
                "eks:UpdateClusterVersion",
                "eks:DescribeUpdate",
                "eks:TagResource",
                "eks:UntagResource",
                "eks:ListTagsForResource",
                "eks:CreateFargateProfile",
                "eks:DeleteFargateProfile",
                "eks:DescribeFargateProfile",
                "eks:ListFargateProfiles",
                "eks:CreateNodegroup",
                "eks:DeleteNodegroup",
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:UpdateNodegroupConfig",
                "eks:UpdateNodegroupVersion",
                "iam:AddRoleToInstanceProfile",
                "iam:AttachRolePolicy",
                "iam:CreateInstanceProfile",
                "iam:CreateOpenIDConnectProvider",
                "iam:CreateServiceLinkedRole",
                "iam:CreatePolicy",
                "iam:CreatePolicyVersion",
                "iam:CreateRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteOpenIDConnectProvider",
                "iam:DeletePolicy",
                "iam:DeletePolicyVersion",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:DeleteServiceLinkedRole",
                "iam:DetachRolePolicy",
                "iam:GetInstanceProfile",
                "iam:GetOpenIDConnectProvider",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:List*",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagOpenIDConnectProvider",
                "iam:TagRole",
                "iam:UntagRole",
                "iam:UpdateAssumeRolePolicy",
                "logs:CreateLogGroup",
                "logs:DescribeLogGroups",
                "logs:DeleteLogGroup",
                "logs:ListTagsLogGroup",
                "logs:PutRetentionPolicy",
                "kms:CreateAlias",
                "kms:CreateGrant",
                "kms:CreateKey",
                "kms:DeleteAlias",
                "kms:DescribeKey",
                "kms:GetKeyPolicy",
                "kms:GetKeyRotationStatus",
                "kms:ListAliases",
                "kms:ListResourceTags",
                "kms:ScheduleKeyDeletion"
            ],
            "Resource": "*"
        }
    ]
}
```

The above JSON grants the following permissions to the policy as described in the [EKS Terraform module documentation](#).

1. EC2 for elastic compute instance creation
2. EKS - full access to EKS features and services
3. CloudWatch for log analysis
4. IAM and KMS full permissions.

Click next and head over to the "**Review policy**" section where you assign the name to the policy.


Create policy

Review policy

Name*: eks-cluster-role

Use alphanumeric and '+=,.@-_' characters. Maximum 128 characters.

Description:

Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Summary

Service	Access level	Resource	Request condition
Allow (6 of 285 services) Show remaining 279			
CloudWatch Logs	**Limited**: List, Write	All resources	None
EC2	**Full**: Tagging **Limited**: List, Read, Write	All resources	None
EC2 Auto Scaling	**Full**: List, Read, Tagging **Limited**: Write	All resources	None
EKS	**Full**: Tagging **Limited**: List, Read, Write	All resources	None
IAM	**Limited**: List, Read, Write, Permissions management, Tagging	All resources	None
KMS	**Limited**: List, Read, Write, Permissions management	All resources	None

Tags

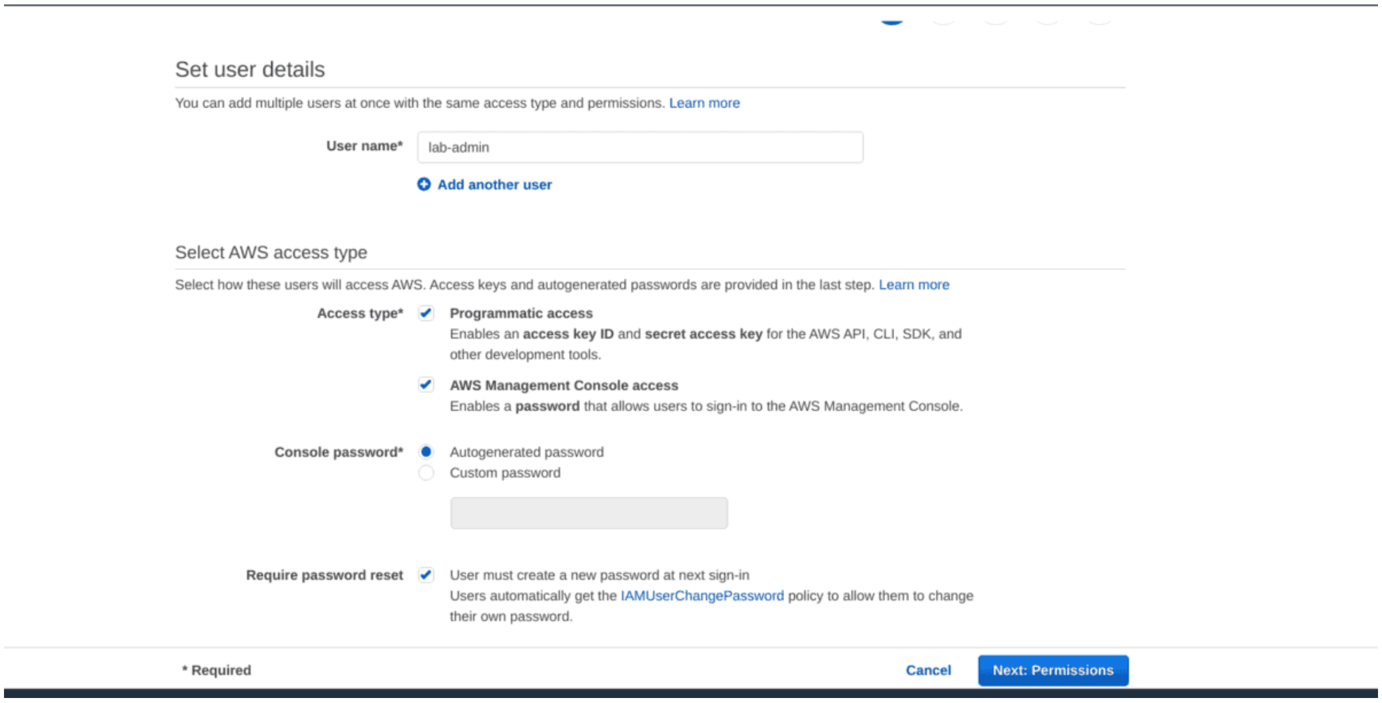
Key	Value

* Required

1.3: Create AWS User

Head over to **IAM** > **Users** > **Add users**.

Choose a username then check the "**Programmatic access**" and "**AWS Management Console access**" options.



Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name*: lab-admin

Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type*: [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[✓] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

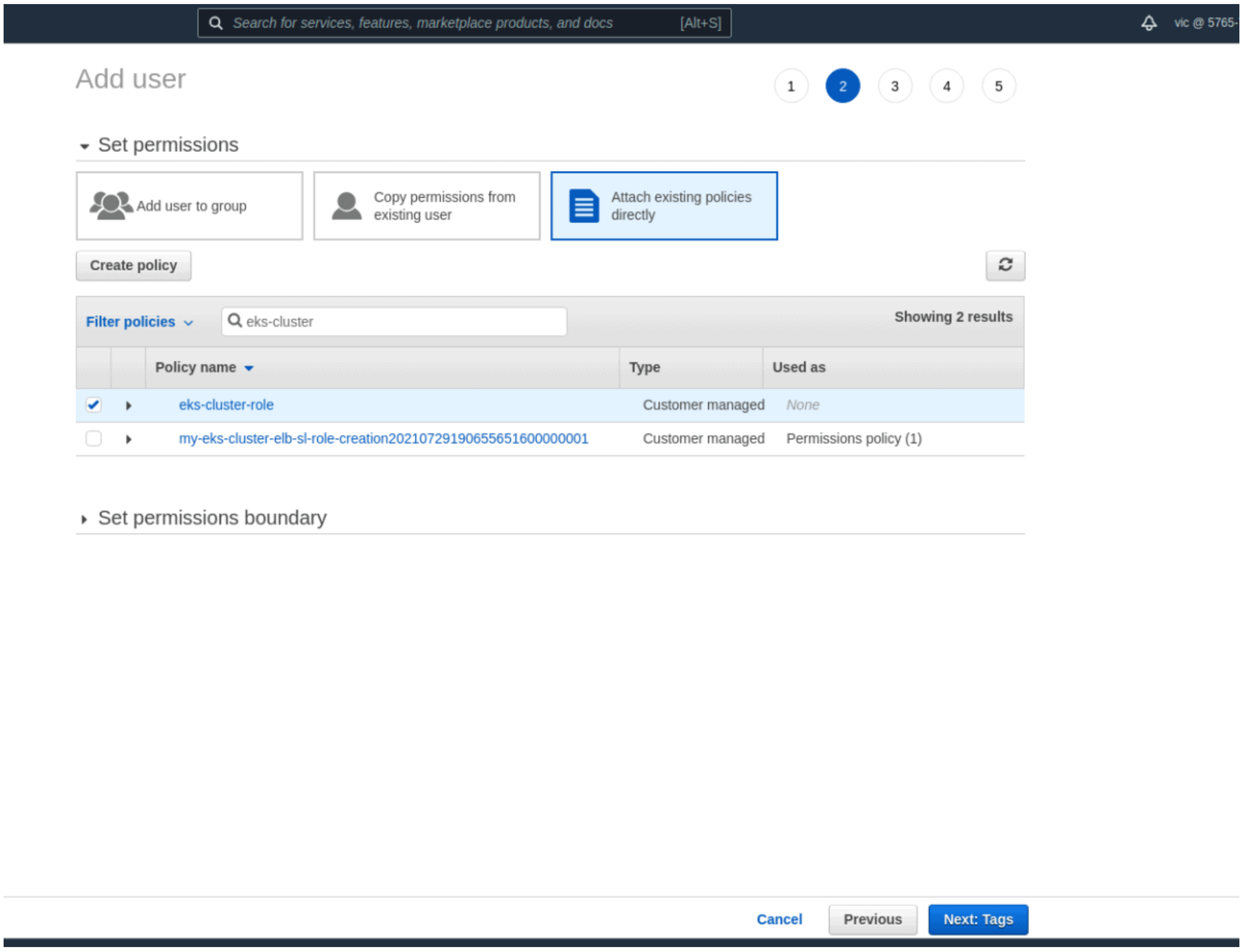
Console password*: [● Autogenerated password] ○ Custom password

Require password reset: [✓] User must create a new password at next sign-in
Users automatically get the IAMUserChangePassword policy to allow them to change their own password.

* Required

Cancel | Next: Permissions

Click **Next** to attach permissions to the user. Select the "**Attach existing policies directly**" option then search for the policy we created in the step above.


Add user

Set permissions

Add user to group | Copy permissions from existing user | Attach existing policies directly

Create policy

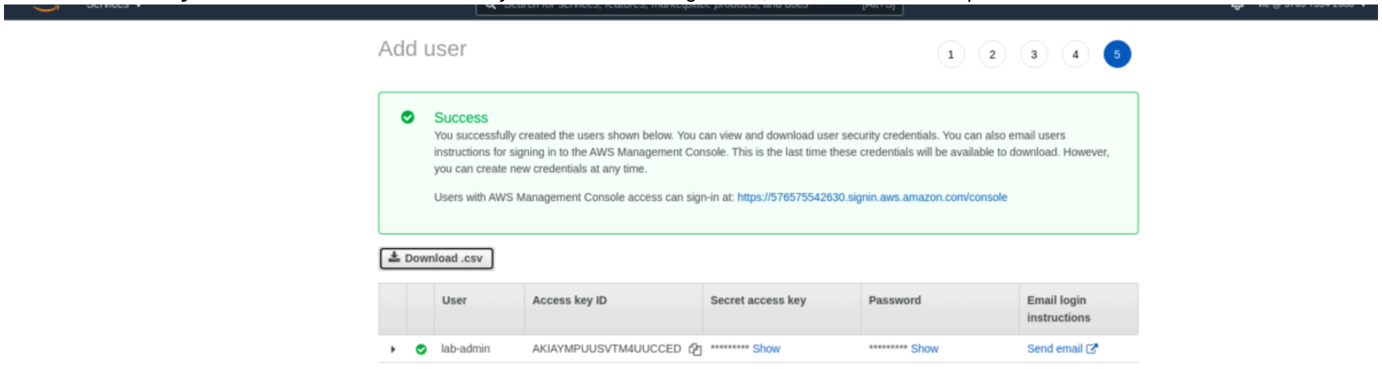
Filter policies: eks-cluster — Showing 2 results

		Policy name	Type	Used as
✓	▸	eks-cluster-role	Customer managed	None
	▸	my-eks-cluster-elb-sl-role-creation20210729190655651600000001	Customer managed	Permissions policy (1)

Set permissions boundary

Cancel | Previous | Next: Tags

Proceed to the next page to **review** and **finish** the user creation steps. **<u>Download the .csv file that contains the Access Key ID and the Secret Access Key</u>**. These two details are necessary for AWS CLI configuration, which is our next step.


Add user

Success
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: https://576575542630.signin.aws.amazon.com/console

Download .csv

		User	Access key ID	Secret access key	Password	Email login instructions
▸	✓	lab-admin	AKIAYMPUUSVTM4UUCCED	********* Show	********* Show	Send email

Step-2: Install AWS CLI

We need AWS CLI to manage our AWS environment from the command line. We shall cover how to install AWS CLI on a Linux environment. For the demonstration, I have brought up Ubuntu 20.04 server as a VM using Oracle VirtualBox on which I will be using AWS CLI to connect to AWS Servers and perform the deployment.

Download the latest version of AWS CLI version 2.

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
```

Extract the downloaded package:

```
$ unzip awscliv2.zip
```

Install AWS CLI

```
$ sudo ./aws/install
```

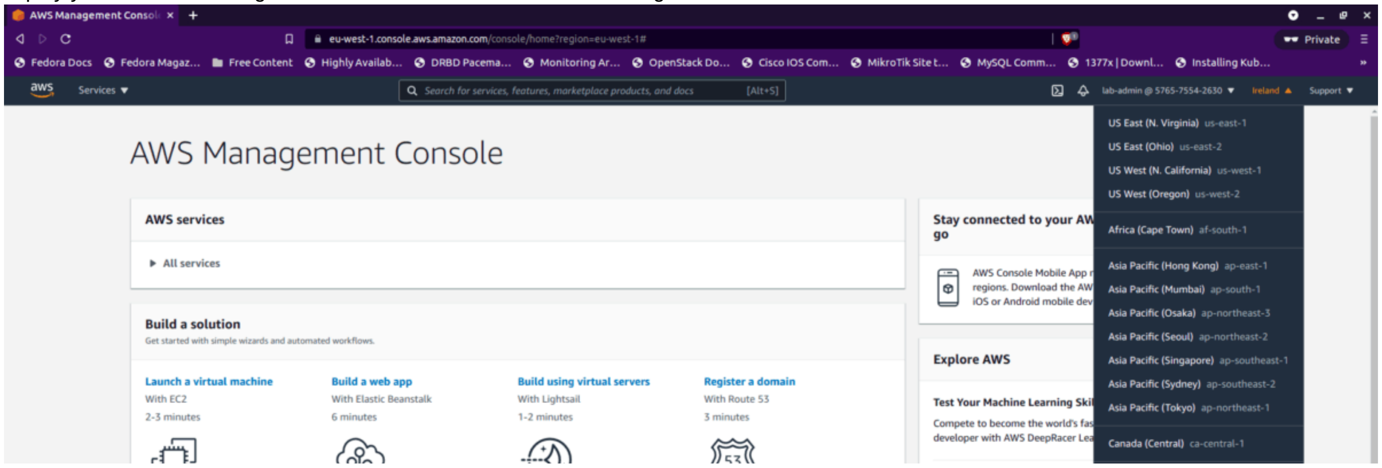
Step-3: Configure AWS CLI

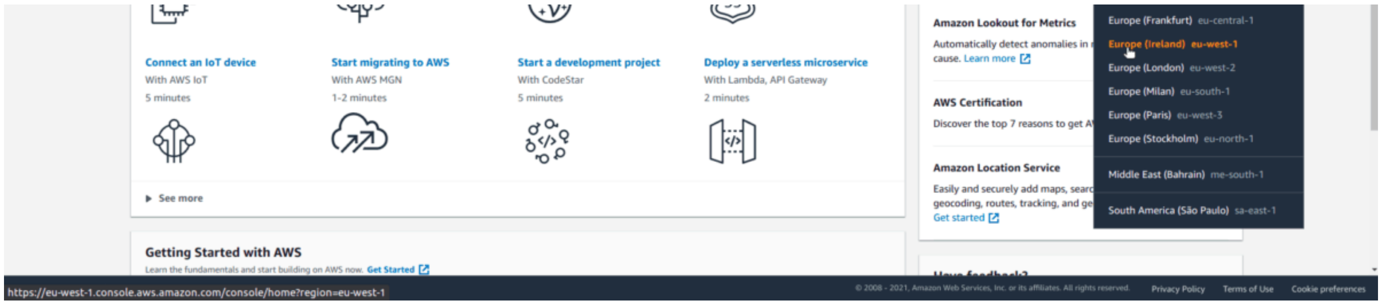
We need to configure AWS CLI so we can manage AWS on the command line. Run the commands below to set up AWS CLI. Provide the details we obtained from the AWS user creation step above.

The AWS Access Key ID and AWS Secret Access Key information is available in the .csv downloaded earlier. You also need to choose your preferred region for your deployments.

```
$ aws configure
AWS Access Key ID [None]: <your-AWS-USER-ACCESS-ID>
AWS Secret Access Key [None]: <YOUR-SECRET-KEY>
Default region name [None]: eu-west-1
Default output format [None]: json
```

To obtain the default region code, login to the AWS web console, go to the second top-right drop-down, and select the region you wish to deploy your cluster. The region code will also be shown beside the region name. This is as shown below:


AWS Management Console
eu-west-1.console.aws.amazon.com/console/home?region=eu-west-1#
Services
Search for services, features, marketplace products, and docs
lab-admin @ 5765-7554-2630
Ireland
Support
AWS Management Console
AWS services
All services
Build a solution
Launch a virtual machine
Build a web app
Build using virtual servers
Register a domain
Explore AWS
US East (N. Virginia) us-east-1
US East (Ohio) us-east-2
US West (N. California) us-west-1
US West (Oregon) us-west-2
Africa (Cape Town) af-south-1
Asia Pacific (Hong Kong) ap-east-1
Asia Pacific (Mumbai) ap-south-1
Asia Pacific (Osaka) ap-northeast-3
Asia Pacific (Seoul) ap-northeast-2
Asia Pacific (Singapore) ap-southeast-1
Asia Pacific (Sydney) ap-southeast-2
Asia Pacific (Tokyo) ap-northeast-1
Canada (Central) ca-central-1


Connect an IoT device
With AWS IoT
5 minutes
Start migrating to AWS
With AWS MGN
1-2 minutes
Start a development project
With CodeStar
5 minutes
Deploy a serverless microservice
With Lambda, API Gateway
2 minutes
See more
Getting Started with AWS
Learn the fundamentals and start building on AWS now. Get Started
Amazon Lookout for Metrics
Automatically detect anomalies in cause. Learn more
AWS Certification
Discover the top 7 reasons to get A
Amazon Location Service
Easily and securely add maps, searc geocoding, routes, tracking, and ge
Get started
Europe (Frankfurt) eu-central-1
Europe (Ireland) eu-west-1
Europe (London) eu-west-2
Europe (Milan) eu-south-1
Europe (Paris) eu-west-3
Europe (Stockholm) eu-north-1
Middle East (Bahrain) me-south-1
South America (São Paulo) sa-east-1
https://eu-west-1.console.aws.amazon.com/console/home?region=eu-west-1
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Cookie preferences

To verify if your credentials have been successfully set, run the command below:

```
$ aws sts get-caller-identity
```

You should get an output similar to this below:

```
{
    "UserId": "AIDAYMPUUSVTMRUXXXXX",
    "Account": "57657XXXXXX",
    "Arn": "arn:aws:iam::57657XXXXXX:user/lab-admin"
}
```

Step-4: Install Kubectl

Install `kubectl` that will be used to manage your Kubernetes cluster once the setup is done. This guide shall cover how to install `kubectl` on Linux.

Download the latest release of `kubectl`:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io
/release/stable.txt)/bin/linux/amd64/kubectl"
```

Check and confirm the version of the installed `kubectl`

```
$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.
21.3", GitCommit:"ca643a4d1f7b
```

Installation guides for other Operating Systems are available on the [Kubernetes website.](https://kubernetes.io)

Step-5: Install Terraform

Before we can use Terraform, we need to install it. This guide will focus on Terraform installation on Linux platforms. Download the latest version of Terraform from the GitHub release page:

```
$ TER_VER=`curl -s https://api.github.com/repos/hashicorp/terraform
/releases/latest | grep tag_name | cut -d: -f2 | tr -d \"\,\v | awk
'{$1=$1};1'`

$ wget https://releases.hashicorp.com/terraform/${TER_VER}
/terraform_${TER_VER}_linux_amd64.zip
```

Extract the downloaded file then move the binary file to `/usr/local/bin` directory

```
$ unzip terraform_${TER_VER}_linux_amd64.zip
Archive:  terraform_xxx_linux_amd64.zip
 inflating: terraform

$ sudo mv terraform /usr/local/bin/
```

Check and confirm the version of Terraform you have just installed

```
$ teraform -v
```

Step-6: Setup Terraform Workspace

We now have all the necessary tools to set up our EKS Cluster on AWS. We now need to create the required file for Terraform to create our cluster.

The files needed are:

1. `eks-cluster.tf` – holds the cluster resources such as the worker nodes.
2. `security-groups.tf` – holds the information about the cluster subnets and VPC details
3. `versions.tf` – holds the information about the needed provider versions.
4. `k8s.tf` – holds the information about the Kubernetes provider
5. `vpc.tf` – contains VPC resource information
6. `outputs.tf` – contains the desired outputs when the deployment is finished.

6.1: Setup Terraform Environment

Create a directory that shall be used as your working directory.

```
$ mkdir -p ~/terraform-deployments && cd ~/terraform-deployments
```

6.2: Create an EKS Cluster configuration file

Create the `eks-cluster.tf`file and add the content below. My deployment has three worker nodes, you could choose to have more or less.

This file contains the following information:

1. The terraform module used to setup the EKS cluster
2. The EKS cluster version
3. Details about the worker nodes such as the volume type, the size of the instances, and the number of nodes
4. The subnets to be used for the EKS cluster

```
$ vim eks-cluster.tf


module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  cluster_name    = local.cluster_name
  cluster_version = "1.20"
  subnets         = module.vpc.private_subnets

  tags = {
    Environment = "development"
    GithubRepo  = "terraform-aws-eks"
    GithubOrg   = "terraform-aws-modules"
  }


  vpc_id = module.vpc.vpc_id

  workers_group_defaults = {
    root_volume_type = "gp2"
  }

cluster_endpoint_private_access = "true"
  cluster_endpoint_public_access  = "true"

  write_kubeconfig      = true
  manage_aws_auth       = true

  worker_groups = [
    {
      name                          = "worker-group-1"
      instance_type                 = "t2.small"
      asg_desired_capacity          = 2
      asg_max_size                  = 5
      asg_min_size                  = 2
      additional_userdata           = "echo foo bar"
      asg_desired_capacity          = 2
      additional_security_group_ids = [aws_security_group.
worker_group_mgmt_one.id]
    },
    {
      name                          = "worker-group-2"
      instance_type                 = "t2.micro"
      asg_desired_capacity          = 1
      asg_max_size                  = 5
      asg_min_size                  = 1
      additional_userdata           = "echo foo bar"
      additional_security_group_ids = [aws_security_group.
worker_group_mgmt_two.id]
      asg_desired_capacity          = 1
    },
  ]
}

data "aws_eks_cluster" "cluster" {
  name = module.eks.cluster_id
}

data "aws_eks_cluster_auth" "cluster" {
  name = module.eks.cluster_id
}
```

6.3: Create VPC Configuration File

Create the `vpc.tf` file to provision the VPC and the subnets. In the below code, take note of the **region**, **cluster name**, and the **VPC subnets**. You should consider using your custom details depending on how you have set up your AWS environment.

In this file, we shall configure the following:

1. The default region for our VPC
2. The name of the EKS cluster
3. The subnets for both private and public subnets.

```
$ vim vpc.tf


variable "region" {
  default     = "us-east-1"
  description = "AWS region"
}

provider "aws" {
  region = "us-east-1"
}

data "aws_availability_zones" "available" {}

locals {
  cluster_name = "my-eks-cluster"
}


module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.66.0"
  name                 = "my-eks-cluster-vpc"
  cidr                 = "10.0.0.0/16"
  azs                  = data.aws_availability_zones.available.names
  private_subnets      = ["10.0.11.0/24", "10.0.22.0/24", "10.0.33.0
/24"]
  public_subnets       = ["10.0.44.0/24", "10.0.55.0/24", "10.0.66.0
/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true

  tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }

  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = "1"
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = "1"
  }
}
  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = "1"
  }
}
```

6.4: Create AWS Security Groups

Create security groups using the `security-groups.tf` configuration file. The security groups will contain the rules on how to access different subnets in the VPC. In this configuration, ingress traffic for SSH service is allowed on port 22 for the three subnets. This is for the management of the worker nodes of our EKS cluster.

```
$ vim security-groups.tf


resource "aws_security_group" "worker_group_mgmt_one" {
  name_prefix = "worker_group_mgmt_one"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"

    cidr_blocks = [
      "10.0.0.0/8",
    ]
  }
}

resource "aws_security_group" "worker_group_mgmt_two" {
  name_prefix = "worker_group_mgmt_two"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"

    cidr_blocks = [
      "192.168.0.0/16",
    ]
  }
}

resource "aws_security_group" "all_worker_mgmt" {
  name_prefix = "all_worker_management"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"

    cidr_blocks = [
      "10.0.0.0/8",
      "172.16.0.0/12",
      "192.168.0.0/16",
    ]
  }
}
```

6.5: Configure Terraform Providers

In the `versions.tf`file, we shall configure the required providers and their respective versions.

```
$ vim versions.tf


terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 3.20.0"
    }

    random = {
      source  = "hashicorp/random"
      version = "3.0.0"
    }

    local = {
      source  = "hashicorp/local"
      version = "2.0.0"
    }

    null = {
      source  = "hashicorp/null"
      version = "3.0.0"
    }

    template = {
      source  = "hashicorp/template"
      version = "2.2.0"
    }

    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = ">= 2.0.1"
    }
  }

  required_version = "> 0.14"
}
```

Add k8s provider:

```
$ vim k8s.tf


provider "kubernetes" {
  host                   = data.aws_eks_cluster.cluster.endpoint
  token                  = data.aws_eks_cluster_auth.cluster.token
  cluster_ca_certificate = base64decode(data.aws_eks_cluster.cluster.
certificate_authority.0.data)
}
```

6.6: Configure Terraform Outputs

Configure the required outputs after the deployment. These outputs include:

1. The cluster ID
2. The cluster endpoint
3. Cluster name

These will be useful for the configuration and management of our EKS cluster once fully deployed.

```
$ vim outputs.tf


output "cluster_id" {
  description = "EKS cluster ID."
  value       = module.eks.cluster_id
}

output "cluster_endpoint" {
  description = "Endpoint for EKS control plane."
  value       = module.eks.cluster_endpoint
}

output "cluster_security_group_id" {
  description = "Security group ids attached to the cluster control
plane."
  value       = module.eks.cluster_security_group_id
}

output "kubectl_config" {
  description = "kubectl config as generated by the module."
  value       = module.eks.kubeconfig
}

output "config_map_aws_auth" {
  description = "A kubernetes configuration to authenticate to this EKS
cluster."
  value       = module.eks.config_map_aws_auth
}

output "region" {
  description = "AWS region"
  value       = var.region
}

output "cluster_name" {
  description = "Kubernetes Cluster Name"
  value       = local.cluster_name
}
```

Step-7: Initialize Terraform Workspace

Initialize Terraform workspace to download the required providers.

```
$ terraform init
```

The workspace initialization should take some moments before it’s done.

```
$ terraform init
Upgrading modules...
Downloading terraform-aws-modules/eks/aws 17.1.0 for eks...
- eks in .terraform/modules/eks
- eks.fargate in .terraform/modules/eks/modules/fargate
- eks.node_groups in .terraform/modules/eks/modules/node_groups
Downloading terraform-aws-modules/vpc/aws 2.66.0 for vpc...
- vpc in .terraform/modules/vpc

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/kubernetes versions matching ">= 1.11.1, >=
2.0.1"...
- Finding hashicorp/aws versions matching ">= 2.68.0, >= 3.20.0, >=
3.40.0, >= 3.43.0"...
- Finding hashicorp/random versions matching "3.0.0"...
- Finding hashicorp/local versions matching ">= 1.4.0, 2.0.0"...
- Finding hashicorp/null versions matching "3.0.0"...
- Finding hashicorp/template versions matching "2.2.0"...
- Finding terraform-aws-modules/http versions matching ">= 2.4.1"...
- Finding latest version of hashicorp/cloudinit...
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)
- Installing terraform-aws-modules/http v2.4.1...
- Installed terraform-aws-modules/http v2.4.1 (self-signed, key ID
B2C1C0641B6B0EB7)
- Installing hashicorp/cloudinit v2.2.0...
- Installed hashicorp/cloudinit v2.2.0 (signed by HashiCorp)
- Installing hashicorp/kubernetes v2.3.2...
- Installed hashicorp/kubernetes v2.3.2 (signed by HashiCorp)
- Installing hashicorp/aws v3.51.0...
- Installed hashicorp/aws v3.51.0 (signed by HashiCorp)
- Installing hashicorp/random v3.0.0...
- Installed hashicorp/random v3.0.0 (signed by HashiCorp)
- Installing hashicorp/local v2.0.0...
- Installed hashicorp/local v2.0.0 (signed by HashiCorp)
- Installing hashicorp/null v3.0.0...
- Installed hashicorp/null v3.0.0 (signed by HashiCorp)
```

```
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about
it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has made some changes to the provider dependency selections
recorded
in the .terraform.lock.hcl file. Review those changes and commit them
to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan"
to see
any changes that are required for your infrastructure. All Terraform
commands
should now work.

If you ever set or change modules or backend configuration for
Terraform,
rerun this command to reinitialize your working directory. If you
forget, other
commands will detect it and remind you to do so if necessary.
```

You can now proceed with the deployment since the environment has been successfully initialized.

Run the `terraform plan` command to confirm the resources that will be provisioned when we run our script.

```
$ terraform plan
....
Plan: 50 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + cluster_endpoint          = (known after apply)
  + cluster_id                = (known after apply)
  + cluster_name              = "my-eks-cluster"
  + cluster_security_group_id = (known after apply)
  + config_map_aws_auth       = [
      + {
          + binary_data = null
          + data        = (known after apply)
          + id          = (known after apply)
          + metadata    = [
              + {
                  + annotations      = null
                  + generate_name    = null
                  + generation       = (known after apply)
                  + labels           = {
                      + "app.kubernetes.io/managed-by" = "Terraform"
                      + "terraform.io/module"          = "terraform-aws-
modules.eks.aws"
                    }
                  + name              = "aws-auth"
                  + namespace         = "kube-system"
                  + resource_version = (known after apply)
                  + uid               = (known after apply)
                },
            ]
        },
    ]
  + kubectl_config            = (known after apply)
  + region                    = "us-"
```

We are now aware that terraform will make 50 changes to our environment, each change has been described in the output above.

Step-8: Terraform EKS Cluster Provisioning

We will proceed to provision the EKS cluster using Terraform with the command below:

```
$ terraform apply
```

You will be asked to confirm if Terraform will proceed with the deployment. Type ‘`yes`’.

```
$ terraform apply
.....
Plan: 52 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + cluster_endpoint          = (known after apply)
  + cluster_id                = (known after apply)
  + cluster_name              = "my-eks-cluster"
  + cluster_security_group_id = (known after apply)
  + config_map_aws_auth       = [
      + {
          + binary_data = null
          + data        = (known after apply)
          + id          = (known after apply)
          + metadata    = [
              + {
                  + annotations      = null
                  + generate_name    = null
                  + generation       = (known after apply)
                  + labels           = {
                      + "app.kubernetes.io/managed-by" = "Terraform"
                      + "terraform.io/module"          = "terraform-aws-
modules.eks.aws"
                    }
                  + name              = "aws-auth"
                  + namespace         = "kube-system"
                  + resource_version = (known after apply)
                  + uid               = (known after apply)
                },
            ]
        },
    ]
  + kubectl_config             = (known after apply)
  + region                     = "us-east-1"

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

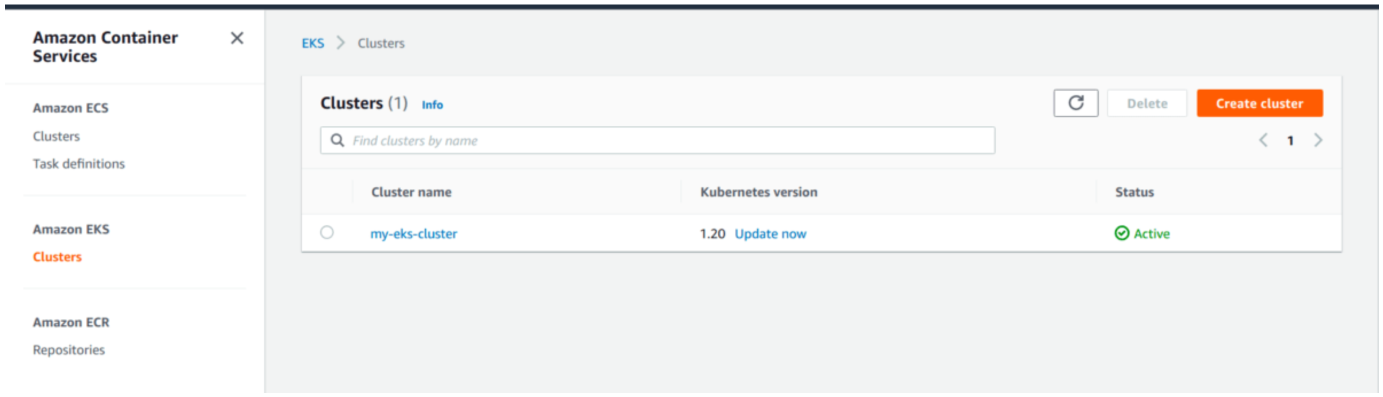
This initializes the provisioning on AWS. The process takes some minutes to complete.

You will then get the output showing you that the provisioning was successful.

```
Apply complete! Resources: 50 added, 0 changed, 0 destroyed.

Outputs:

cluster_endpoint = "https://AC2EC56CB185C351B64gffC421D4E246C58.gr7.us-
east-1.eks.amazonaws.com"
cluster_id = "my-eks-cluster"
cluster_name = "my-eks-cluster"
cluster_security_group_id = "sg-0c05b3ffe3c5110f8e9"
config_map_aws_auth = [
  {
    "binary_data" = tomap(null) /* of string */
    "data" = tomap({
      "mapAccounts" = <<-EOT
      []

      EOT
      "mapRoles" = <<-EOT
      - "groups":
        - "system:bootstrappers"
        - "system:nodes"
        "rolearn": "arn:aws:iam::788210555522308:role/my-eks-
cluster20211212130252973300000000e"
        "username": "system:node:{{EC2PrivateDNSName}}"

      EOT
      "mapUsers" = <<-EOT
      []

      EOT
    })
    "id" = "kube-system/aws-auth"
    "metadata" = tolist([
      {
        "annotations" = tomap(null) /* of string */
        "generate_name" = ""
        "generation" = 0
        "labels" = tomap({
          "app.kubernetes.io/managed-by" = "Terraform"
          "terraform.io/module" = "terraform-aws-modules.eks.aws"
        })
        "name" = "aws-auth"
        "namespace" = "kube-system"
        "resource_version" = "944"
        "uid" = "fa62487a-5891-448a-a8db-84ad01f8e39a"
      },
    ])
  },
]
kubectl_config = <<EOT
apiVersion: v1
preferences: {}
kind: Config

clusters:
- cluster:
    server: https://AC2EC56CB185fffC351B64C421D4E246C58.gr7.us-east-1.
eks.amazonaws.com
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM1KJGFNDQWMrZ0F3SUJBZ0lCQURBTk
Jna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRF
RJeE1USXhNakV5TlRVek1Wb1hEVE14TVRJeE1ERXlOVFV6TVZvd0ZURVRNQkVHQTFVRQpBeE
1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2
dnRUJBTFc0CjU4RGVzL0M0aDlaNUZWbjQ4cis2YzhEZU0zcUlnV1dvSGRWZEt4RTdPODY2eG
E0VWg4aEcyU0I2NUJ1bzJLb0QKeExxmRkNoeXlieU55N21seE5Fd3I3bXdzQ0svZXZQWm83aG
FGb0NLa3libWd5R3pVelpPOXcxTDBSRi9BQjYrYQpTZzNGREhaeVpCbjExRVJ5Q0RhQVJXZW
RCVTBxa2Vya0E3S0ovM3MxSWNUL1VKbnAxWDNaTlJYMm55dEVrWnJGCk5Rck9taGhQeU4wK0
FBL2QxbVJxemFhaFM1VHFBYUpsdTV5YXVTV1JGaFU0cTlaWnlBM2M1alBnZzFQelBkZlEKeT
VoZHVOWXAwMzJiOTVZbVNFWkZmSUo4bGU5MGFiWk4vQ0kvSE1VdjJKT2NPcmFOYU9XS2NLUU
hRZExRZENkMApWbWh6SlNEaW16UVpoSDJ5UmtzQ0F3RUFBYU5DTUVBd0RnWURWUjBQQVFIL0
JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZOWktvSjNyZ3
dEUnorK1cvZXlod1pKVG1aTUFNQTBHQ1NxR1NJYjMKRFFFQkN3VUFBNElCQVFBUUhrY1dDU1
ZadVJ5MnMzWkdnWTZTVHBNVGJ0eXN1SWRpbGx3MjRYT1ZFVnNXdFBObgpKc2FNcVFsOEtPbF
ZEajMrUGw1K1NZMVJoZHdyT2VOeEZ0MFc4YldjRzZIa3RPbU5rWTNhNEdTVVo5WFhsK01lCj
BXbjFZZUhMTEwvdWd5UnhrMCtpSHZHZjZlZ2ViT2pFV2tIQ0pTYzBncWkrL3pYM2I1aitXRD
lZQXcyQTIydU0KWmxXWHdvaldyUnplVEdQQ1lWV2JvUHF5QlYvYlhLWG9ZNlBTbDRYSVFCan
JjZ0JsbEw2ekxDSFMxaTNSZkxLVApMcVUvSUVLbUEvWFZIdkZ5a2pjSGZaWWVhY2xuQ0I5Qn
```

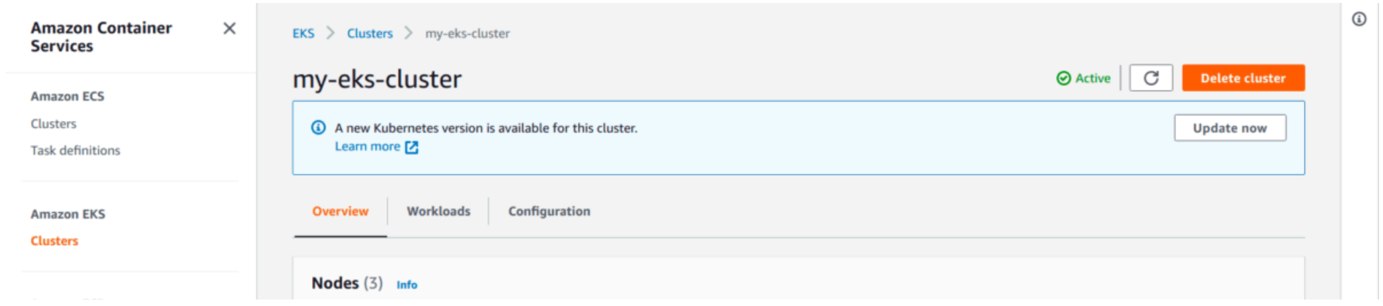
```
N0bUlJRVdFNHhpUVYzaUJXQm8rUUI3SUJVb0QzN3NiCkN5L2Zrb3ZXN0dtczNwYk13anJwem
5PVzNyV1ZPbG5saXBBUQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  name: eks_my-eks-cluster

contexts:
- context:
    cluster: eks_my-eks-cluster
    user: eks_my-eks-cluster
  name: eks_my-eks-cluster

current-context: eks_my-eks-cluster

users:
- name: eks_my-eks-cluster
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "my-eks-cluster"

EOT
region = "us-east-1"
```

You can also log in to the AWS web console to confirm if the cluster really exists. Navigate to **Elastic Kubernetes service** > **Amazon EKS** > **Clusters**.

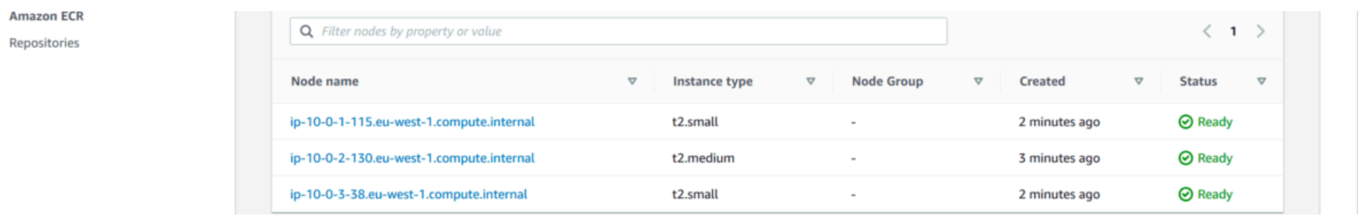


Cluster name	Kubernetes version	Status
my-eks-cluster	1.20 Update now	Active

We can verify that the cluster with the name `my-eks-cluster` is now available. We can also check the available worker nodes in the cluster.



A new Kubernetes version is available for this cluster.
Learn more



Node name	Instance type	Node Group	Created	Status
ip-10-0-1-115.eu-west-1.compute.internal	t2.small	-	2 minutes ago	Ready
ip-10-0-2-130.eu-west-1.compute.internal	t2.medium	-	3 minutes ago	Ready
ip-10-0-3-38.eu-west-1.compute.internal	t2.small	-	2 minutes ago	Ready

Step-9: Export EKS kubeconfig to manage Kubernetes Cluster

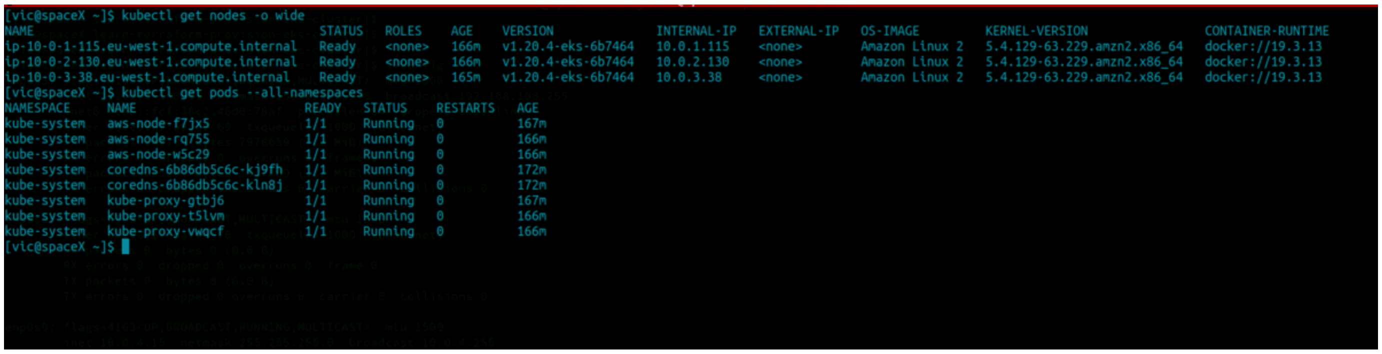
To manage our EKS cluster on CLI, we need for configuring `kubectl` context by importing the EKS `kubeconfig` as below:

```
$ aws eks update-kubeconfig --name my-eks-cluster --region us-east-1
```

The above command exports the `eks kubeconfig` and you can now manage your [Kubernetes cluster]() using kubectl.

We can run some basic commands for Kubernetes to confirm this.

```
$ kubectl get nodes

$ kubectl get pods --all-namespaces
```



```
[vic@spaceX ~]$ kubectl get nodes -o wide
NAME                                       STATUS   ROLES    AGE    VERSION               INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION                  CONTAINER-RUNTIME
ip-10-0-1-115.eu-west-1.compute.internal   Ready    <none>   166m   v1.20.4-eks-6b7464    10.0.1.115    <none>        Amazon Linux 2   5.4.129-63.229.amzn2.x86_64     docker://19.3.13
ip-10-0-2-130.eu-west-1.compute.internal   Ready    <none>   166m   v1.20.4-eks-6b7464    10.0.2.130    <none>        Amazon Linux 2   5.4.129-63.229.amzn2.x86_64     docker://19.3.13
ip-10-0-3-38.eu-west-1.compute.internal    Ready    <none>   165m   v1.20.4-eks-6b7464    10.0.3.38     <none>        Amazon Linux 2   5.4.129-63.229.amzn2.x86_64     docker://19.3.13
[vic@spaceX ~]$ kubectl get pods --all-namespaces
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE
kube-system   aws-node-f7jx5             1/1     Running   0          167m
kube-system   aws-node-rq755             1/1     Running   0          166m
kube-system   aws-node-w5c29             1/1     Running   0          166m
kube-system   coredns-6b86db5c6c-kj9fh   1/1     Running   0          172m
kube-system   coredns-6b86db5c6c-kln8j   1/1     Running   0          172m
kube-system   kube-proxy-gtbj6           1/1     Running   0          167m
kube-system   kube-proxy-t5lvm           1/1     Running   0          166m
kube-system   kube-proxy-vwqcf           1/1     Running   0          166m
[vic@spaceX ~]$
```

Step-10: Destroy terraform

You can also destroy your stack using Terraform with the command below:

```
$ terraform destroy
```

This will ask you to confirm if you really want to completely destroy the stack as the process is irreversible.

```
....
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown
```

```
above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

Summary

That’s it! You now have your cluster configured and you can now start deploying your Cloud Native applications to the cloud using EKS. In my opinion, setting up an EKS cluster using Terraform is much easier and straightforward than using any other method. This has been demonstrated in the above steps.

Feel free to get in touch in case you experience any issues deploying EKS on AWS using Terraform.