Kubernetes on AWS: EKS Deployment Best Practices with CloudFormation, Terraform, and eksctl

Check GitHub for helpful DevOps tools:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Ask Personal AI Document assistant to learn interactively (FASTER)!

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesAWS.pdf

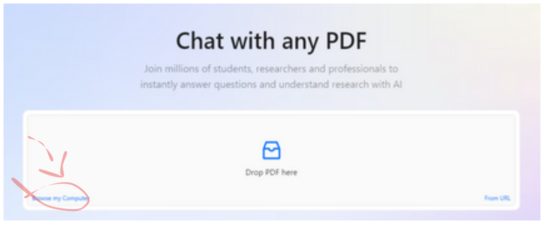
1 Download PDF

2 Go to website

Click there to go to ChatPdf website 2

3 Browse file

3


Chat with any PDF
Join millions of students, researchers and professionals to instantly answer questions and understand research with AI
Drop PDF here
Browse my Computer
From URL

4 Chat with Document

Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes AWS: Deploy EKS with CloufFormation

1) What is Cloud Formation

AWS CloudFormation enables you to define and provision infrastructure as code using declarative templates. It manages resources as stacks, supports version control, automates provisioning, and simplifies replicating infrastructure across environments.

2) Cloud formation yaml sections

Parameters:

This section defines the parameters that can be customized when deploying the CloudFormation stack, such as the cluster name, VPC ID, subnet IDs, key pair, node group name, instance type, and desired capacity.

```
Parameters:

  VpcBlock:
    Type: String
    Default: 192.168.0.0/16
    Description: The CIDR range for the VPC. This should be a valid private (RFC 1918) CIDR
range.
  PublicSubnet01Block:
    Type: String
    Default: 192.168.0.0/18
    Description: CidrBlock for public subnet 01 within the VPC
  PrivateSubnet01Block:
    Type: String
    Default: 192.168.128.0/18
    Description: CidrBlock for private subnet 01 within the VPC
```

Resources:

This section defines the AWS resources required for the EKS cluster. It includes the EKS cluster itself, the cluster's role, security group, node group, and the role for the worker nodes etc.

```
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock:  !Ref VpcBlock
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-VPC'

  InternetGateway:
    Type: "AWS::EC2::InternetGateway"

  VPCGatewayAttachment:
    Type: "AWS::EC2::VPCGatewayAttachment"
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC
[...]
```

Outputs:

This section defines the outputs of the CloudFormation stack, which provide information about the created EKS cluster and node group.

```
Outputs:
  ClusterNameOutput:
    Description: Name of the EKS cluster
    Value: !Ref ClusterName
  NodeGroupNameOutput:
    Description: Name of the EKS node group
    Value: !Ref NodeGroupName
[...]
```

3) Create underlying EKS infrastructure: VPC

Download cloudformation VPC configuration yaml. It includes subnets, routes, internet Gateway and NAT.

```
curl -O
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/EKS/CloudFormation/
eks-vpc-stack.yaml
```

VPC

VPC with a specified IP range, enabling DNS support and hostnames for instances.

```
VPC:
  Type: AWS::EC2::VPC
  Properties:
    CidrBlock: !Ref VpcBlock
    EnableDnsSupport: true
    EnableDnsHostnames: true
```

Public Subnet:

PublicSubnet01 (192.168.0.0/18 as defined in PublicSubnet01Block) automatically assigns public IPs to instances. Subnet is tagged for Elastic Load Balancers. It helps ELBs identify it for internet-facing traffic. The subnet is created in the first AZ of the AWS region.

```
PublicSubnet01Block:
  Type: String
  Default: 192.168.0.0/18
  Description: CidrBlock for public subnet 01 within the VPC
```

```
PublicSubnet01:
  Type: AWS::EC2::Subnet
  Metadata:
    Comment: Subnet 01
  Properties:
    MapPublicIpOnLaunch: true
    AvailabilityZone:
      Fn::Select:
      - '0'
      - Fn::GetAZs:
          Ref: AWS::Region
    CidrBlock:
      Ref: PublicSubnet01Block
    VpcId:
      Ref: VPC
    Tags:
    - Key: Name
      Value: !Sub "${AWS::StackName}-PublicSubnet01"
    - Key: kubernetes.io/role/elb
```

Internet gateway

Internet Gateway get attached to a specified VPC, enabling internet access for the VPC's resources.

```
InternetGateway:
    Type: "AWS::EC2::InternetGateway"
```

```
VPCGatewayAttachment:
  Type: "AWS::EC2::VPCGatewayAttachment"
  Properties:
    InternetGatewayId: !Ref InternetGateway
    VpcId: !Ref VPC
```

Public Subnet and RouteTable associations

The PublicSubnet01RouteTableAssociation links PublicSubnet01 to the PublicRouteTable, enabling internet access by applying the route rules to the subnet.

```
PublicSubnet01RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet01
    RouteTableId: !Ref PublicRouteTable
```

PublicRouteTable:

A container for routing rules associated with public subnets and specific VPC.

```
PublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: Public Subnets
    - Key: Network
      Value: Public
```

PublicRoute

A specific route inside that table is directing internet-bound traffic to the Internet Gateway.

```
PublicRoute:
  DependsOn: VPCGatewayAttachment
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway
```

Private Subnets

PrivateSubnet01 (192.168.0.0/18 as defined in PrivateSubnet01Block) is tagged with the name of the stack and subnet index. Subnet is intended for internal load balancers.

```
PrivateSubnet01Block:
  Type: String
  Default: 192.168.128.0/18
  Description: CidrBlock for private subnet 01 within the VPC
```

```
PrivateSubnet01:
  Type: AWS::EC2::Subnet
  Metadata:
    Comment: Subnet 03
  Properties:
    AvailabilityZone:
      Fn::Select:
      - '0'
      - Fn::GetAZs:
          Ref: AWS::Region
    CidrBlock:
      Ref: PrivateSubnet01Block
    VpcId:
      Ref: VPC
    Tags:
    - Key: Name
      Value: !Sub "${AWS::StackName}-PrivateSubnet01"
    - Key: kubernetes.io/role/internal-elb
      Value: 1
```

Private Subnet and RouteTable associations

The PrivateSubnet01RouteTableAssociation links PrivateSubnet01 to the PrivateRouteTable. It ensures that all network traffic from the subnet follows the routing rules defined in the associated route table.

```
PrivateSubnet01RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PrivateSubnet01
    RouteTableId: !Ref PrivateRouteTable01
```

PrivateRouteTable:

PrivateRouteTable01 creates a private route table within the VPC for managing traffic for private subnets. Indicates the subnet's AZ and its private nature.

```
PrivateRouteTable01:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: Private Subnet AZ1
    - Key: Network
      Value: Private01
```

PrivateRoute01

he PrivateRoute01 creates a route in PrivateRouteTable01 to send all internet-bound traffic (0.0.0.0/0) from PrivateSubnet01 through NatGateway01, enabling secure outbound internet access while depending on the VPC gateway and NAT gateway being attached.

```
 PrivateRoute01:
    DependsOn:
    - VPCGatewayAttachment
    - NatGateway01
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable01
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway01
```

NatGateway01

The NatGateway01 creates a NAT Gateway in PublicSubnet01, using an Elastic IP (NatGatewayEIP1) to provide internet access for resources in private subnets, with dependencies ensuring the public subnet and VPC gateway are ready before creation.

```
NatGateway01:
    DependsOn:
    - NatGatewayEIP1
    - PublicSubnet01
    - VPCGatewayAttachment
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt 'NatGatewayEIP1.AllocationId'
      SubnetId: !Ref PublicSubnet01
      Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-NatGatewayAZ1'
```

NatGatewayEIP1

The NatGatewayEIP1 resource creates an Elastic IP (EIP) for the NAT Gateway in the VPC, ensuring it has a static public IP address for outbound internet traffic, with a dependency on the VPC gateway attachment to ensure network readiness.

```
NatGatewayEIP1:
  DependsOn:
  - VPCGatewayAttachment
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: vpc
```

SecurityGroup

The ControlPlaneSecurityGroup defines an AWS security group that allows secure communication between the EKS control plane and worker nodes in the private subnets. It permits all outbound traffic and incoming traffic from specific private subnets, enabling cluster operations and management while maintaining network isolation.

```
ControlPlaneSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Cluster communication with worker nodes
    VpcId: !Ref VPC
    SecurityGroupEgress:
      - CidrIp: 0.0.0.0/0
        IpProtocol: "-1"
    SecurityGroupIngress:
      - CidrIp: !Ref PrivateSubnet01Block
        IpProtocol: "-1"  # Allow all protocols from PrivateSubnet01
        FromPort: "0"
        ToPort: "65535"
      - CidrIp: !Ref PrivateSubnet02Block
        IpProtocol: "-1"  # Allow all protocols from PrivateSubnet02
        FromPort: "0"
        ToPort: "65535"
      - CidrIp: 192.168.0.0/16
        IpProtocol: "-1"
        FromPort: "0"
        ToPort: "65535"
```

Deploy VPC:

Change Region and name accordingly to your needs.

```
aws cloudformation create-stack \
 --region us-east-1 \
 --stack-name my-eks-vpc \
 --template-body file://eks-vpc-stack.yaml
```

4) Create EKS

Download cloudformation EKS configuration yaml. It includes EKS configuration, IAM roles and nodegroup definition.

```
curl -O
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/EKS/CloudFormation/eks
-stack.yaml
```

Parameters

The parameters define the EKS cluster name, worker node count, instance type, version

```
Parameters:
 ClusterName:
   Type: String
   Default: my-eks-cluster
 NumberOfWorkerNodes:
   Type: Number
   Default: 1
 WorkerNodesInstanceType:
   Type: String
   Default: t2.micro
 KubernetesVersion:
   Type: String
   Default: 1.22
```

IAM roles

The EksRole resource creates an IAM role for the EKS cluster, with permissions to allow the EKS service to assume the role (sts:AssumeRole). It attaches the AmazonEKSClusterPolicy managed policy to the role, granting necessary permissions for EKS cluster operations.

```
EksRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: my.eks.cluster.role
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - eks.amazonaws.com
          Action:
            - sts:AssumeRole
    Path: /
    ManagedPolicyArns:
      - "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
```

The EksNodeRole resource creates an IAM role for EC2 worker nodes in EKS, granting permissions for EKS worker node operations, container registry access, and CNI plugin functionality.

```
EksNodeRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: my.eks.node.role
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
[...]
    Path: /
    ManagedPolicyArns:
      - "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
      - "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
      - "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
```

EKS Config

The EksCluster resource creates an Amazon EKS cluster, using the specified name, Kubernetes version, and IAM role, and configures it with VPC resources by associating it with security groups and subnets for networking.

```
EksCluster:
    Type: AWS::EKS::Cluster
    Properties:
      Name: !Ref ClusterName
      Version: !Ref KubernetesVersion
      RoleArn: !GetAtt EksRole.Arn
      ResourcesVpcConfig:
        SecurityGroupIds:
          - !ImportValue ControlPlaneSecurityGroupId
        SubnetIds: !Split [ ',', !ImportValue PrivateSubnetIds ]
```

NodeGroups Deploy

The EksNodegroup resource creates an EKS node group, which defines the worker nodes for the EKS cluster, specifying the node role, scaling configuration (min, desired, and max size), and the subnets in which the nodes will run.

```
EksNodegroup:
  Type: AWS::EKS::Nodegroup
  DependsOn: EksCluster
  Properties:
    ClusterName: !Ref ClusterName
    NodeRole: !GetAtt EksNodeRole.Arn
    ScalingConfig:
      MinSize:
        Ref: NumberOfWorkerNodes
      DesiredSize:
        Ref: NumberOfWorkerNodes
      MaxSize:
        Ref: NumberOfWorkerNodes
    Subnets: !Split [ ',', !ImportValue PrivateSubnetIds ]
```

5) Deploy EKS

Change Region and name accordingly to your needs. Run deployment from folder, where yaml is located.

```
aws cloudformation create-stack \
  --region us-east-1 \
  --stack-name my-eks-cluster \
  --capabilities CAPABILITY_NAMED_IAM \
  --template-body file://eks-stack.yaml
```

6) Delete Resources

To avoid unnecessary bills, destroy resources after you ve done all you wanted:

Delete VPC

```
aws cloudformation delete-stack \
  --region us-east-1 \
  --stack-name my-eks-vpc
```

Delete EKS

```
aws cloudformation delete-stack \
  --region us-east-1 \
  --stack-name my-eks-cluster
```

Kubernetes AWS: Deploy EKS with Terraform

1) What is Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool by HashiCorp that allows you to define and automate the provisioning of cloud infrastructure using a declarative configuration language. It enables consistent, repeatable deployments across multiple providers like AWS

2) Terraform blocks

modules

Terraform modules are reusable packages of Terraform code that help organize and manage infrastructure. They make it easy to share and use consistent setups across projects.

```
module "eks" {
  source          = "terraform-aws-modules/eks/aws"
  version         = "20.8.4"
  cluster_name    = local.cluster_name
[...]
  eks_managed_node_groups = {
    node_group = {
      min_size     = 2
      max_size     = 6
      desired_size = 2
    }
  }
}
```

This Terraform module configures an Amazon EKS cluster by using the terraform-aws-modules/eks/aws module. It specifies details like the cluster name, Kubernetes version etc.

resource

It is a single component of infrastructure, like an EC2 instance or S3 bucket, defined in your configuration

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

The resource block defines a single security group in AWS. It sets the name_prefix for the group and associates it with a specific VPC using the vpc_id from the module.vpc.

provider

A provider in Terraform is a plugin that allows Terraform to interact with and manage resources on a specific platform, like AWS or Azure. It defines the connection details and the resources Terraform can create or modify.

```
provider "aws" {
  region = var.aws_region
}
```

The provider "aws" block configures Terraform to use the AWS provider, specifying settings like the region (var.aws_region) where resources will be managed.

Variable

In Terraform, a variable is an input value you define to make your configurations flexible and reusable.

```
variable "kubernetes_version" {
  default     = 1.27
  description = "kubernetes version"
}
```

The variable block defines kubernetes_version with a default value of 1.27 and a description, allowing users to customize the Kubernetes version in their Terraform configuration.

Output

An output block in Terraform is used to display specific values after a Terraform run, such as resource attributes or configuration details. It helps users retrieve key information, like resource IDs or endpoints, from their infrastructure.

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

The output "cluster_id" block in Terraform displays the EKS cluster ID, sourced from the eks module, after the infrastructure is provisioned.

Terraform

The terraform block configures settings like required Terraform version, providers, and backend for managing state and execution.

```
terraform {
  required_version = ">= 1.0"
}
```

3) Create VPC

VPC tf file defines a local variable for a unique EKS cluster name, generated by appending a random 8-character string suffix. A VPC is created using the Terraform AWS VPC module, configured with specified CIDR blocks, public and private subnets, and support for DNS and a single NAT gateway. VPC CIDR is defined through paramatere stored in tf variables file.

```
provider "aws" {
  region = var.aws_region
}

data "aws_availability_zones" "available" {}

locals {
  cluster_name = "michael-eks-${random_string.suffix.result}"
}

resource "random_string" "suffix" {
  length  = 8
  special = false
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.7.0"

  name                 = "michael-eks-vpc"
  cidr                 = var.vpc_cidr
  azs                  = data.aws_availability_zones.available.names
  private_subnets      = ["10.0.1.0/24", "10.0.2.0/24"]
  public_subnets       = ["10.0.4.0/24", "10.0.5.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true
  enable_dns_support   = true
}
```

3) Create EKS

This Terraform configuration deploys an Amazon EKS cluster using the terraform-aws-modules/eks module. It specifies the cluster name, version, private subnets, and enables IRSA for secure service account roles. Managed node groups use t3.medium instances with autoscaling between 2 and 6 nodes. The setup ensures scalable and secure Kubernetes workloads within a VPC. Security rules are defined in separate tf file

```
module "eks" {
 source          = "terraform-aws-modules/eks/aws"
 version         = "20.8.4"
 cluster_name    = local.cluster_name
 cluster_version = var.kubernetes_version
 subnet_ids      = module.vpc.private_subnets

 enable_irsa = true

 tags = {
  cluster = "demo"
 }

 vpc_id = module.vpc.vpc_id

 eks_managed_node_group_defaults = {
  ami_type               = "AL2_x86_64"
  instance_types         = ["t3.medium"]
  vpc_security_group_ids = [aws_security_group.all_worker_mgmt.id]
 }

 eks_managed_node_groups = {

  node_group = {
   min_size     = 2
   max_size     = 6
   desired_size = 2
  }
 }
}
```

Security Group

Security tf file defines an ingress rule allowing inbound traffic from private CIDR ranges commonly used in VPCs. An egress rule is also set up to permit outbound traffic to any destination. Together, these rules enable secure communication for the worker nodes within the VPC and with external resources as needed.

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

resource "aws_security_group_rule" "all_worker_mgmt_ingress" {
  description       = "allow inbound traffic from eks"
  from_port         = 0
  protocol          = "-1"
  to_port           = 0
  security_group_id = aws_security_group.all_worker_mgmt.id
  type              = "ingress"
  cidr_blocks = [
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16",
  ]
}

resource "aws_security_group_rule" "all_worker_mgmt_egress" {
  description       = "allow outbound traffic to anywhere"
  from_port         = 0
  protocol          = "-1"
  security_group_id = aws_security_group.all_worker_mgmt.id
  to_port           = 0
  type              = "egress"
  cidr_blocks       = ["0.0.0.0/0"]
}
```

4) S3 state backend

S3 as a Terraform state backend stores the Terraform state file, enabling collaboration and versioning. It integrates with DynamoDB for state locking to prevent concurrent modifications. S3 provides scalability, security (via encryption and IAM), and high availability for managing infrastructure state.

```
resource "aws_s3_bucket" "s3_bucket" {
  bucket = "michael-s3-demo-xyz" # change this
}

resource "aws_dynamodb_table" "terraform_lock" {
  name           = "terraform-lock"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

5) Terraform Initialize, plan, apply, destroy

Initializes the Terraform environment.

```
terraform init
```

Previews changes to infrastructure.

```
terraform plan
```

Apply the changes to your infrastructure.

```
terraform apply
```

After s3 bucket is created, uncomment backend.tf file

```
terraform {
  backend "s3" {
    bucket         = "michael-s3-demo-xyz" # change this
    key            = "abhi/terraform.tfstate"
    region         = "us-east-1"
    encrypt        = true
    dynamodb_table = "terraform-lock"
  }
}
```

Destory infrastructure, to save money:

```
terraform destroy
```

6) Other Blocks: Variables, Outputs

Modify those for future deployments when project versions change, new variables are introduced, or to view different outputs after EKS deployment, in order to gather relevant information about the cluster's state.

Kubernetes AWS: Deploy EKS with eksctl

1) What is eksctl

eksctl is a command-line tool for managing Amazon EKS clusters. It simplifies cluster creation, node management, and configuration on AWS. eksctl streamlines Kubernetes deployment and management with minimal effort.

2) Setup variables

1. AWS CLI
2. kubectl - the Kubernetes CLI
3. eksctl (>= v0.191.0) - the CLI for AWS EKS

Install those on your system and configure connection to aws CLI.

```
export KARPENTER_NAMESPACE="kube-system"
export KARPENTER_VERSION="1.0.8"
export K8S_VERSION="1.31"
export AWS_PARTITION="aws"
export CLUSTER_NAME="${USER}-karpenter-demo"
export AWS_DEFAULT_REGION="us-west-2"
export AWS_ACCOUNT_ID="$(aws sts get-caller-identity --query Account --output text)"
export TEMPOUT="$(mktemp)"
```

Those variables will be used in EKS creation through eksctl, change them accordingly to your needs.

3) Deploy VPC and EKS

eksctl can only create EKS cluster. To create VPC Karpenter cloudformation will be used:

```
curl -fsSL https://raw.githubusercontent.com/aws/karpenter-provider-
aws/v"${KARPENTER_VERSION}"/website/content/en/preview/getting-started/getting-
started-with-karpenter/cloudformation.yaml  > "${TEMPOUT}" \
&& aws cloudformation deploy \
  --stack-name "Karpenter-${CLUSTER_NAME}" \
  --template-file "${TEMPOUT}" \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides "ClusterName=${CLUSTER_NAME}"
```

Now download EKS yaml file:

```
curl -O
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/EKS/eksctl/eks.ya
ml
```

And deploy it:

```
eksctl create cluster -f eks.yaml
```

4) Destroy resources

```
aws cloudformation delete-stack --stack-name "Karpenter-${CLUSTER_NAME}"
eksctl delete cluster --name "${CLUSTER_NAME}"
```

common troubleshooting

1) Node Group Scaling Fails

Cause: Subnets lack available IP addresses or Auto Scaling Group misconfigurations
Solution: Check available IPs in your subnets using the AWS console and ensure ScalingConfig parameters are correctly set in the node group definition.

2) Worker Nodes Not Joining the Cluster

Cause: IAM role misconfiguration or missing node group permissions.
Solution: Verify that the EksNodeRole includes policies for AmazonEKSWorkerNodePolicy, AmazonEKS_CNI_Policy, and AmazonEC2ContainerRegistryReadOnly. Use the AWS Management Console or:

3) Pod Connectivity Issues

Cause: Misconfigured security groups or routing issues.
Solution: Verify that your VPC, subnets, and security groups are correctly configured. Ensure ControlPlaneSecurityGroup allows traffic between the control plane and worker nodes.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!