* **Cluster Autoscaller & Karpenter & AWS Fargate**
* **Create AWS VPC Using Terraform**
* **Create EKS Cluster Using Terraform**
* **Create Karpenter Controller IAM Role**
* **Deploy Karpenter to EKS**
* **Create Karpenter Provisioner**
* **Demo: Automatic Node Provisioning**

Create AWS VPC Using Terraform

First of all, we need to create VPC using terraform. In this video, I'm not going to go over each configuration parameter of each terraform resource as I did in the previous videos.

Provider with some variables such as EKS **cluster** name and a **region**.

|  |  |
| --- | --- |
| **0-provider.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | variable "cluster\_name" {  default = "demo"  }    provider "aws" {  region = "us-east-1"  }    terraform {  required\_version = "~> 1.0"    required\_providers {  aws = {  source = "hashicorp/aws"  version = "~> 4.0"  }  helm = {  source = "hashicorp/helm"  version = "~> 2.6"  }  }  } |

VPC resource with **EFS** specific parameters.

|  |  |
| --- | --- |
| **1-vpc.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11 | resource "aws\_vpc" "main" {  cidr\_block = "10.0.0.0/16"    # Must be enabled for EFS  enable\_dns\_support = true  enable\_dns\_hostnames = true    tags = {  Name = "main"  }  } |

Internet Gateway.

|  |  |
| --- | --- |
| **2-igw.tf** | |
| 1  2  3  4  5  6  7 | resource "aws\_internet\_gateway" "igw" {  vpc\_id = aws\_vpc.main.id    tags = {  Name = "igw"  }  } |

Four subnets, two **private** and two **public**.

|  |  |
| --- | --- |
| **3-subnets.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | resource "aws\_subnet" "private-us-east-1a" {  vpc\_id = aws\_vpc.main.id  cidr\_block = "10.0.0.0/19"  availability\_zone = "us-east-1a"    tags = {  "Name" = "private-us-east-1a"  "kubernetes.io/role/internal-elb" = "1"  "kubernetes.io/cluster/${var.cluster\_name}" = "owned"  }  }    resource "aws\_subnet" "private-us-east-1b" {  vpc\_id = aws\_vpc.main.id  cidr\_block = "10.0.32.0/19"  availability\_zone = "us-east-1b"    tags = {  "Name" = "private-us-east-1b"  "kubernetes.io/role/internal-elb" = "1"  "kubernetes.io/cluster/${var.cluster\_name}" = "owned"  }  }    resource "aws\_subnet" "public-us-east-1a" {  vpc\_id = aws\_vpc.main.id  cidr\_block = "10.0.64.0/19"  availability\_zone = "us-east-1a"  map\_public\_ip\_on\_launch = true    tags = {  "Name" = "public-us-east-1a"  "kubernetes.io/role/elb" = "1"  "kubernetes.io/cluster/${var.cluster\_name}" = "owned"  }  }    resource "aws\_subnet" "public-us-east-1b" {  vpc\_id = aws\_vpc.main.id  cidr\_block = "10.0.96.0/19"  availability\_zone = "us-east-1b"  map\_public\_ip\_on\_launch = true    tags = {  "Name" = "public-us-east-1b"  "kubernetes.io/role/elb" = "1"  "kubernetes.io/cluster/${var.cluster\_name}" = "owned"  }  } |

NAT Gateway.

|  |  |
| --- | --- |
| **4-nat.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | resource "aws\_eip" "nat" {  vpc = true    tags = {  Name = "nat"  }  }    resource "aws\_nat\_gateway" "nat" {  allocation\_id = aws\_eip.nat.id  subnet\_id = aws\_subnet.public-us-east-1a.id    tags = {  Name = "nat"  }    depends\_on = [aws\_internet\_gateway.igw]  } |

Finally two routes: one public with **default route to internet gateway** and a private with **default route to NAT Gateway**.

|  |  |
| --- | --- |
| **5-routes.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | resource "aws\_route\_table" "private" {  vpc\_id = aws\_vpc.main.id    route {  cidr\_block = "0.0.0.0/0"  nat\_gateway\_id = aws\_nat\_gateway.nat.id  }    tags = {  Name = "private"  }  }    resource "aws\_route\_table" "public" {  vpc\_id = aws\_vpc.main.id    route {  cidr\_block = "0.0.0.0/0"  gateway\_id = aws\_internet\_gateway.igw.id  }    tags = {  Name = "public"  }  }    resource "aws\_route\_table\_association" "private-us-east-1a" {  subnet\_id = aws\_subnet.private-us-east-1a.id  route\_table\_id = aws\_route\_table.private.id  }    resource "aws\_route\_table\_association" "private-us-east-1b" {  subnet\_id = aws\_subnet.private-us-east-1b.id  route\_table\_id = aws\_route\_table.private.id  }    resource "aws\_route\_table\_association" "public-us-east-1a" {  subnet\_id = aws\_subnet.public-us-east-1a.id  route\_table\_id = aws\_route\_table.public.id  }    resource "aws\_route\_table\_association" "public-us-east-1b" {  subnet\_id = aws\_subnet.public-us-east-1b.id  route\_table\_id = aws\_route\_table.public.id  } |

Let's initialize terraform and create all those components with terraform apply.

terraform init

terraform apply

Create EKS Cluster Using Terraform

Next, we need to create an **EKS cluster** and a **node group**. EKS requires an IAM role to access AWS API on your behave to create resources.

|  |  |
| --- | --- |
| **6-eks.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | resource "aws\_iam\_role" "eks-cluster" {  name = "eks-cluster"    assume\_role\_policy = <<POLICY  {  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Principal": {  "Service": "eks.amazonaws.com"  },  "Action": "sts:AssumeRole"  }  ]  }  POLICY  }    resource "aws\_iam\_role\_policy\_attachment" "amazon-eks-cluster-policy" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"  role = aws\_iam\_role.eks-cluster.name  }    resource "aws\_eks\_cluster" "cluster" {  name = var.cluster\_name  version = "1.22"  role\_arn = aws\_iam\_role.eks-cluster.arn    vpc\_config {    endpoint\_private\_access = false  endpoint\_public\_access = true  public\_access\_cidrs = ["0.0.0.0/0"]    subnet\_ids = [  aws\_subnet.private-us-east-1a.id,  aws\_subnet.private-us-east-1b.id,  aws\_subnet.public-us-east-1a.id,  aws\_subnet.public-us-east-1b.id  ]  }    depends\_on = [aws\_iam\_role\_policy\_attachment.amazon-eks-cluster-policy]  } |

Now we need to create another **IAM role** for **Kubernetes nodes**. It's going to be used by the regular node pool and not Karpenter.

You have **two options**, either to use the **same IAM role** and create an instance profile for Karpenter or you can create a **dedicated IAM role**. But in this case, you would need to **manually update auth configmap** to authorize nodes created by karpenter with a new IAM role to join the cluster.

|  |  |
| --- | --- |
| **7-nodes.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | resource "aws\_iam\_role" "nodes" {  name = "eks-node-group"    assume\_role\_policy = jsonencode({  Statement = [{  Action = "sts:AssumeRole"  Effect = "Allow"  Principal = {  Service = "ec2.amazonaws.com"  }  }]  Version = "2012-10-17"  })  }    resource "aws\_iam\_role\_policy\_attachment" "amazon-eks-worker-node-policy" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"  role = aws\_iam\_role.nodes.name  }    resource "aws\_iam\_role\_policy\_attachment" "amazon-eks-cni-policy" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy"  role = aws\_iam\_role.nodes.name  }    resource "aws\_iam\_role\_policy\_attachment" "amazon-ec2-container-registry-read-only" {  policy\_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"  role = aws\_iam\_role.nodes.name  }    resource "aws\_eks\_node\_group" "private-nodes" {  cluster\_name = aws\_eks\_cluster.cluster.name  version = "1.22"  node\_group\_name = "private-nodes"  node\_role\_arn = aws\_iam\_role.nodes.arn    subnet\_ids = [  aws\_subnet.private-us-east-1a.id,  aws\_subnet.private-us-east-1b.id  ]    capacity\_type = "ON\_DEMAND"  instance\_types = ["t3.small"]    scaling\_config {  desired\_size = 1  max\_size = 10  min\_size = 0  }    update\_config {  max\_unavailable = 1  }    labels = {  role = "general"  }    depends\_on = [  aws\_iam\_role\_policy\_attachment.amazon-eks-worker-node-policy,  aws\_iam\_role\_policy\_attachment.amazon-eks-cni-policy,  aws\_iam\_role\_policy\_attachment.amazon-ec2-container-registry-read-only,  ]    # Allow external changes without Terraform plan difference  lifecycle {  ignore\_changes = [scaling\_config[0].desired\_size]  }  } |

Now let's again apply the terraform to create an EKS cluster.

terraform apply

To connect to the cluster you need to **update the Kubernetes context** with this command.

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

Then the **quick check** if we can reach Kubernetes. It should return the default k8s service.

kubectl get svc

As I mentioned before, if you decide to create a **separate IAM role** and instance profile **you would need to edit the auth configmap** to add the ARN of the new role.

kubectl edit configmap aws-auth -n kube-system

Create Karpenter Controller IAM Role

Karpenter needs **permissions** to create EC2 instances in AWS. If you use a self-hosted Kubernetes cluster, for example by using kOps. You can add additional IAM policies to the existing IAM role **attached to Kubernetes nodes**. We use EKS, the best way to grant access to internal service would be with **IAM roles for service accounts**.

First, we need to create an **OpenID Connect provider**.

|  |  |
| --- | --- |
| **8-iam-oidc.tf** | |
| 1  2  3  4  5  6  7  8  9 | data "tls\_certificate" "eks" {  url = aws\_eks\_cluster.cluster.identity[0].oidc[0].issuer  }    resource "aws\_iam\_openid\_connect\_provider" "eks" {  client\_id\_list = ["sts.amazonaws.com"]  thumbprint\_list = [data.tls\_certificate.eks.certificates[0].sha1\_fingerprint]  url = aws\_eks\_cluster.cluster.identity[0].oidc[0].issuer  } |

Next is a trust policy to allow the Kubernetes service account to assume the IAM role. Make sure that you deploy Karpenter to the **karpenter** **namespace** with the same **service account name**.

|  |  |
| --- | --- |
| **9-karpenter-controller-role.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | data "aws\_iam\_policy\_document" "karpenter\_controller\_assume\_role\_policy" {  statement {  actions = ["sts:AssumeRoleWithWebIdentity"]  effect = "Allow"    condition {  test = "StringEquals"  variable = "${replace(aws\_iam\_openid\_connect\_provider.eks.url, "https://", "")}:sub"  values = ["system:serviceaccount:karpenter:karpenter"]  }    principals {  identifiers = [aws\_iam\_openid\_connect\_provider.eks.arn]  type = "Federated"  }  }  }    resource "aws\_iam\_role" "karpenter\_controller" {  assume\_role\_policy = data.aws\_iam\_policy\_document.karpenter\_controller\_assume\_role\_policy.json  name = "karpenter-controller"  }    resource "aws\_iam\_policy" "karpenter\_controller" {  policy = file("./controller-trust-policy.json")  name = "KarpenterController"  }    resource "aws\_iam\_role\_policy\_attachment" "aws\_load\_balancer\_controller\_attach" {  role = aws\_iam\_role.karpenter\_controller.name  policy\_arn = aws\_iam\_policy.karpenter\_controller.arn  }    resource "aws\_iam\_instance\_profile" "karpenter" {  name = "KarpenterNodeInstanceProfile"  role = aws\_iam\_role.nodes.name  } |

Let's create the **controller-trust-policy.json** file.

|  |  |
| --- | --- |
| **controller-trust-policy.json** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | {  "Statement": [  {  "Action": [  "ssm:GetParameter",  "iam:PassRole",  "ec2:RunInstances",  "ec2:DescribeSubnets",  "ec2:DescribeSecurityGroups",  "ec2:DescribeLaunchTemplates",  "ec2:DescribeInstances",  "ec2:DescribeInstanceTypes",  "ec2:DescribeInstanceTypeOfferings",  "ec2:DescribeAvailabilityZones",  "ec2:DeleteLaunchTemplate",  "ec2:CreateTags",  "ec2:CreateLaunchTemplate",  "ec2:CreateFleet"  ],  "Effect": "Allow",  "Resource": "\*",  "Sid": "Karpenter"  },  {  "Action": "ec2:TerminateInstances",  "Condition": {  "StringLike": {  "ec2:ResourceTag/Name": "\*karpenter\*"  }  },  "Effect": "Allow",  "Resource": "\*",  "Sid": "ConditionalEC2Termination"  }  ],  "Version": "2012-10-17"  } |

Since we've added an additional provider we need to initialize before we can apply the terraform code.

terraform init

terraform apply

Deploy Karpenter to EKS

To deploy Karpenter to our cluster, we're going to use **Helm**. First of all, you need to **authenticate** with EKS using the helm provider. Then the helm release.

|  |  |
| --- | --- |
| **10-karpenter-helm.tf** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | provider "helm" {  kubernetes {  host = aws\_eks\_cluster.cluster.endpoint  cluster\_ca\_certificate = base64decode(aws\_eks\_cluster.cluster.certificate\_authority[0].data)    exec {  api\_version = "client.authentication.k8s.io/v1beta1"  args = ["eks", "get-token", "--cluster-name", aws\_eks\_cluster.cluster.id]  command = "aws"  }  }  }    resource "helm\_release" "karpenter" {  namespace = "karpenter"  create\_namespace = true    name = "karpenter"  repository = "https://charts.karpenter.sh"  chart = "karpenter"  version = "v0.13.1"    set {  name = "serviceAccount.annotations.eks\\.amazonaws\\.com/role-arn"  value = aws\_iam\_role.karpenter\_controller.arn  }    set {  name = "clusterName"  value = aws\_eks\_cluster.cluster.id  }    set {  name = "clusterEndpoint"  value = aws\_eks\_cluster.cluster.endpoint  }    set {  name = "aws.defaultInstanceProfile"  value = aws\_iam\_instance\_profile.karpenter.name  }    depends\_on = [aws\_eks\_node\_group.private-nodes]  } |

Let's apply and check if the controller is running.

terraform apply

Check if the helm was deployed successfully. Then the karpenter pod in its dedicated namespace.

helm list -A

kubectl get pods -n karpenter

Create Karpenter Provisioner

Before we can test Karpenter, we need to create a **Provisioner**. Karpenter defines a Custom Resource called a Provisioner to specify provisioning configuration. Each provisioner manages a **distinct set of nodes**. You need to replace the **demo** with your **EKS cluster name**.

|  |  |
| --- | --- |
| **provisioner.yaml** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | ---  apiVersion: karpenter.sh/v1alpha5  kind: Provisioner  metadata:  name: default  spec:  ttlSecondsAfterEmpty: 60 # scale down nodes after 60 seconds without workloads (excluding daemons)  ttlSecondsUntilExpired: 604800 # expire nodes after 7 days (in seconds) = 7 \* 60 \* 60 \* 24  limits:  resources:  cpu: 100 # limit to 100 CPU cores  requirements:  # Include general purpose instance families  - key: karpenter.k8s.aws/instance-family  operator: In  values: [c5, m5, r5]  # Exclude small instance sizes  - key: karpenter.k8s.aws/instance-size  operator: NotIn  values: [nano, micro, small, large]  providerRef:  name: my-provider  ---  apiVersion: karpenter.k8s.aws/v1alpha1  kind: AWSNodeTemplate  metadata:  name: my-provider  spec:  subnetSelector:  kubernetes.io/cluster/demo: owned  securityGroupSelector:  kubernetes.io/cluster/demo: owned |

Finally, use kubectl to create those resources in the cluster.

kubectl apply -f k8s/provisioner.yaml

Demo: Automatic Node Provisioning

Lastly, let's create a Kubernetes deployment to test how quickly Karpenter can **create EC2 instances** and **schedule new pods**.

|  |  |
| --- | --- |
| **deployment.yaml** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-deployment  labels:  app: nginx  spec:  replicas: 5  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:1.14.2  resources:  requests:  cpu: "4"  memory: 4Gi  ports:  - containerPort: 80 |

When you just getting started with Karpenter, it's a good idea to check logs in case you get any **errors**.

kubectl logs -f -n karpenter \

-l app.kubernetes.io/name=karpenter

In another window, let's run get pods.

watch -n 1 -t kubectl get pods

Then let's get all the nodes available in the Kubernetes cluster.

watch -n 1 -t kubectl get nodes

Finally, create the deployment with 5 replicas.

kubectl apply -f k8s/deployment.yaml