**EKS Usecase**

**🡪EKS Cluster Autoscaling using Karpenter**

**🡪 Rolling Update**

**🡪Multi-AZ Deployment**

**🡪ETCD Backup**

**1. Cluster Autoscaling using Karpenter:**

**Create a set up with necessary IAM roles and policies for running Karpenter on an EKS cluster use Helm to deploy Karpenter Controller to the EKS cluster. The key steps are:**

1. Setting up the EKS cluster with the required IAM role and policies.

2. Creating IAM policies and roles for Karpenter.

3. Deploying Karpenter using Helm with appropriate configurations.

<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/tree/main/terraform>

**1. Setting up the EKS cluster with the required IAM role and policies.**

1.1 Create EKS Cluster using terraform and add the necessary IAM Role and Permission for EKS Cluster

resource "aws\_eks\_cluster" "demo" {

  name     = var.cluster\_name

  role\_arn = aws\_iam\_role.demo.arn

  version = "1.29"

  vpc\_config {

    subnet\_ids = [

      aws\_subnet.private-ap-south-1a.id,

      aws\_subnet.private-ap-south-1b.id,

      aws\_subnet.public-ap-south-1a.id,

      aws\_subnet.public-ap-south-1b.id

    ]

  }

  depends\_on = [aws\_iam\_role\_policy\_attachment.demo-AmazonEKSClusterPolicy]

}

**2. Creating IAM policies and roles for Karpenter.**

<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/terraform/13-karpenter.tf>

2.1 Creates an IAM policy document that allows the Karpenter controller to assume a role using a web identity token and policy grants the sts:AssumeRoleWithWebIdentity action and ensures that the identity comes from a specific service account in the karpenter namespace which is the OIDC provider associated with the EKS cluster

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"

    }

  }

}

2.2 Create IAM Policy to access the EC2 instance to Auto scale the nodes

{

    "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"

}

**3. Deploying Karpenter using Helm with appropriate configurations.**

3.1 Use Helm to deploy Karpenter to the karpenter namespace and Specify Helm chart repository and version, ARN of the IAM role, cluster name, cluster endpoint, instance profile

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.demo.id

  }

  set {

    name  = "clusterEndpoint"

    value = aws\_eks\_cluster.demo.endpoint

  }

  set {

    name  = "aws.defaultInstanceProfile"

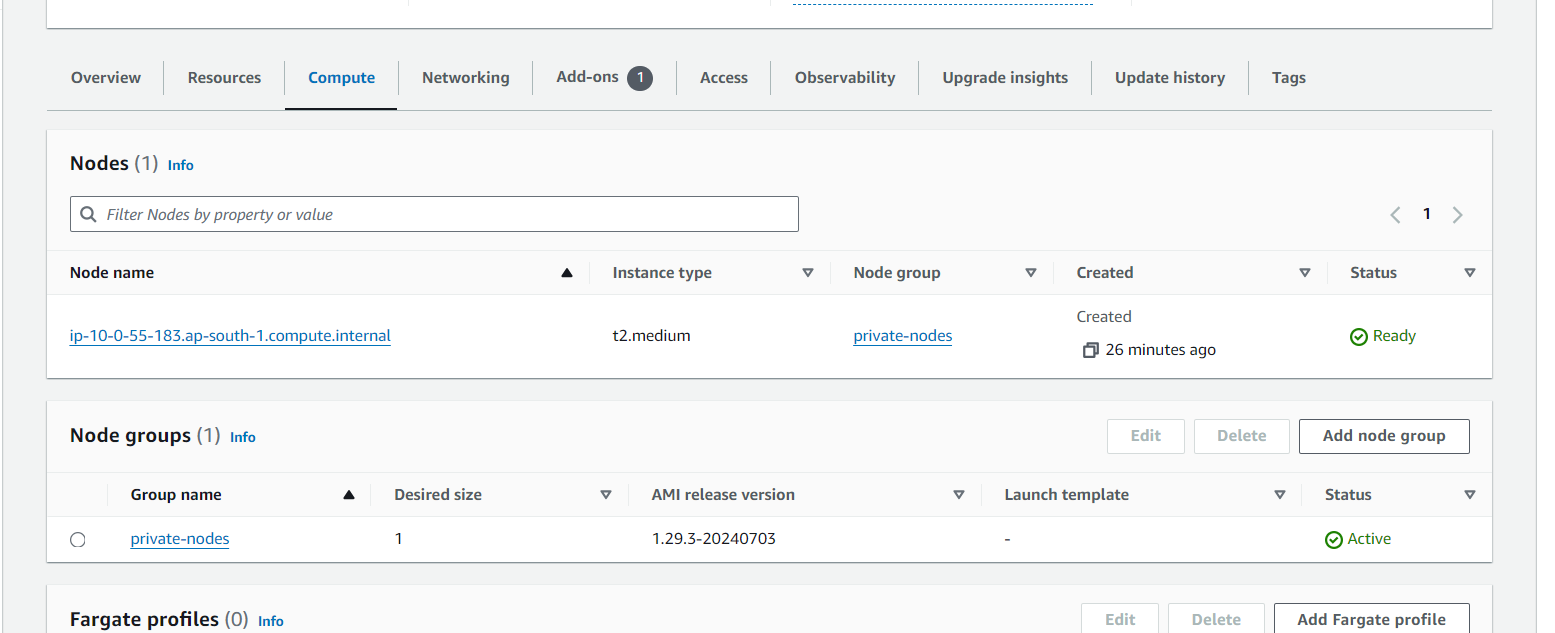
    value = aws\_iam\_instance\_profile.karpenter.name

  }

  depends\_on = [aws\_eks\_node\_group.private-nodes]

}

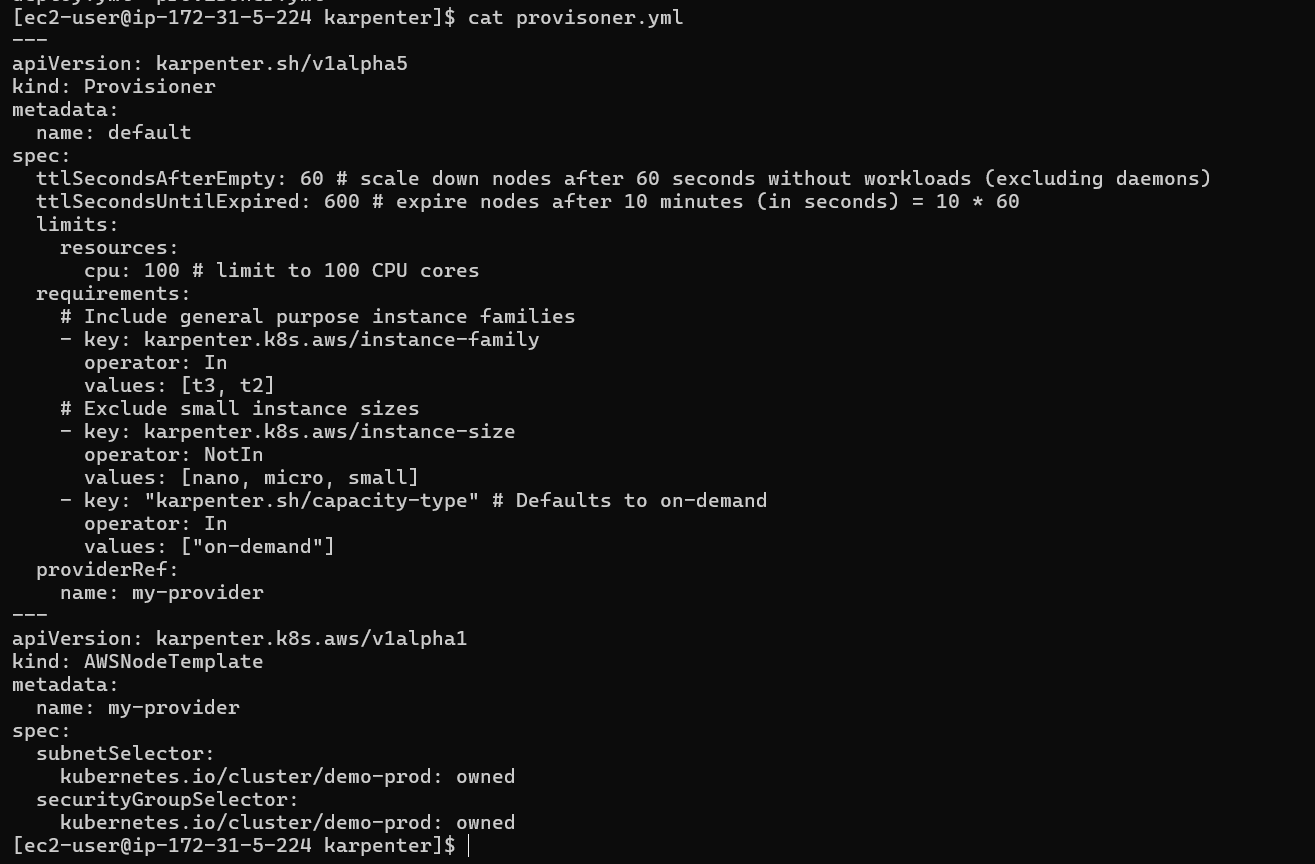
3.2 Initially EKS node is set to one



3.3 Create karpenter provisioner for autoscaling the nodes

**kubectl apply -f provisoner.yml**

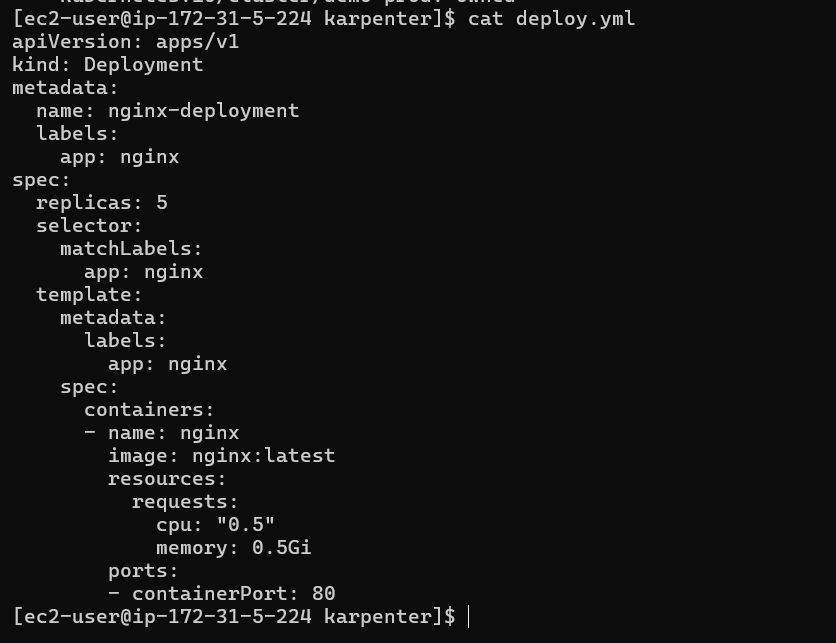
<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/files/provisoner.yml>



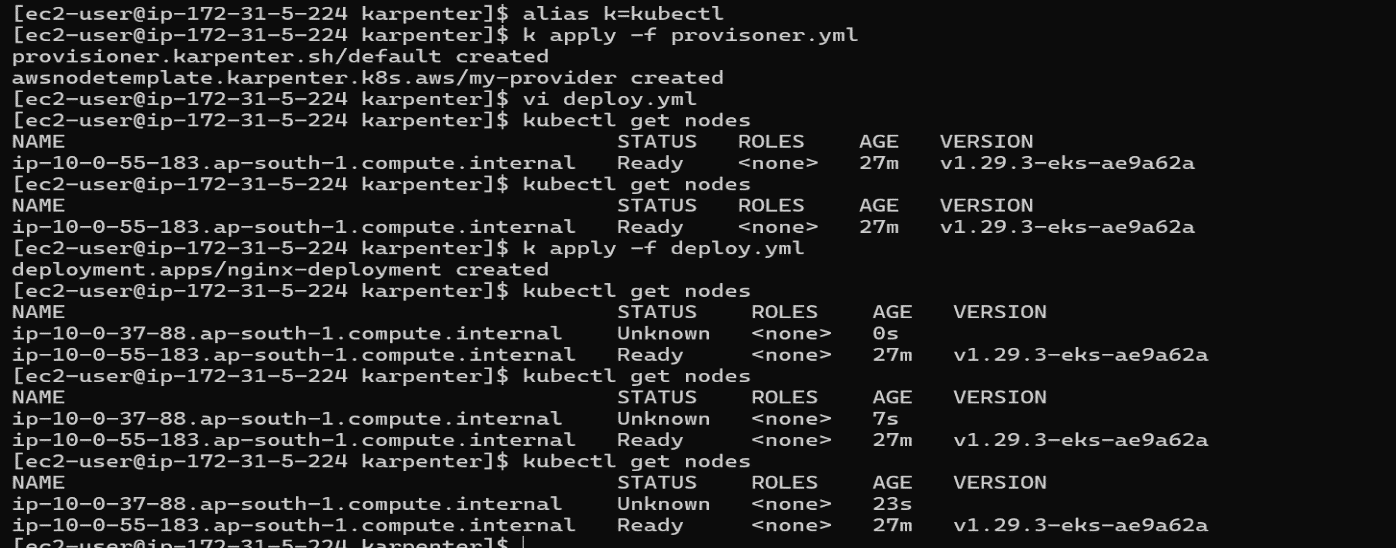
To Increase the Memory and CPU, we are deploying nginx with high CPU and memory unit

**kubectl apply -f deploy.yml**

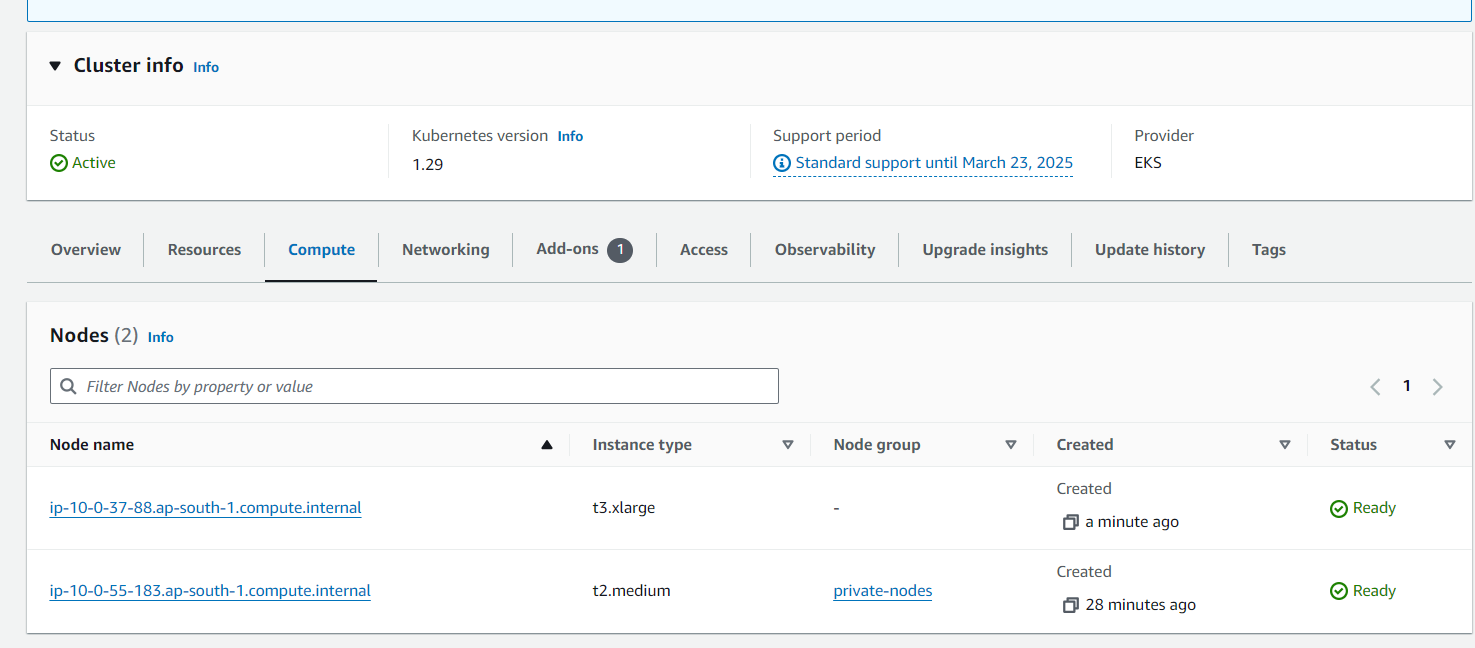
<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/files/deployer.yml>



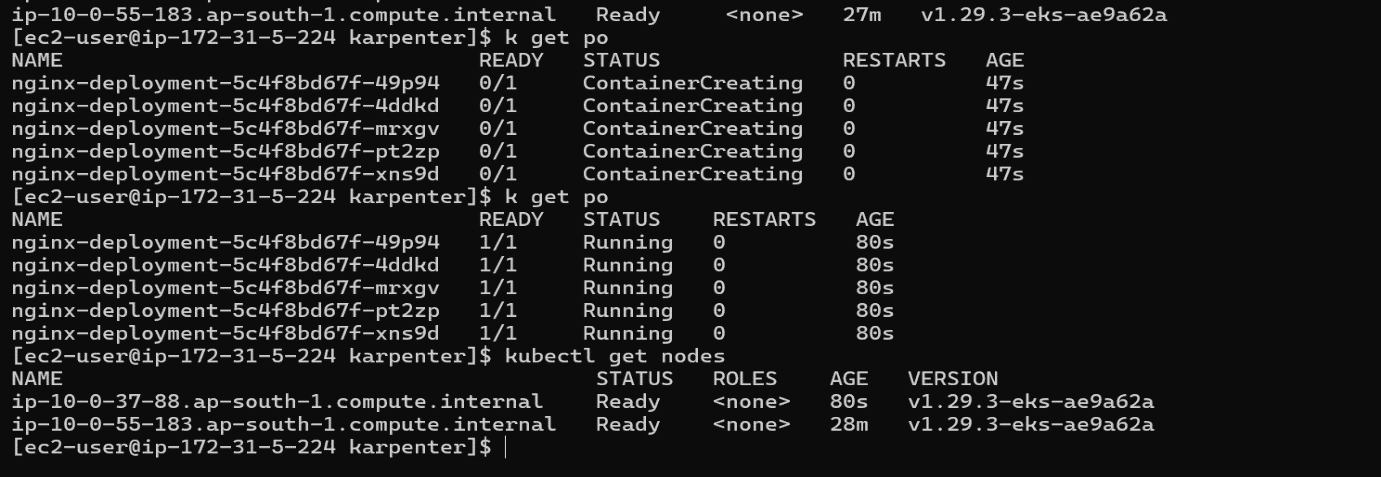
You can see before one node was present, after applying the deploy.yml manifest file, nodes gets increased



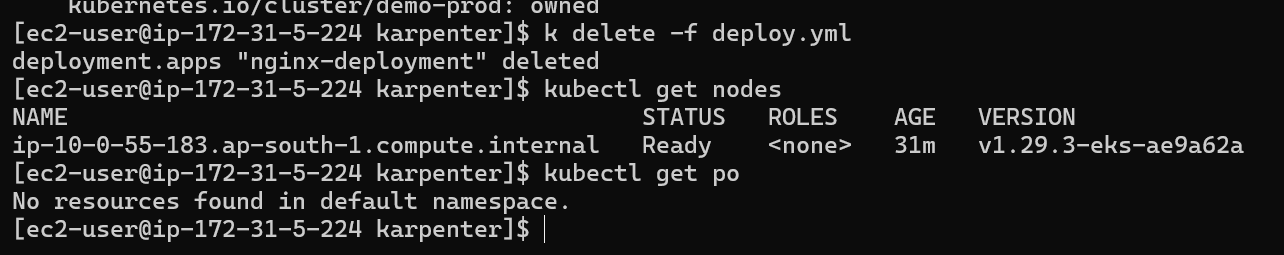
3.4 New node is scaling up with instance type t3.xlarge



3.5 In deploy.yml manifest file, given replicas as 5



3.6 when deleting the deployment file nodes get scaled down to one



**2. Rolling Update:**

**Rolling Update**

Kubernetes will handle the rolling update automatically. It will start creating new pods with the updated image version and gradually terminate the old pods, ensuring zero downtime

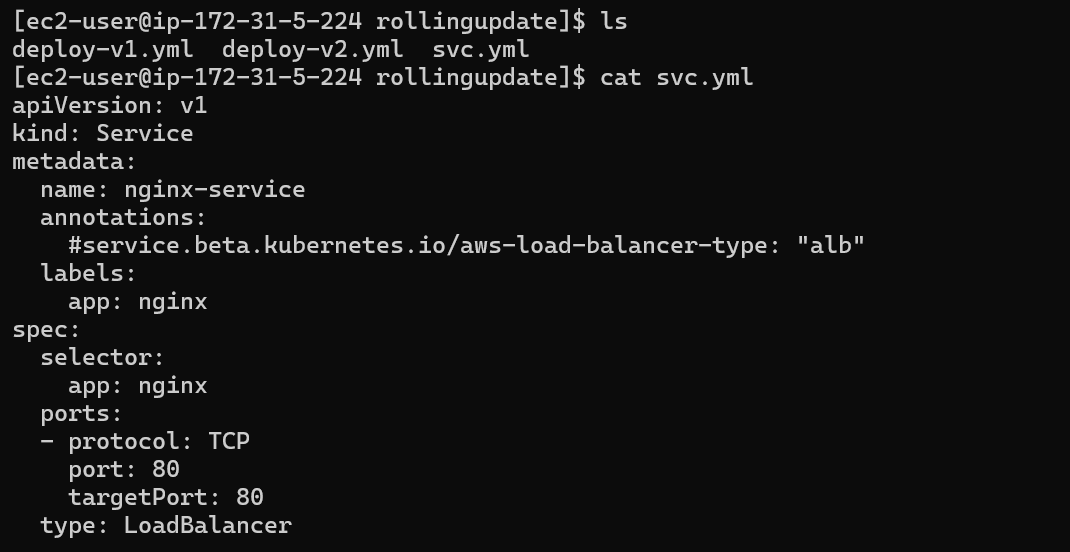
**Rollback**

If you need to roll back to the previous version (e.g., due to issues with the new version), you can use the following commands

1.Create service for nginx deployment

**kubectl apply -f svc.yml**

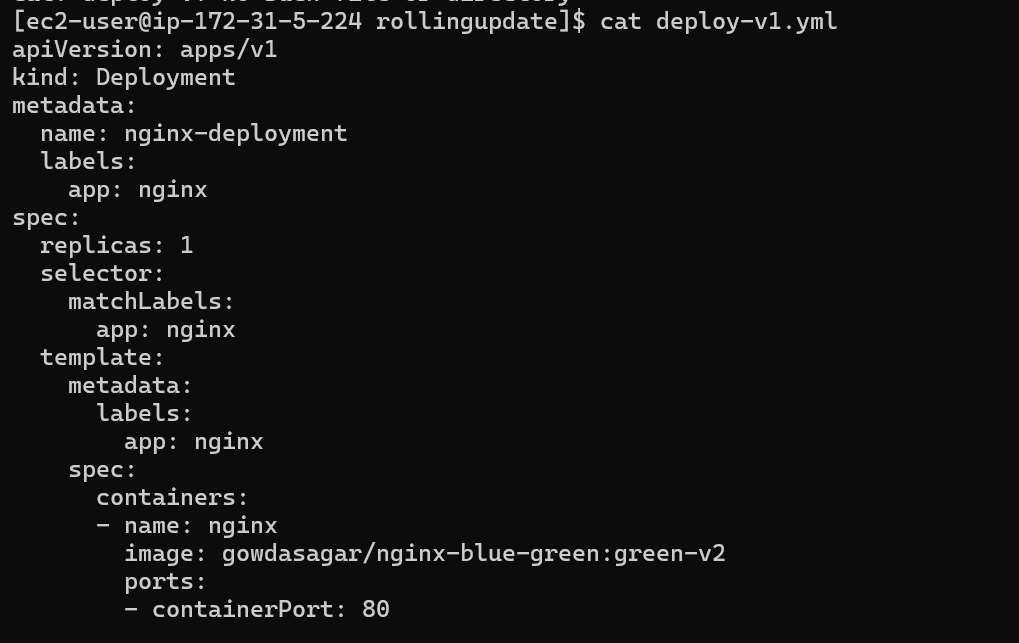
<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/files/rollingupdate-svc.yml>



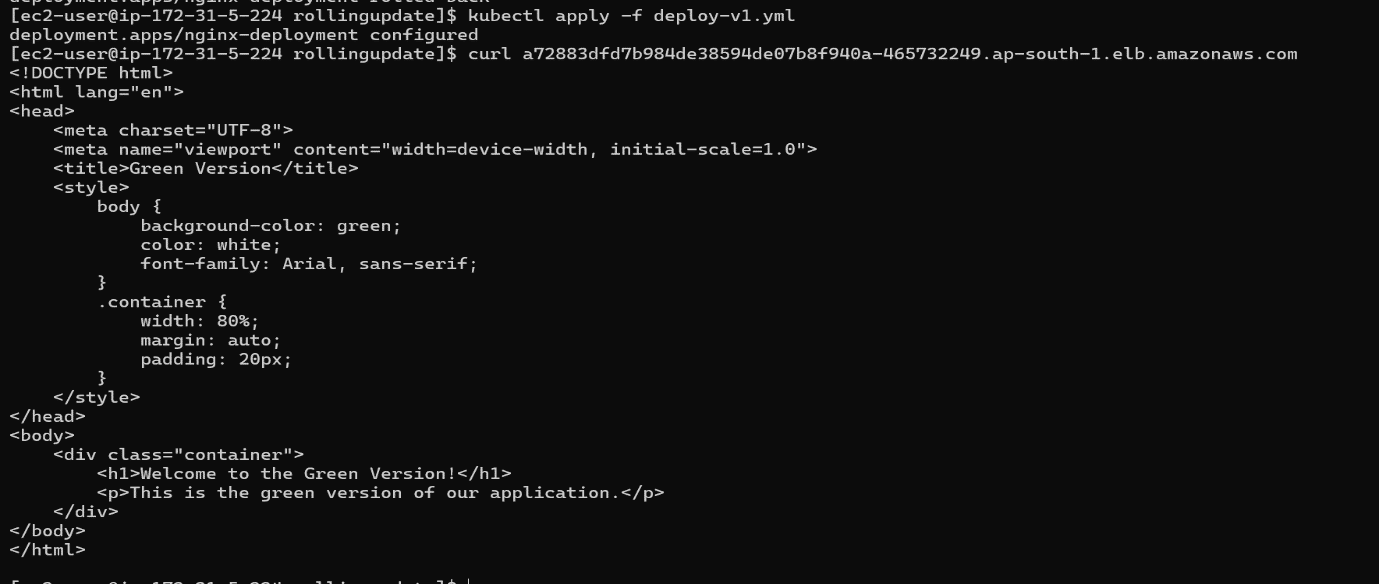
2. Create deployment with one image for nginx deployment

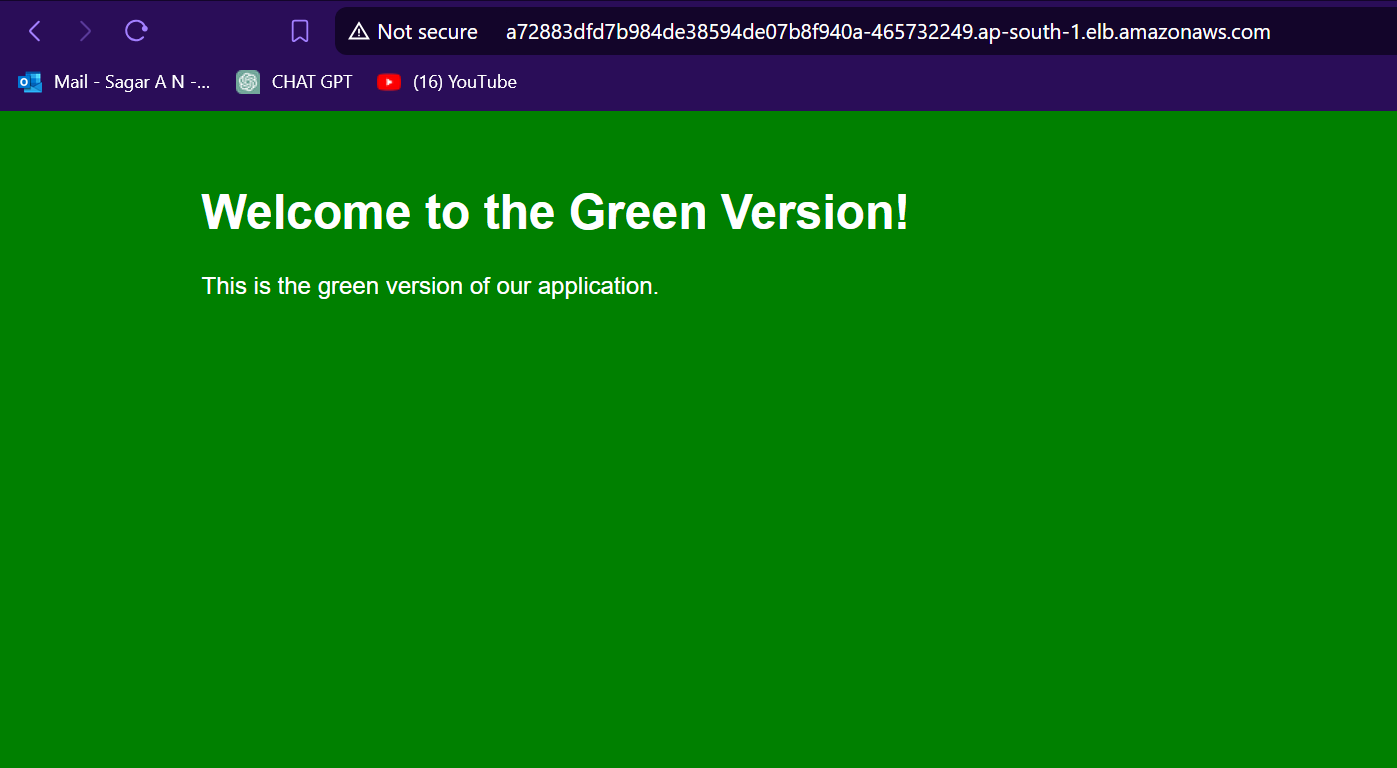
**kubectl apply -f deploy-v1.yml**

<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/files/rollingupdate-deploy-v1.yml>





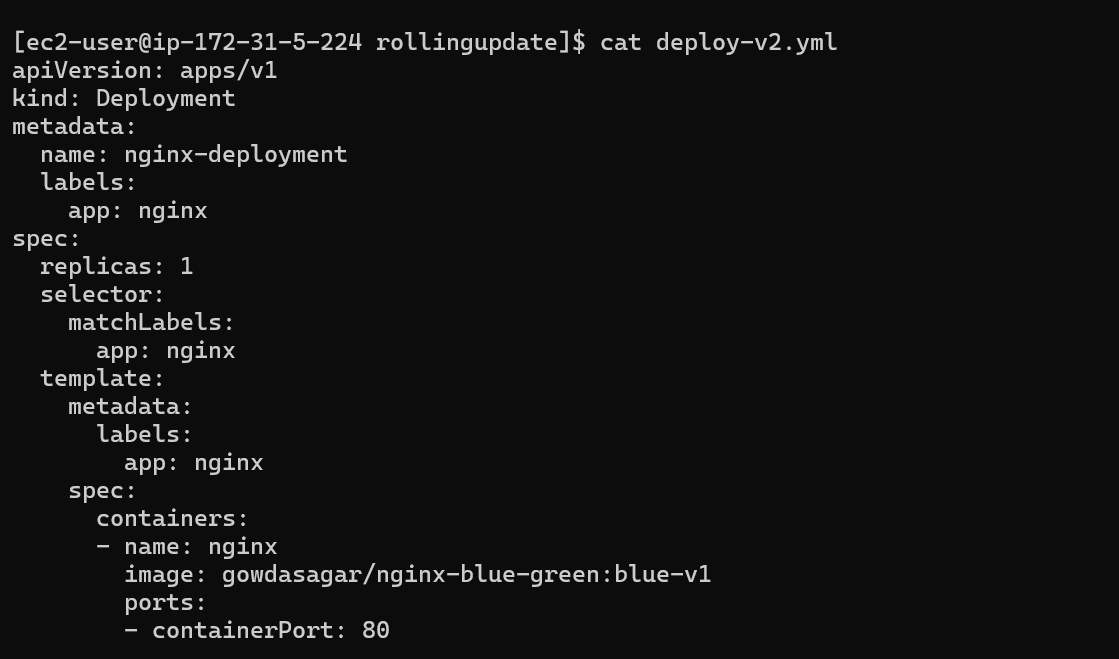


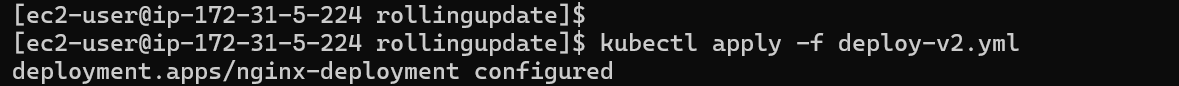


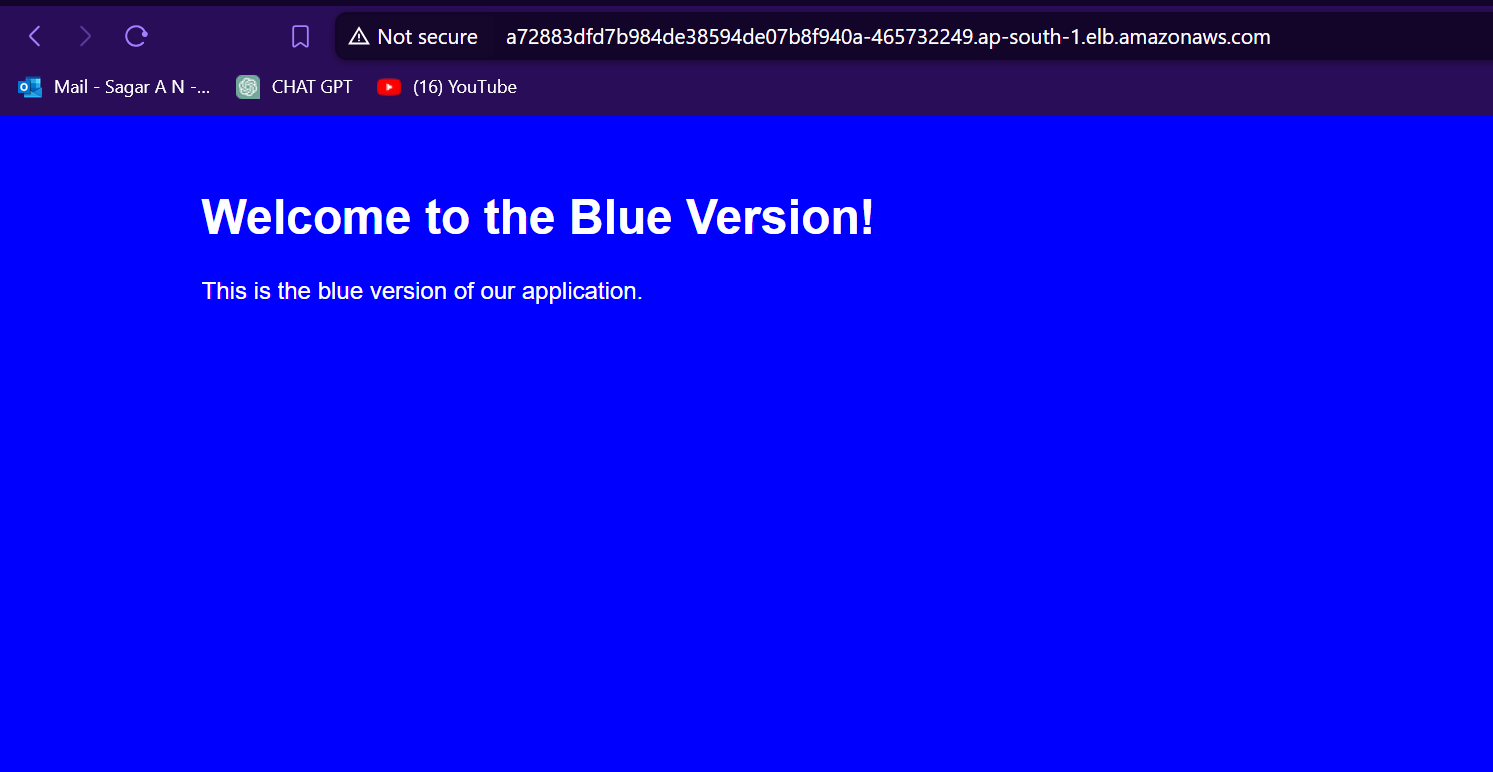
3. Create Redeployment with updated image for nginx deployment

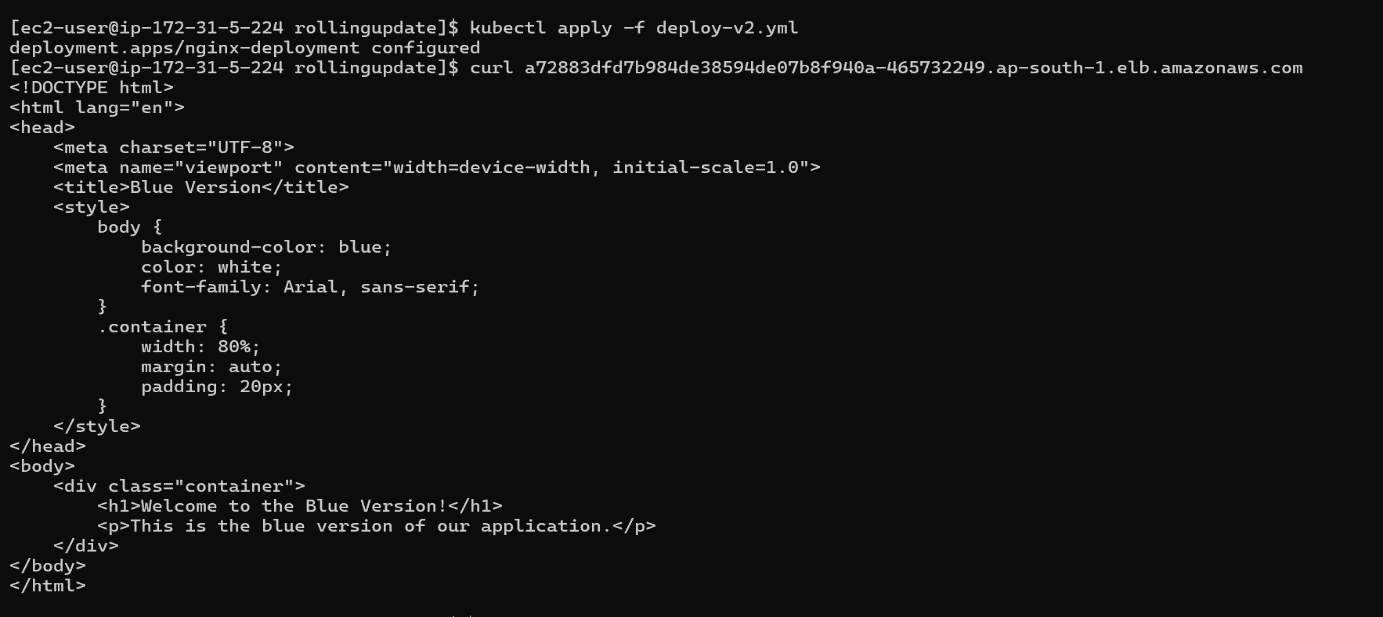
**kubectl apply -f deploy-v2.yml**

<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/files/rollingupdate-deploy-v2.yml>





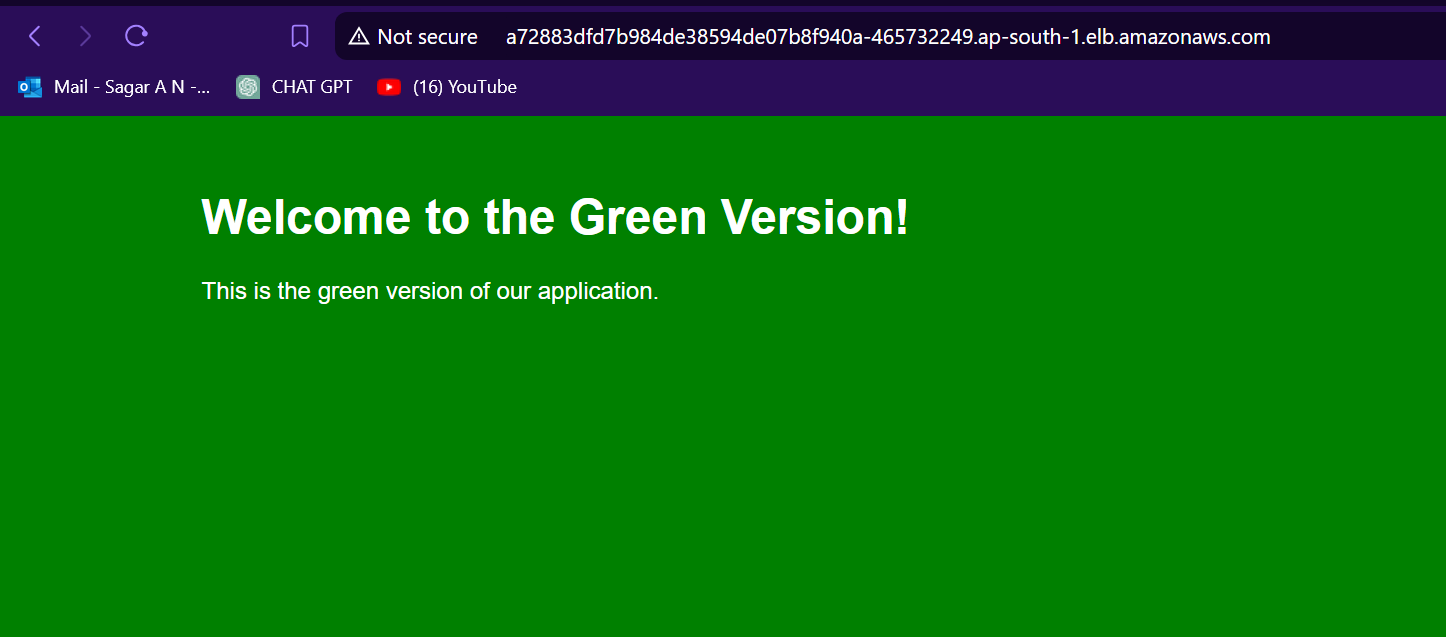


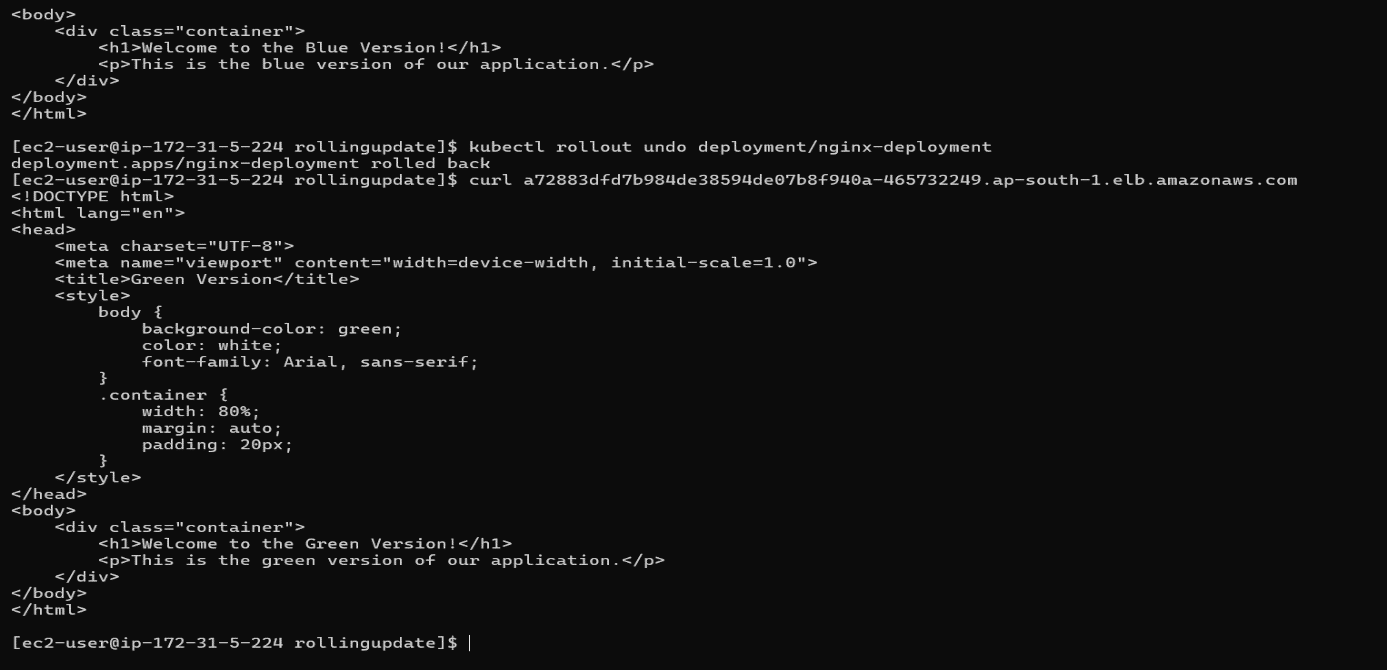


4. Using Rollout undo command you can rollout to previous deployment if any issue arise in current deployment

**kubectl rollout undo deployment/nginx-deployment**





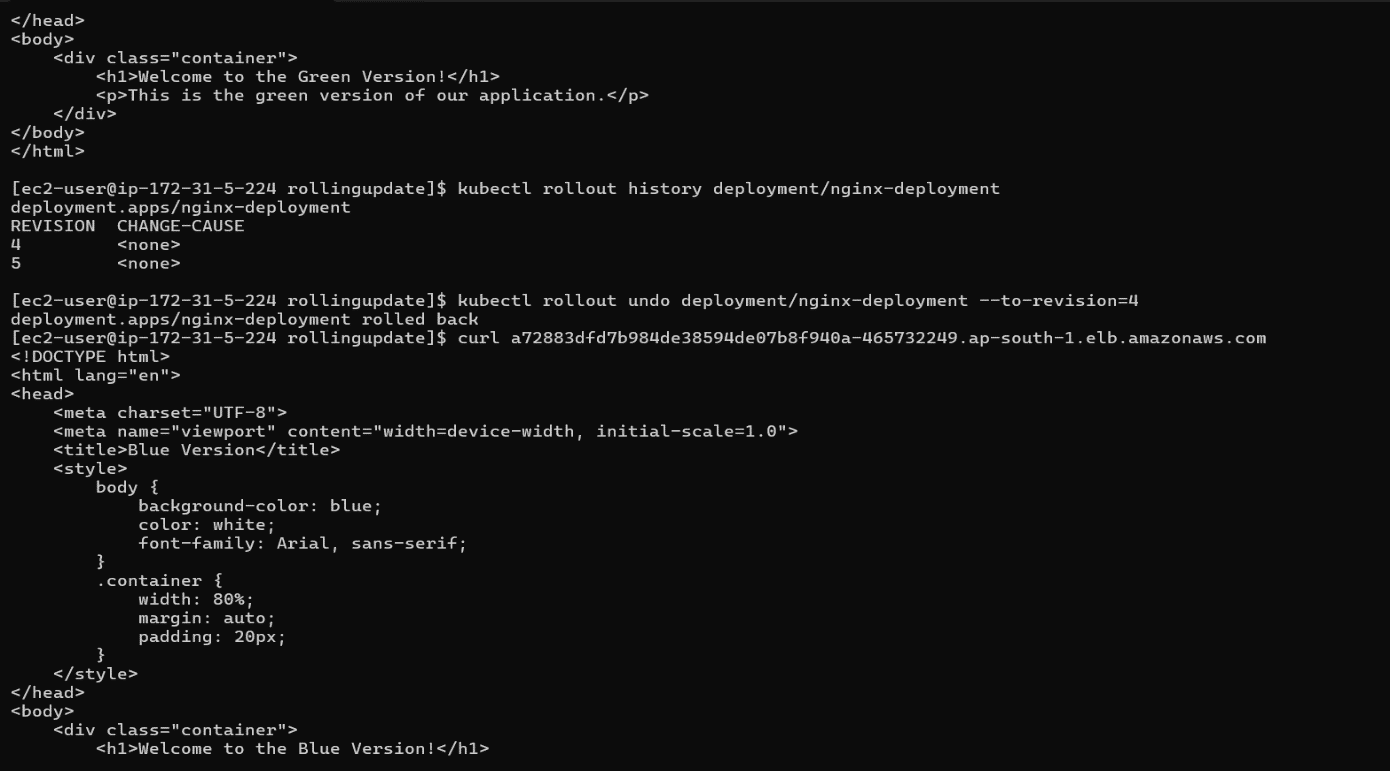


5. You can view history of deployment with revision number

**kubectl rollout history deployment/nginx-deployment**

6.Using Rollout undo --to-revision=<revison-number> tag you can rollout to particular version

**kubectl rollout undo deployment/nginx-deployment --to-revision=<revison-number>**



**3. Multi-AZ Deployment:**

Deploying an Amazon EKS cluster across multiple Availability Zones (AZs) enhances the availability and resilience of applications. It ensures that Kubernetes workloads are distributed across different AZs, reducing the risk of downtime in case of an AZ failure and persistent storage across multiple AZs, you can achieve high availability, fault tolerance, and improved performance

While creating node group, for choosing subnet, select the subnets which are created in the different AZs for High Availability

<https://github.com/gowdasagar06/cluster-autoscaling-karpenter-rolling-update/blob/main/terraform/7-nodes.tf>

resource "aws\_eks\_node\_group" "private-nodes" {

  cluster\_name    = aws\_eks\_cluster.demo.name

  node\_group\_name = "private-nodes"

  node\_role\_arn   = aws\_iam\_role.nodes.arn

  subnet\_ids = [

    aws\_subnet.private-ap-south-1a.id,

    aws\_subnet.private-ap-south-1b.id

  ]

  capacity\_type  = "ON\_DEMAND"

  instance\_types = var.node\_types

  scaling\_config {

    desired\_size = var.node\_desired\_size

    max\_size     = var.node\_maximum\_size

    min\_size     = var.node\_minimum\_size

  }

  update\_config {

    max\_unavailable = 1

  }

  labels = {

    role = "general"

  }

  depends\_on = [

    aws\_iam\_role\_policy\_attachment.nodes-AmazonEKSWorkerNodePolicy,

    aws\_iam\_role\_policy\_attachment.nodes-AmazonEKS\_CNI\_Policy,

    aws\_iam\_role\_policy\_attachment.nodes-AmazonEC2ContainerRegistryReadOnly,

  ]

}

**4. ETCD Backup:**

Amazon EKS automatically manages the ETCD database for control plane components, including automated backups. These backups help ensure that your cluster can be restored in the event of an issue with the control plane. Here are key points regarding ETCD backups in EKS

**1. Managed Control Plane:**

EKS runs and manages the Kubernetes control plane, including the ETCD database, across multiple Availability Zones (AZs) to ensure high availability and durability.

Amazon EKS automatically performs ETCD backups for disaster recovery.

**2. Automated Backups:**

EKS automatically takes backups of the ETCD database. This ensures that the cluster's state can be restored to a known good state in case of an issue.

These backups are part of the managed control plane and do not require manual intervention.

**3. Disaster Recovery:**

In the event of an ETCD failure, AWS will restore the control plane from the most recent backups to minimize data loss and downtime.

This managed recovery process ensures that the cluster can return to a functioning state quickly.

**4. Limitations:**

While EKS manages the ETCD backups, users do not have direct access to these backups or the ability to trigger manual backups.

Custom backup and restore strategies for specific workloads running on worker nodes are still the user's responsibility.