ADVANCE DEVOPS CASE STUDY

ANSH SARFARE **D15A/50**

1. Introduction

Case Study Overview: This case study focuses on the deployment and management of multi-cloud infrastructure using Terraform, Kubernetes, and AWS S3. The goal was to create an S3 bucket on AWS to host a static website and a Kubernetes cluster to deploy a Node.js application. This showcases the use of Terraform for automating cloud infrastructure management.

Key Feature and Application: The standout feature of this project is the automation of cloud infrastructure using Terraform. It demonstrates how to seamlessly create and manage an S3 bucket and a Kubernetes cluster, highlighting the integration of multiple cloud services for a single application deployment. In addition, Terraform's capability to abstract and automate complex cloud infrastructure management makes it highly practical for teams working with multi-cloud environments.

Setup of Terraform and AWS S3

Step-1: Set Up AWS Credentials in provider.tf:

Step-2: Create S3 Bucket using Terraform (main.tf):

statichosting > main.tf > ...

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "5.64.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.6.2"
    }
  }
}

resource "random_id" "rand_id" {
  byte_length = 8
}

resource "aws_s3_bucket" "staticweb-bucket" {
  bucket = "demo-bucket-${random_id.rand_id.hex}"
}

resource "aws_s3_object" "index_html" {
  bucket = aws_s3_bucket.staticweb-bucket.bucket
  source = "./index.html"
  key    = "index.html"
  content_type = "text/html"
}
resource "aws_s3_object" "style_css" {
  bucket = aws_s3_bucket.staticweb-bucket.bucket
  source = "./styles.css"
  key    = "styles.css"
  content_type = "text/css"
}

resource "aws_s3_bucket_public_access_block" "example" {
  bucket = aws_s3_bucket.staticweb-bucket.bucket

  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}
```

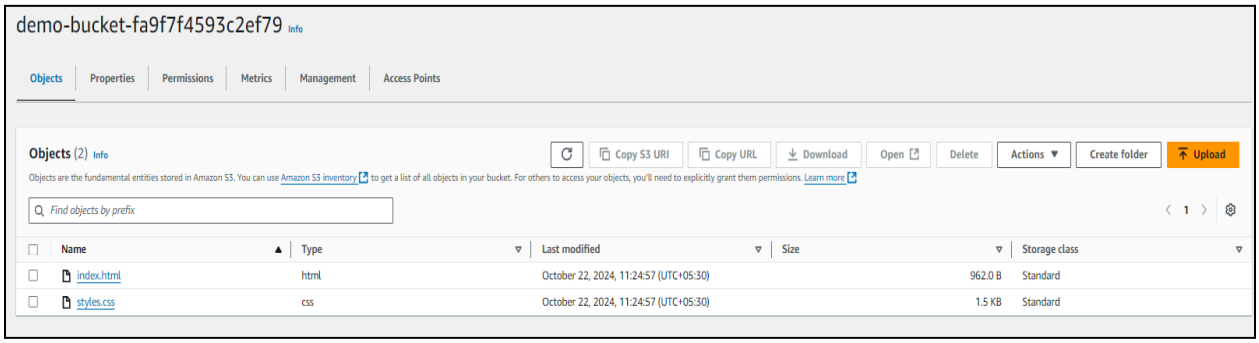
```hcl
resource "aws_s3_bucket_policy" "bucket-policy" {
  bucket = aws_s3_bucket.staticweb-bucket.bucket
  policy = jsonencode(
    {
      Version = "2012-10-17",
      Statement = [
        {
          Sid = "PublicReadGetObject",
          Effect = "Allow",
          Principal = "*",
          Action = [
            "s3:GetObject"
          ],
          Resource = [
            "arn:aws:s3:::${aws_s3_bucket.staticweb-bucket.id}/*"
          ]
        }
      ]
    }
  )

}
resource "aws_s3_bucket_website_configuration" "example" {
  bucket = aws_s3_bucket.staticweb-bucket.id


  index_document {
    suffix = "index.html"
  }
}

output "website_endpoint" {
  value = aws_s3_bucket_website_configuration.example.website_endpoint
}
```

Step-3:Run Terraform commands

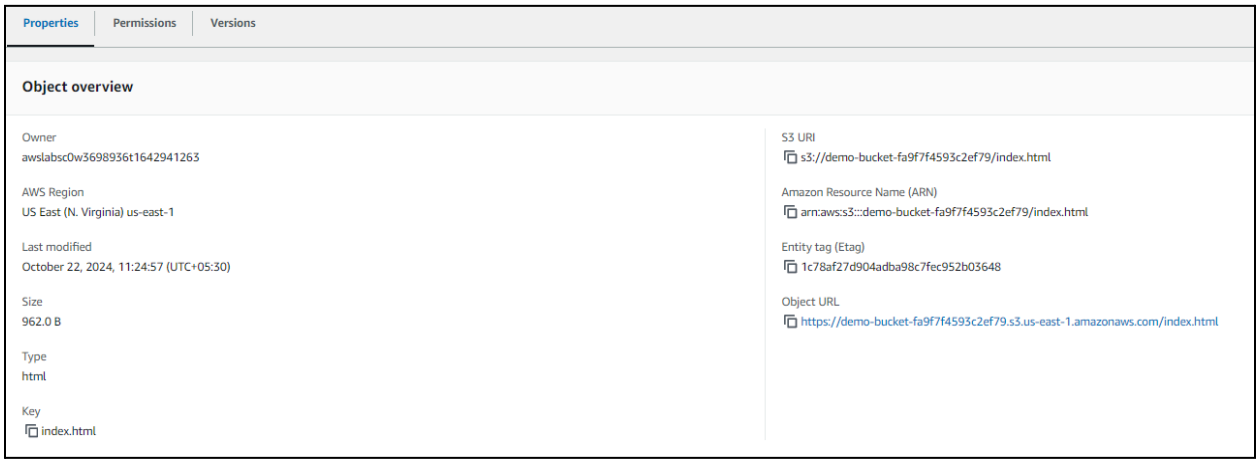
- Terraform init
- Terraform plan
- Terraform apply

The bucket:

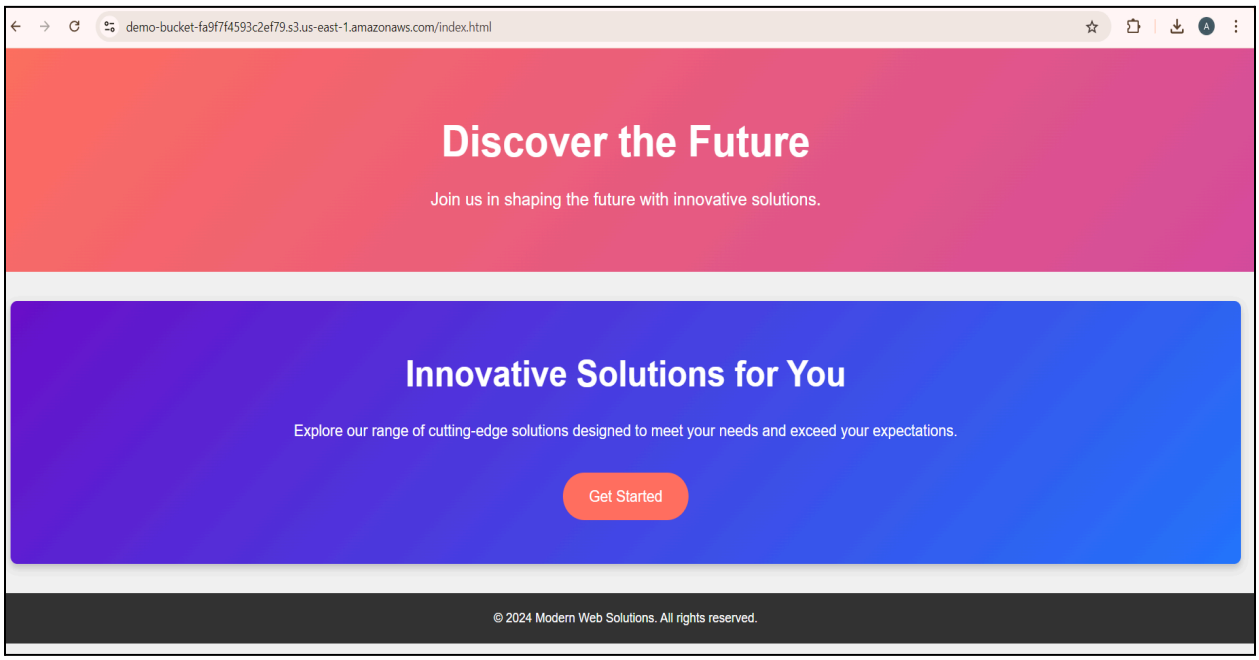
Objects in bucket:



The Object Url in Object properties to access the page:



Hosted Static page:

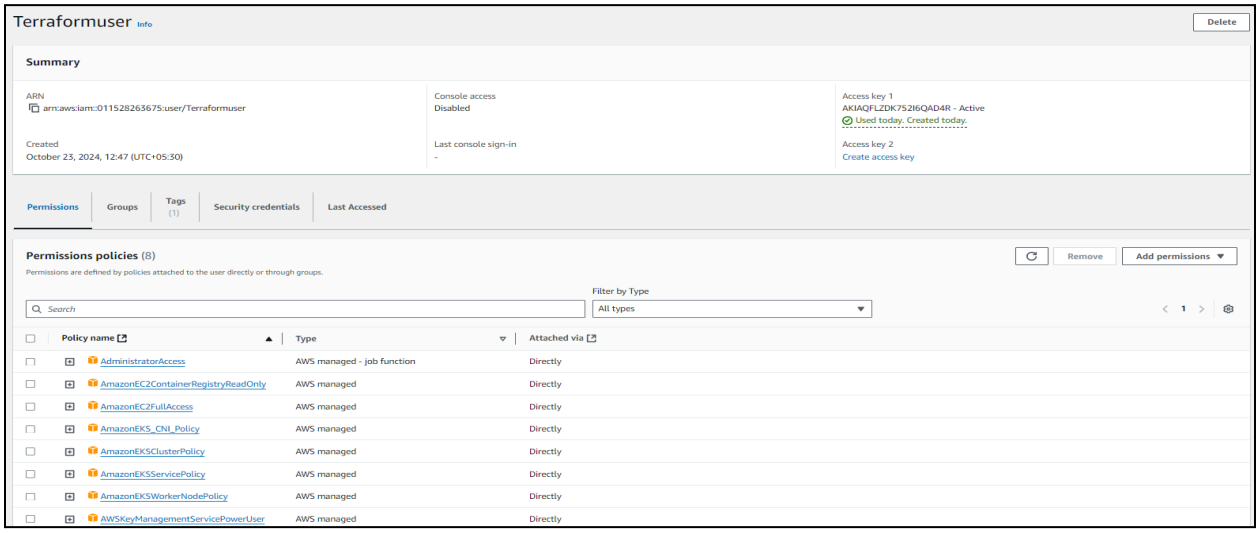


Setup for kubernetes on AWS:

Step-1: Setup the access and secret key in AWS CLI:

```
C:\Users\Ansh>aws configure
AWS Access Key ID [****************AD4R]: AKIAQFLZDK75216QAD4R
AWS Secret Access Key [****************+din]: at1hnz2NybB7ewjXo2US3vlPus8bBm1t7yRY+din
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

Step-2: Create IAM User and set up the required Policies:



Step-3: Create the main.tf script for deploying

instademo > terraform > main.tf > module "eks" > tags

```hcl
provider "aws" {
  region = "us-east-1"
}
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "~> 4.0"
  name = "my-vpc"
  cidr = "10.0.0.0/16"
  azs             = ["us-east-1a", "us-east-1b", "us-east-1c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true
}
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "~> 19.0"

  cluster_name    = "my-eks-cluster"
  cluster_version = "1.27"
  vpc_id     = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets
```

```
  eks_managed_node_group_defaults = {
    ami_type       = "AL2_x86_64"
    instance_types = ["t3.medium"]
  }
  eks_managed_node_groups = {
    example = {
      min_size     = 1
      max_size     = 3
      desired_size = 2
    }
  }
  tags = {
    Name = "my-eks-cluster"
  }
}
output "cluster_endpoint" {
  description = "Endpoint for EKS control plane"
  value       = module.eks.cluster_endpoint
}
output "cluster_name" {
  description = "Kubernetes Cluster Name"
  value       = module.eks.cluster_name
}
```

Step-4: Run terraform commands

- **Terraform init**
- **Terraform plan**
- **Terraform apply:**

```
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

aws_eks_cluster.my_eks_cluster: Creating...
aws_eks_cluster.my_eks_cluster: Still creating... [10s elapsed]
aws_eks_cluster.my_eks_cluster: Still creating... [20s elapsed]
aws_eks_cluster.my_eks_cluster: Still creating... [30s elapsed]
aws_eks_cluster.my_eks_cluster: Still creating... [40s elapsed]
aws_eks_cluster.my_eks_cluster: Still creating... [50s elapsed]
```

```
aws_eks_cluster.my_eks_cluster: Still creating... [8m41s elapsed]
aws_eks_cluster.my_eks_cluster: Still creating... [8m51s elapsed]
aws_eks_cluster.my_eks_cluster: Creation complete after 8m59s [id=my-eks-cluster]
aws_eks_node_group.my_eks_node_group: Creating...
kubernetes_deployment.nodejs_app: Creating...
aws_eks_node_group.my_eks_node_group: Still creating... [10s elapsed]
aws_eks_node_group.my_eks_node_group: Still creating... [20s elapsed]
```

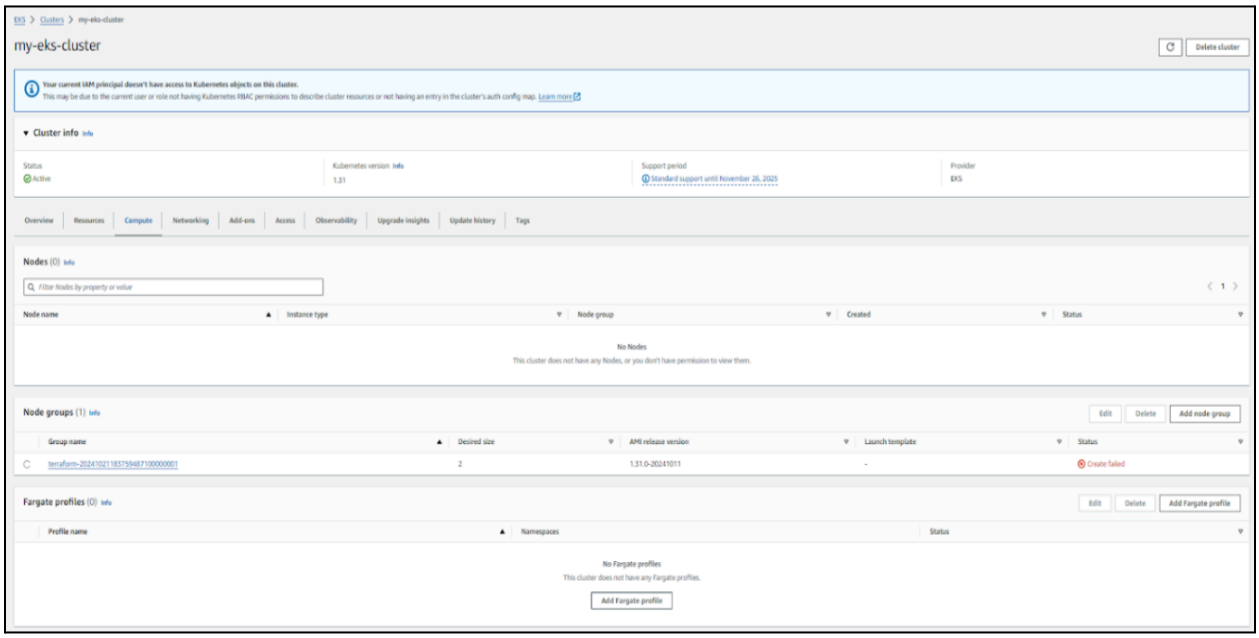
```
aws_eks_node_group.my_eks_node_group: Still creating... [22m26s elapsed]
aws_eks_node_group.my_eks_node_group: Still creating... [22m36s elapsed]
aws_eks_node_group.my_eks_node_group: Still creating... [22m46s elapsed]
aws_eks_node_group.my_eks_node_group: Still creating... [22m56s elapsed]
aws_eks_node_group.my_eks_node_group: Still creating... [23m7s elapsed]
aws_eks_node_group.my_eks_node_group: Still creating... [23m17s elapsed]

Error: waiting for EKS Node Group (my-eks-cluster:terraform-20241021183759487100000001) create: unexpected state 'CREATE_FAILED', wanted target 'ACTIVE'. last
i-01e0d376721c6ca1d, i-0c9e19e10eb34b17b: NodeCreationFailure: Instances failed to join the kubernetes cluster

  with aws_eks_node_group.my_eks_node_group,
  on main.tf line 28, in resource "aws_eks_node_group" "my_eks_node_group":
  28: resource "aws_eks_node_group" "my_eks_node_group" {


Error: Failed to create deployment: Post "http://localhost/apis/apps/v1/namespaces/default/deployments": dial tcp [::1]:80: connectex: No connection could be
cause the target machine actively refused it.

  with kubernetes_deployment.nodejs_app,
  on main.tf line 99, in resource "kubernetes_deployment" "nodejs_app":
  99: resource "kubernetes_deployment" "nodejs_app" {
```

Error:

Here in the error it is mentioned that no connection could be made because the target machine actively refused it so we cannot proceed further.

The cluster will be created in the EKS but the creation of node group is failed due to connection failure by target machine

EKS Cluster:



Setup for Minikube:

Step-1: Setting up minikube path and context in provider.tf

terraform > provider.tf > ...

```
provider "kubernetes" {
  config_path    = "~/.kube/config"
  config_context = "minikube"
}
```

Step-2: Create the main.tf script for deploying

orm > main.tf > resource "kubernetes_service" "nodejs_service" > spec > port

```
resource "kubernetes_deployment" "nodejs_deployment" {
  metadata {
    name      = "nodejs-deployment"
    namespace = "default"
  }
  spec {
    replicas = 1

    selector {
      match_labels = {
        app = "nodejs"
      }
    }
    template {
      metadata {
        labels = {
          app = "nodejs"
        }
      }
      spec {
        container {
          name  = "nodejs"
          image = "anshsarfare/nodejsapp:1.0"

          port {
            container_port = 3000
          }
        }
      }
    }
  }
}
resource "kubernetes_service" "nodejs_service" {
  metadata {
    name      = "nodejs-service"
    namespace = "default"
  }
  spec {
    selector = {
      app = "nodejs"
    }
    port {
      port        = 3000
      target_port = 3000
      protocol    = "TCP"
    }
    type = "NodePort"
  }
}
```

Step-3: Starting minikube in the terminal

```
C:\Users\Ansh>minikube start
* minikube v1.34.0 on Microsoft Windows 11 Home Single Language 10.0.22621.2715 Build 22621.2715
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.45 ...
* Restarting existing docker container for "minikube" ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: default-storageclass, storage-provisioner

! C:\Program Files\Docker\Docker\resources\bin\kubectl.exe is version 1.29.2, which may have incompatibilities with Kubernetes 1.31.0.
  - Want kubectl v1.31.0? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Step-4: Applying Terraform commands

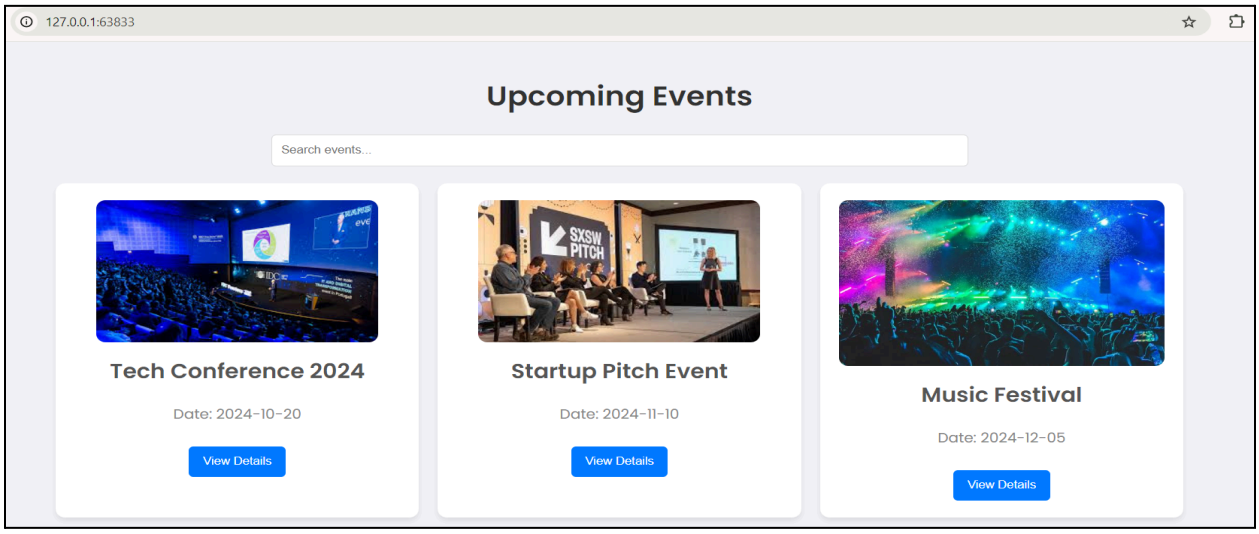
- **Terraform init**
- **Terraform plan**
- **Terraform apply:**

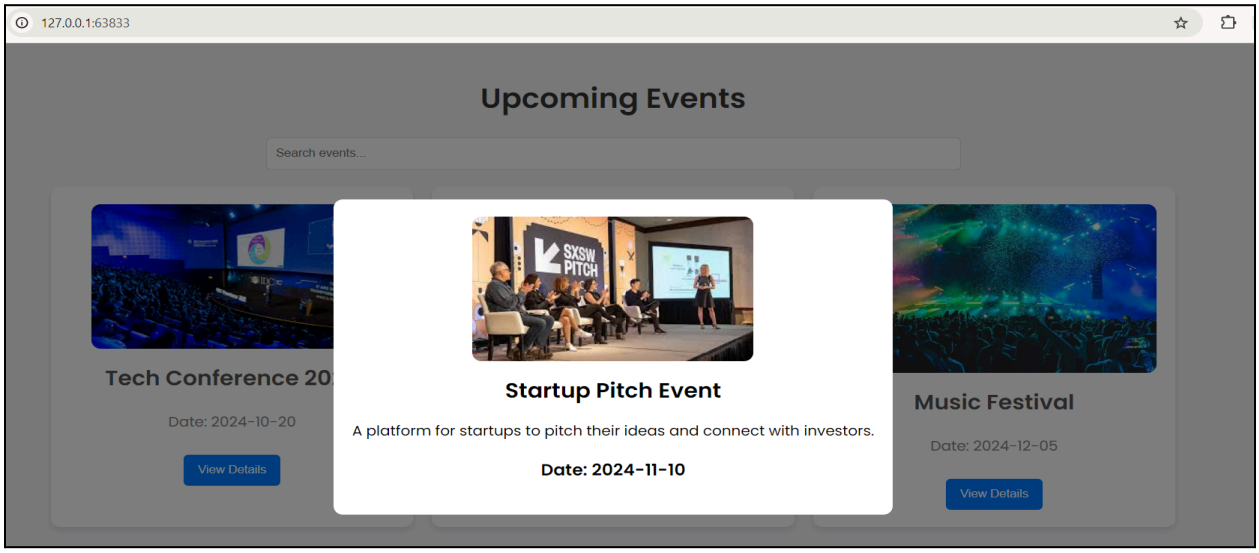
```
PS C:\Users\Ansh\Desktop\nodejsapp\terraform> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # kubernetes_deployment.nodejs_deployment will be created
  + resource "kubernetes_deployment" "nodejs_deployment" {
      + id               = (known after apply)
      + wait_for_rollout = true

      + metadata {
          + generation       = (known after apply)
          + name             = "nodejs-deployment"
          + namespace        = "default"
```

Step-5: run “minikube service nodejs-service –url” command to get the url to your node js app

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\Ansh\Desktop\nodejsapp\terraform> minikube service nodejs-service --url
http://127.0.0.1:63833
  !  Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

The node js app is successfully accessible on the url:





Step-6: run “terraform destroy” command to remove the deployed node js app from the cluster and “minikube stop” to stop it.

```
kubernetes_service.nodejs_service: Destroying... [id=default/nodejs-service]
kubernetes_deployment.nodejs_deployment: Destroying... [id=default/nodejs-deployment]
kubernetes_deployment.nodejs_deployment: Destruction complete after 0s
kubernetes_service.nodejs_service: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
```

```
C:\Users\Ansh>minikube stop
* Stopping node "minikube"  ...
* Powering off "minikube" via SSH ...
* 1 node stopped.
```

Conclusion:

This project showcased how we leveraged Terraform to automate and manage multi-cloud infrastructure, primarily focusing on AWS. We aimed to deploy a Kubernetes cluster on Amazon EKS for container orchestration and set up an S3 bucket for data storage. However, due to deployment errors, we were unable to complete the setup on EKS, but successfully deployed the application on Minikube in our local environment.

Key takeaways from this project include:

1. **Terraform's Flexibility**: Terraform simplified infrastructure management by enabling us to define resources as code, allowing for efficient provisioning, scaling, and management of services.
2. **Kubernetes Scalability**: Although the EKS deployment faced challenges, Minikube provided a local Kubernetes environment, demonstrating scalability and reliable application deployment.
3. **Cloud-native Storage with S3**: AWS S3 offered a scalable, durable storage solution, meeting the data needs of the deployed application.
4. **Troubleshooting Cluster Errors**: We gained valuable experience in diagnosing and resolving issues during cluster creation, particularly with EKS.
5. **Permission Policies**: We learned about the impact of various permission policies on Terraform and EKS, which will be useful in future projects.

This project underscored the importance of troubleshooting and adapting solutions to ensure successful deployment, even in challenging environments.