

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:

```
statichosting > provider.tf > provider "aws"

1 provider "aws" {
2   access_key = "ASIAVOX3ARPVO4XWJAMD"
3   secret_key = "240TwOVGws1YgqBakkm5++JWuYNXgy8zE4v65GIP"
4   token = "IQoJb3JpZ2luX2VjEFkaCXVzLXdlc3QtMiJHMEUCICYN14X/
   yzBQb022CYbC2VsnfyukT8QM7i5CeZgPxpzAiEAXA1Jkcjr7d4VEY/JAHqYkj1IBEmWK6+ARWL10eoA54qvgIIwv////////
   ARAAGgwzNzUyNjIzMTc1NDYiDL3vv0Wc2LCcn0vZsiqSagdS5h23AgECWkmY9QJHXkdY70J/
   cUkCsiy7W7CzeHwNz01DgYaRocMACRikdpcD00d3Vkw2pfu7/feSLEncEsY00x1u1Wh2p4q3j7N1Ud8qtS/Cdt11ECj0gut/
   EXT0JGR2IQKfg63Q370XEYhgaUvSmWmvQ+5aoHTESJc39DhdvXLltCEoWUZuZWOU4oHGbQ1/
   WXnzN16dlL2cMwn1Tnr01iLiOprBYSh1phmVNHFQ2ezxEbcWBSjqB+8Z+Qt/
   +c5v8EPG8JKX7eR5I3SUZav5ckbFLdbfrRwjRLP0WOGQTaD7vwc5MV297cvG7HCDi5zmi0ZjsaqwBJMnqr8/
   JqsZgocSBjEikLbtGY0UgdmxQUw/
   sjkuAY6nQGGJhA90E0Amuy6xjnwXIN5UaqHUB2zGNf1vbGJlzkgxIHQyGsyLNSsbv7NkDpqVzCCMEjjjnhCnPdHu+syYt9p
   +JacZxjfpOzWMy5ztkHUBNMR2Y2UxdLtroA0MMW28/Y9b0o1WtihdWUEhbTr/0d346F+6bZbdM1MIhLvwNKvNvwPLr1ftxdmli1CK
   +rY61YR0XyjaZK0YIF6MHubf"
5   region = "us-east-1"
6 }
```

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

```
statichosting > main.tf > ...
1  terraform {
2      required_providers {
3          aws = {
4              source = "hashicorp/aws"
5              version = "5.64.0"
6          }
7          random = {
8              source = "hashicorp/random"
9              version = "3.6.2"
10         }
11     }
12 }
13
14 resource "random_id" "rand_id" {
15     byte_length = 8
16 }
17
18 resource "aws_s3_bucket" "staticweb-bucket" {
19     bucket = "demo-bucket-${random_id.rand_id.hex}"
20 }
21
22
23 resource "aws_s3_object" "index_html" {
24     bucket = aws_s3_bucket.staticweb-bucket.bucket
25     source = "./index.html"
26     key    = "index.html"
27     content_type = "text/html"
28 }
29 resource "aws_s3_object" "style_css" {
30     bucket = aws_s3_bucket.staticweb-bucket.bucket
31     source = "./styles.css"
32     key    = "styles.css"
33     content_type = "text/css"
34 }
35
36 resource "aws_s3_bucket_public_access_block" "example" {
37     bucket = aws_s3_bucket.staticweb-bucket.bucket
38
39     block_public_acls      = false
40     block_public_policy    = false
41     ignore_public_acls    = false
42     restrict_public_buckets = false
43 }
44
```

```

45 resource "aws_s3_bucket_policy" "bucket-policy" {
46   bucket = aws_s3_bucket.staticweb-bucket.bucket
47   policy = jsonencode(
48     {
49       Version = "2012-10-17",
50       Statement = [
51         {
52           Sid = "PublicReadGetObject",
53           Effect = "Allow",
54           Principal = "*",
55           Action = [
56             "s3:GetObject"
57           ],
58           Resource = [
59             "arn:aws:s3:::${aws_s3_bucket.staticweb-bucket.id}/*"
60           ]
61         }
62       ]
63     }
64   )
65 }
66
67 resource "aws_s3_bucket_website_configuration" "example" {
68   bucket = aws_s3_bucket.staticweb-bucket.id
69
70
71   index_document {
72     suffix = "index.html"
73   }
74 }
75
76 output "website_endpoint" {
77   value = aws_s3_bucket_website_configuration.example.website_endpoint
78 }

```

Step-3:Run Terraform commands

- Terraform init
- Terraform plan
- Terraform apply

The bucket:

General purpose buckets					Directory buckets	
General purpose buckets (1) Info All AWS Regions					Refresh Copy ARN Empty Delete Create bucket	
Buckets are containers for data stored in S3.						
<input type="text" value="Find buckets by name"/>					< 1 > ⚙️	
Name	▲	AWS Region	▼	IAM Access Analyzer	Creation date	
demo-bucket-fa9f7f4593c2ef79	○	US East (N. Virginia) us-east-1		View analyzer for us-east-1	October 22, 2024, 11:24:52 (UTC+05:30)	

Objects in bucket:

demo-bucket-fa9f7f4593c2ef79

Info

Objects

Properties

Permissions

Metrics

Management

Access Points

Objects (2)

Info

Refresh

Copy S3 URI

Copy URL

Download

Open

Delete

Actions



Create folder

Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

< 1 > ⚙

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 index.html	html	October 22, 2024, 11:24:57 (UTC+05:30)	962.0 B	Standard
<input type="checkbox"/>	 styles.css	css	October 22, 2024, 11:24:57 (UTC+05:30)	1.5 KB	Standard

The Object Url in Object properties to access the page:

Properties

Permissions

Versions

Object overview

Owner

awslabsc0w3698936t1642941263

AWS Region

US East (N. Virginia) us-east-1

Last modified

October 22, 2024, 11:24:57 (UTC+05:30)


Size

962.0 B


Type

html


Key

 index.html


S3 URI

 s3://demo-bucket-fa9f7f4593c2ef79/index.html


Amazon Resource Name (ARN)

 arn:aws:s3::demo-bucket-fa9f7f4593c2ef79/index.html

Entity tag (Etag)

 1c78af27d904adba98c7fec952b03648

Object URL

 <https://demo-bucket-fa9f7f4593c2ef79.s3.us-east-1.amazonaws.com/index.html>

Hosted Static page:

demo-bucket-fa9f7f4593c2ef79.s3.us-east-1.amazonaws.com/index.html

☆ ⌵ ⬇ A ⋮

Discover the Future

Join us in shaping the future with innovative solutions.

Innovative Solutions for You

Explore our range of cutting-edge solutions designed to meet your needs and exceed your expectations.

Get Started

© 2024 Modern Web Solutions. All rights reserved.

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]: at1hnz2NybB7ewjXo2US3v1Pus8bBm1t7yRY+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:

Terraformuser [Info](#) [Delete](#)

Summary

ARN arn:aws:iam::011528263675:user/Terraformuser	Console access Disabled	Access key 1 AKIAQFLZDK75216QAD4R - Active Used today. Created today.
Created October 23, 2024, 12:47 (UTC+05:30)	Last console sign-in -	Access key 2 Create access key

Permissions | Groups | Tags (1) | Security credentials | Last Accessed

Permissions policies (8) [Refresh](#) [Remove](#) [Add permissions](#)

Permissions are defined by policies attached to the user directly or through groups.

Filter by Type: All types

<input type="checkbox"/>	Policy name	Type	Attached via
<input type="checkbox"/>	AdministratorAccess	AWS managed - job function	Directly
<input type="checkbox"/>	AmazonEC2ContainerRegistryReadOnly	AWS managed	Directly
<input type="checkbox"/>	AmazonEC2FullAccess	AWS managed	Directly
<input type="checkbox"/>	AmazonEKS_CNI_Policy	AWS managed	Directly
<input type="checkbox"/>	AmazonEKSClusterPolicy	AWS managed	Directly
<input type="checkbox"/>	AmazonEKSServicePolicy	AWS managed	Directly
<input type="checkbox"/>	AmazonEKSWorkerNodePolicy	AWS managed	Directly
<input type="checkbox"/>	AWSSystemsManagerServicePowerUser	AWS managed	Directly

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

```
instademo > terraform > main.tf > module "eks" > tags
1  provider "aws" {
2    region = "us-east-1"
3  }
4  module "vpc" {
5    source = "terraform-aws-modules/vpc/aws"
6    version = "~> 4.0"
7    name = "my-vpc"
8    cidr = "10.0.0.0/16"
9    azs = ["us-east-1a", "us-east-1b", "us-east-1c"]
10   private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
11   public_subnets = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
12   enable_nat_gateway = true
13   single_nat_gateway = true
14   enable_dns_hostnames = true
15 }
16 module "eks" {
17   source = "terraform-aws-modules/eks/aws"
18   version = "~> 19.0"
19
20   cluster_name = "my-eks-cluster"
21   cluster_version = "1.27"
22   vpc_id = module.vpc.vpc_id
23   subnet_ids = module.vpc.private_subnets
```

```

24     eks_managed_node_group_defaults = {
25         ami_type      = "AL2_x86_64"
26         instance_types = ["t3.medium"]
27     }
28     eks_managed_node_groups = {
29         example = {
30             min_size      = 1
31             max_size      = 3
32             desired_size = 2
33         }
34     }
35     tags = {
36         Name = "my-eks-cluster"
37     }
38 }
39 output "cluster_endpoint" {
40     description = "Endpoint for EKS control plane"
41     value       = module.eks.cluster_endpoint
42 }
43 output "cluster_name" {
44     description = "Kubernetes Cluster Name"
45     value       = module.eks.cluster_name
46 }

```

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-202410211837594871000000001) 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:

The screenshot displays the AWS Management Console for an EKS cluster named 'my-eks-cluster'. The cluster is in an 'Active' state. The 'Nodes' section shows 'No Nodes' with a message: 'This cluster does not have any nodes, or you don't have permission to view them.' The 'Node groups' section shows one node group in a 'Create failed' state. The 'Fargate profiles' section shows 'No Fargate profiles' with a message: 'This cluster does not have any Fargate profiles.'

Setup for Minikube:

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

```
terraform > provider.tf > ...  
1  provider "kubernetes" {  
2      config_path      = "~/.kube/config"  
3      config_context    = "minikube"  
4  }  
5
```

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:

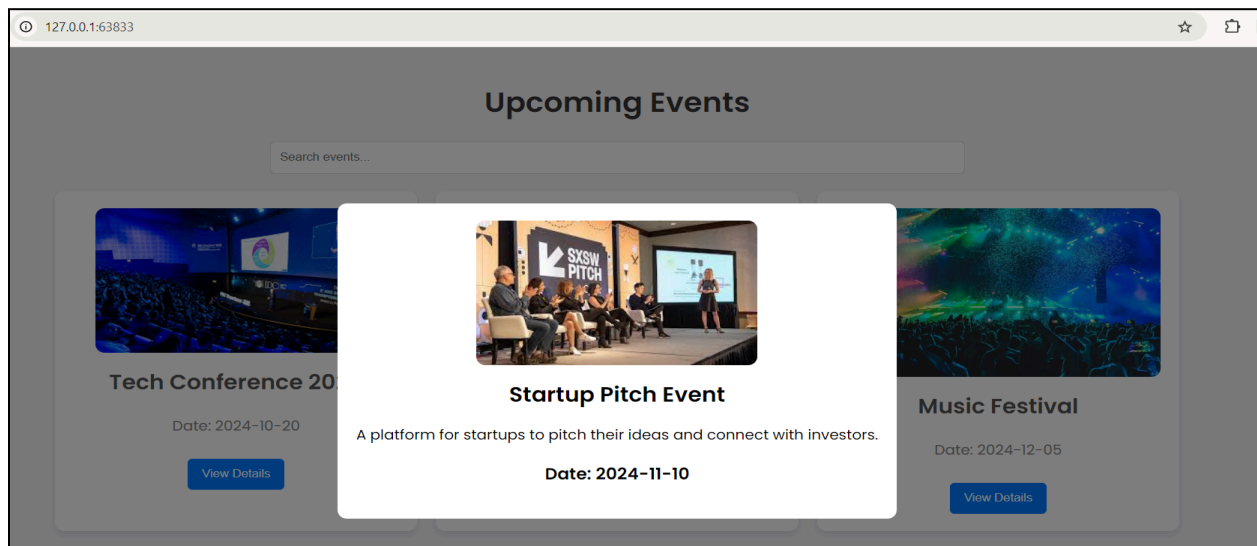
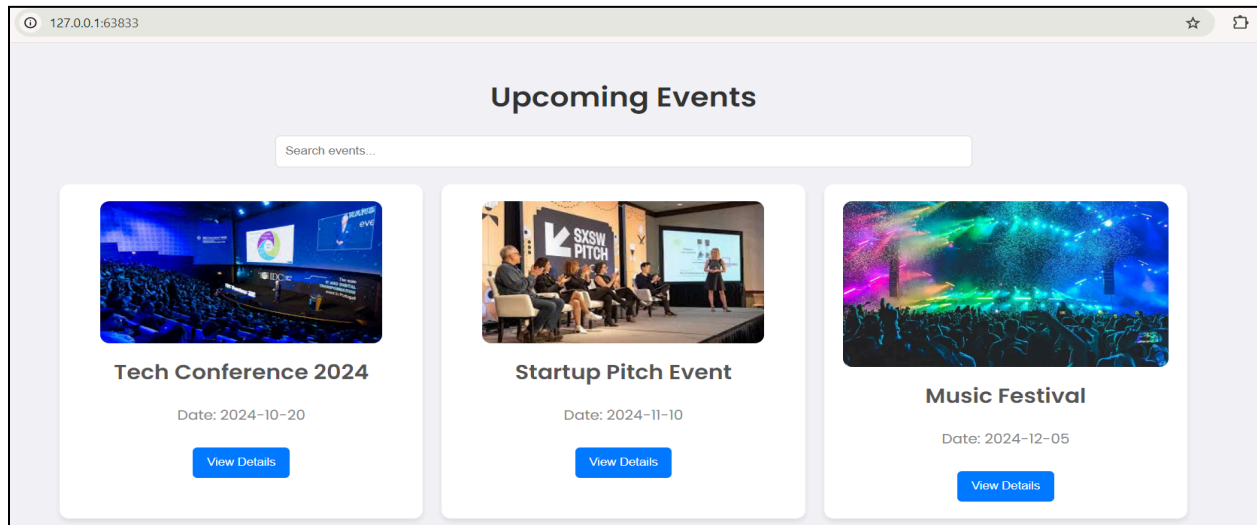
# kubernetes_deployment.nodejs_deployment will be created
+ resource "kubernetes_deployment" "nodejs_deployment" {
  + id              = (known after apply)
  + wait_for_rollout = true

  + 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.