

Building and Managing Multi-Cloud Infrastructure

Deploying Node.js application using Terraform , S3 and Kubernetes

1. Introduction

- **Case Study Overview:**

This case study centers on the development and management of multi-cloud infrastructure through the use of Terraform and Kubernetes on AWS. The goal is to automate the creation and management of resources, including an AWS S3 bucket and a Kubernetes cluster, utilizing Terraform scripts. Following this, a sample application will be deployed on the Kubernetes cluster to verify its operational capabilities.

- **Key features & Application**

A key feature of this project is the seamless integration of multi-cloud infrastructure management using Terraform. This is achieved by automating the setup and management of both storage and computing resources, specifically AWS S3 and Kubernetes, across different cloud services. The project demonstrates how Terraform makes infrastructure management easier by treating it as code, allowing for consistent application deployment in various cloud environments.

- **Third year Project Integration (optional):**

My third year project, 'Saarthi' aims to connect Nurses and care providers with patients and others who may require the services. Kubernetes deployment for the project will allow higher scalability leading to better customer reach. Also it will provide high availability, hence improving customer experience and reducing downtime. It can also help with load balancing and resource management. Furthermore it will allow CI/CD which will help with automated deployments/

2. Step by Step Explanation

1. Set Up Your Environment / Pre-requisites

- **Install Terraform:** Ensure Terraform is installed on your system.
- **Install AWS CLI:** Make sure the AWS CLI is set up and configured with your credentials.
- **Install Kubernetes Tools**

```
C:\Users\veyda>terraform -version
Terraform v1.9.8
on windows_amd64
```

```
C:\Users\veyda>aws --version
aws-cli/2.18.10 Python/3.12.6 Windows/10 exe/AMD64
```

```
C:\Users\veyda>kubect1 version --client
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

Add users and roles:

IAM User

I had Created a user called TerraformUser with the following permissions

Users (1) Info

Refresh

Delete

Create user

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Search

< 1 > ⚙

<input type="checkbox"/>	User name ▲	Path ▼	Group: ▼	Last activity ▼	MFA ▼	Password age ▼	Console last sign-in
<input type="checkbox"/>	TerraformUser	/	0	🟢 8 minutes ago	-	🟢 3 days	-

Permissions policies (7)

Refresh

Remove

Add permissions ▼

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

Search

Filter by Type

All types ▼

< 1 > ⚙

<input type="checkbox"/>	Policy name 📄 ▲	Type ▼	Attached via 📄
<input type="checkbox"/>	Addednew	Customer inline	Inline
<input type="checkbox"/>	AmazonEC2FullAccess	AWS managed	Directly
<input type="checkbox"/>	AmazonEKSClusterPolicy	AWS managed	Directly
<input type="checkbox"/>	AmazonS3FullAccess	AWS managed	Directly
<input type="checkbox"/>	CloudWatchLogsFullAccess	AWS managed	Directly
<input type="checkbox"/>	IAMFullAccess	AWS managed	Directly
<input type="checkbox"/>	PassEKSNODEGROUPRole	Customer managed	Directly

Permissions defined in this policy Info

Edit

Summary

JSON

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Search






Allow (5 of 423 services)

☐ Show remaining 418 services

Service ▲	Access level ▼	Resource	Request condition
EC2	Limited: List, Write, Tagging	All resources	None
EKS	Limited: List, Read, Write	All resources	None
Elastic Container Registry	Limited: Read, Write	All resources	None
IAM	Limited: Write	RoleName string like EKSNODEGROUPRole	None
STS	Limited: Read	All resources	None

IAM Role: (Used for EKS)

I created a role called EKSNodeGroupRole with the following permissions

Permissions policies (5) Info			
You can attach up to 10 managed policies.			
<input type="text" value="Search"/>		Filter by Type All types	< 1 > ⚙
<input type="checkbox"/>	Policy name ↗	Type	Attached entities
<input type="checkbox"/>	 AmazonEC2ContainerRegistryReadOnly	AWS managed	3
<input type="checkbox"/>	 AmazonEC2FullAccess	AWS managed	2
<input type="checkbox"/>	 AmazonEKS_CNI_Policy	AWS managed	3
<input type="checkbox"/>	 AmazonEKSWorkerNodePolicy	AWS managed	3
<input type="checkbox"/>	 new	Customer inline	0

2. Create an S3 Bucket and Kubernetes Cluster using terraform

create a directory, I named mine terraform-multi-cloud, change your directory to the one you just created.

```
cd Desktop
```

```
mkdir terraform-multi-cloud
```

```
main.tf
main.tf > ...
1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 5.0"
6     }
7     kubernetes = {
8       source = "hashicorp/kubernetes"
9       version = "~> 2.0"
10    }
11  }
12 }
13
14 provider "aws" {
15   region = "eu-north-1"
16   access_key = "AKIATQPD7SBWLD7OEY4X"
17   secret_key = "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"
18 }
19
20 resource "aws_vpc" "my_vpc" {
21   cidr_block = "10.0.0.0/16"
22 }
23
24 data "aws_availability_zones" "available" {}
25
26 resource "aws_subnet" "my_subnet" {
27   count = 2
28   vpc_id = aws_vpc.my_vpc.id
29   cidr_block = "10.0.${count.index}.0/24"
30   availability_zone = element(data.aws_availability_zones.available.names, count.index)
31 }
32
```

```

33 module "eks" {
34   source           = "terraform-aws-modules/eks/aws"
35   cluster_name     = "my-cluster"
36   cluster_version  = "1.24"
37   vpc_id           = aws_vpc.my_vpc.id
38   subnet_ids       = aws_subnet.my_subnet[*].id
39
40   tags = {
41     Environment = "dev"
42   }
43
44   create_cloudwatch_log_group = false
45 }
46
47
48 data "aws_s3_bucket" "my_bucket" {
49   bucket = "terraform-project-bucket-241533161580-oct2024"
50 }
51
52 output "s3_bucket_arn" {
53   value = data.aws_s3_bucket.my_bucket.arn
54 }
55

```

After adding the code, open terminal and switch to the new directory, run the following commands

terraform init

```

> .terraform
> .terraform.lock.hcl
> main.tf
{} terraform.tfstate
≡ terraform.tfstate.back..

```

terraform plan

```

C:\Users\veyda\Desktop\terraform-multi-cloud>terraform plan
module.eks.data.aws_caller_identity.current: Reading...
module.eks.module.kms.data.aws_caller_identity.current[0]: Reading...
module.eks.data.aws_partition.current: Reading...
data.aws_availability_zones.available: Reading...
data.aws_s3_bucket.my_bucket: Reading...
module.eks.module.kms.data.aws_partition.current[0]: Reading...
module.eks.data.aws_iam_policy_document.assume_role_policy[0]: Reading...
aws_vpc.my_vpc: Refreshing state... [id=vpc-09117ec96c191019d]
module.eks.data.aws_partition.current: Read complete after 0s [id=aws]
module.eks.module.kms.data.aws_partition.current[0]: Read complete after 0
module.eks.data.aws_iam_policy_document.assume_role_policy[0]: Read comple
module.eks.aws_iam_role.this[0]: Refreshing state... [id=my-cluster-cluste
module.eks.data.aws_caller_identity.current: Read complete after 0s [id=24

```

Terraform apply When it prompts to enter a value, type 'Yes'

```
C:\Users\veyda\Desktop\terraform-multi-cloud>terraform apply
data.aws_availability_zones.available: Reading...
module.eks.data.aws_caller_identity.current: Reading...
data.aws_s3_bucket.my_bucket: Reading...
module.eks.data.aws_iam_policy_document.assume_role_policy[0]: Reading...
module.eks.data.aws_partition.current: Reading...
module.eks.module.kms.data.aws_caller_identity.current[0]: Reading...
module.eks.module.kms.data.aws_partition.current[0]: Reading...
aws_vpc.my_vpc: Refreshing state... [id=vpc-09117ec96c191019d]
```

If the command execute successfully you get the following output

```
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

s3_bucket_arn = "arn:aws:s3:::terraform-project-bucket-241533161580-oct2024"
```

3. Kubernetes EKS Cluster & S3 Bucket

Successfully created a S3 bucket

The screenshot shows the AWS Management Console for the Amazon S3 service. The 'General purpose buckets' tab is active, showing a list of buckets. One bucket is listed:

Name	AWS Region	IAM Access Analyzer	Creation date
terraform-project-bucket-241533161580-oct2024	Europe (Stockholm) eu-north-1	View analyzer for eu-north-1	October 20, 2024, 08:46:32 (UTC+05:30)

I created a cluster called mycluster2 Assigned role of EKS group custom role Then created a node group with the same role

A new Kubernetes version is available for this cluster.

Upgrade version

[EKS](#) > [Clusters](#) > mycluster2

mycluster2

Delete cluster

Upgrade version

End of standard support for Kubernetes version 1.30 is July 28, 2025. On that date, your cluster will enter the extended support period with additional fees. For more information, see the [pricing page](#).

Upgrade now

Cluster info

Info

Status	Kubernetes version	Support period	Provider
Active	1.30	Standard support until July 28, 2025	EKS

Nodes (2)

Info

Filter Nodes by property or value

< 1 >

Node name	Instance type	Node group	Created	Status
ip-172-31-11-167.eu-north-1.compute.internal	t3.medium	node1	Created 6 hours ago	Ready
ip-172-31-29-114.eu-north-1.compute.internal	t3.medium	node1	Created 6 hours ago	Ready

Node groups (1)

Info

Edit

Delete

Add node group

Group name	Desired size	AMI release version	Launch template	Status
node1	2	1.30.4-20241011	-	Active

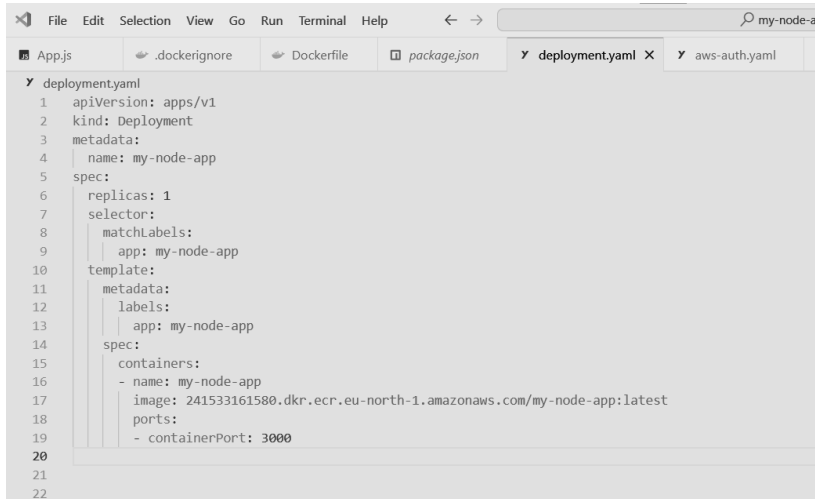
Now create Node.js Application

4. Create Node.js Application

App.js

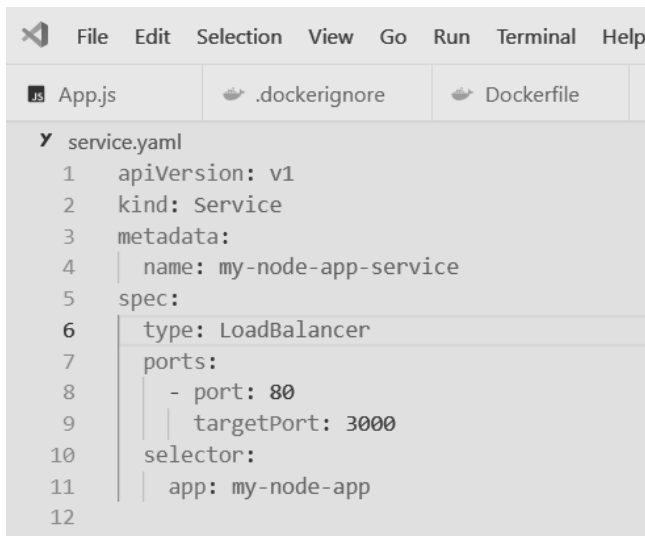
```
App.js > ...
1  const express = require('express');
2  const app = express();
3  const port = process.env.PORT || 3000;
4
5  app.get('/', (req, res) => {
6    res.send('Hello, World!');
7  });
8
9  app.listen(port, () => {
10   console.log(`Server running on port ${port}`);
11 });
12
```

deployment.yaml



```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: my-node-app
5 spec:
6   replicas: 1
7   selector:
8     matchLabels:
9       app: my-node-app
10  template:
11    metadata:
12      labels:
13        app: my-node-app
14    spec:
15      containers:
16        - name: my-node-app
17          image: 241533161580.dkr.ecr.eu-north-1.amazonaws.com/my-node-app:latest
18          ports:
19            - containerPort: 3000
```

Service.yaml



```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: my-node-app-service
5 spec:
6   type: LoadBalancer
7   ports:
8     - port: 80
9       targetPort: 3000
10  selector:
11    app: my-node-app
```

Create service and deployment yaml files, these files are necessary for the deployment

5. Approach: Adding Docker image to Amazon Elastic Container Registry:

Login to AWS ECR:

```
C:\Users\veyda\Desktop\my-node-app>docker login --username AWS --password-stdin 241533161580.dkr.ecr.eu-north-1.amazonaws.com < pass.txt
Login Succeeded
```

Then execute the following commands to build docker image , make sure Docker Desktop is running

```
C:\Users\veyda\Desktop\my-node-app>docker build -t my-node-app .
[+] Building 3.6s (11/11) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 408B                               0.0s
=> [internal] load metadata for docker.io/library/node:14         2.6s
=> [auth] library/node:pull token for registry-1.docker.io        0.0s
=> [internal] load .dockerignore                                  0.1s
=> => transferring context: 69B                                     0.0s
=> [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa 0.0s
=> [internal] load build context                                  0.1s
=> => transferring context: 2.15kB                                 0.0s
=> CACHED [2/5] WORKDIR /usr/src/app                               0.0s
=> CACHED [3/5] COPY package*.json ./                              0.0s
=> CACHED [4/5] RUN npm install                                    0.0s
=> [5/5] COPY . .                                                 0.2s
=> exporting to image                                              0.2s
=> => exporting layers                                             0.1s
=> => writing image sha256:ef10af7d4e1b77f7ac85e3dcfbbe801ff5881323efa791b7b0f0e056f2afd2ad 0.0s
=> => naming to docker.io/library/my-node-app                     0.0s

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\veyda\Desktop\my-node-app>docker tag my-node-app:latest 241533161580.dkr.ecr.eu-north-1.amazonaws.com/my-node-app:latest
```

Then Push the image to ECR:

```
C:\Users\veyda\Desktop\my-node-app>docker push 241533161580.dkr.ecr.eu-north-1.amazonaws.com/my-node-app:latest
The push refers to repository [241533161580.dkr.ecr.eu-north-1.amazonaws.com/my-node-app]
672a4d37395a: Pushed
b77a097565ee: Layer already exists
2cbfd73256b4: Layer already exists
9d8eca4ec15e: Layer already exists
0d5f5a015e5d: Layer already exists
3c777d951de2: Layer already exists
f8a91dd5fc84: Layer already exists
cb81227abde5: Layer already exists
e01a454893a9: Layer already exists
c45660adde37: Layer already exists
fe0fb3ab4a0f: Layer already exists
f1186e5061f2: Layer already exists
b2dba7477754: Layer already exists
latest: digest: sha256:102604de5635009bdecfb83c4982b01a36ffbd4c09eac3d206e9748776e1489 size: 3050

C:\Users\veyda\Desktop\my-node-app>
```

Amazon ECR > Private registry > Repositories

Private repositories

Create repository

Repositories (1)



View push commands

Delete

Actions ▼

Search by repository substring

	Repository name ▲	URI	Created at ▼	Tag immutability	Encryption type
<input type="radio"/>	my-node-app	241533161580.dkr.ecr.eu-north-1.amazonaws.com/my-node-app	October 20, 2024, 17:37:36 (UTC+05.5)	Mutable	AES-256

6. Kubernetes deployment

In a command prompt, switch to the directory where you created the node.js app and deployment and service files.

```
kubectl apply -f deployment.yaml
```

```
C:\Users\veyda\Desktop\my-node-app>kubectl apply -f deployment.yaml
error: error validating "deployment.yaml": error validating data: failed to download openapi: Get "https://127.0.0.1:49698/openapi/v2?timeout=32s": dial tcp 127.0.0.1:49698: connectex: No connection could be made because the target machine actively refused it.; if you choose to ignore these errors, turn validation off with --validate=false
```

Here we encountered an error due to the kubeconfig context, i.e it is located on minikube instead of cluster.

```
kubectl config current-context
```

```
C:\Users\veyda\Desktop\my-node-app>kubectl config get-contexts
CURRENT  NAME                                     CLUSTER
AUTHINFO  NAMESPACE
1  arn:aws:eks:eu-north-1:241533161580:cluster/my-cluster  arn:aws:eks:eu-north-1:241533161580:cluster/my-cluster
2  arn:aws:eks:eu-north-1:241533161580:cluster/mycluster2  arn:aws:eks:eu-north-1:241533161580:cluster/mycluster2
*  minikube                                                minikube
minikube                                                default
```

We update the context to our cluster using the eks command for updating the kubeconfig

```
kubectl config use-context arn:aws:eks:eu-north-1:241533161580:cluster/mycluster2
```

```
C:\Users\veyda\Desktop\my-node-app>aws eks update-kubeconfig --name mycluster2 --region eu-north-1
Updated context arn:aws:eks:eu-north-1:241533161580:cluster/mycluster2 in C:\Users\veyda\.kube\config
```

The error occurs with deployment.yaml not being accepted by EKS Cluster and also a failure to fetch the pods

```
kubectl get pods
```

```
C:\Users\veyda\Desktop\my-node-app>kubectl get pods
E1022 18:25:23.435792 14632 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credentials"
E1022 18:25:25.929634 14632 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credentials"
^C
C:\Users\veyda\Desktop\my-node-app>Exception ignored in: <_io.TextIOWrapper name='<stdout>' mode='w' encoding='cp1252'>
OSError: [Errno 22] Invalid argument
```

Apply the Deployment and Service:

Run the following commands:

```
kubectl apply -f deployment.yaml
```

```
kubectl apply -f service.yaml
```

Our configuration files get accepted

7. Verify the Deployment

Check the Service Status: Use the following command to retrieve details about your service:

```
kubectl get svc hello-world-node-service
```

Find the External IP: In the output, locate the **EXTERNAL-IP** column. Note that it might take a few minutes for the external IP to be assigned.

Access Your Application: Once the external IP is available, open a web browser and enter the URL:

```
http://<EXTERNAL-IP>/
```

You should see a "Hello, World!" message displayed.

8. Clean Up Resources

After testing, you may want to remove the resources to avoid incurring charges.

Use the following commands to destroy a particular resource

```
C:\Users\veyda\Desktop\terraform-multi-cloud>del *.tfstate  
C:\Users\veyda\Desktop\terraform-multi-cloud>del *.tfstate.backup
```

Run Terraform Destroy:

```
C:\Users\veyda\Desktop\terraform-multi-cloud>terraform destroy  
No changes. No objects need to be destroyed.  
  
Either you have not created any objects yet or the existing objects were already deleted outside of Terraform.  
Destroy complete! Resources: 0 destroyed.  
C:\Users\veyda\Desktop\terraform-multi-cloud>_
```

Conclusion:

This project demonstrated how we leveraged Terraform to automate the setup and management of multi-cloud resources, specifically on AWS. Our approach included creating an S3 bucket for data storage and establishing a Kubernetes cluster on AWS to handle container orchestration. By deploying a sample application, like a Node.js app, on the Kubernetes cluster, we highlighted the effective integration of infrastructure and applications across cloud environments.

Key insights from this project include:

1. Flexibility of Terraform: Terraform streamlines infrastructure management by allowing us to define resources as code, enabling efficient deployment, management, and scaling of services.
2. Kubernetes Scalability: The Kubernetes cluster created an optimal environment for deploying and scaling applications, ensuring consistent and reliable performance.
3. Cloud-native Storage with S3: The AWS S3 bucket provided a scalable and durable solution for storing data, meeting the needs of the deployed application.
4. Error Resolution in Cluster Creation: We also learned how to troubleshoot and resolve errors encountered during the creation of the cluster.
5. Amazon ECR: We also worked and learnt about Amazon ECR service that allows to upload docker images for project, and can be used for further deployment.

We also learned about various Permission policies and their effect on Terraform, EKS, and ECR.