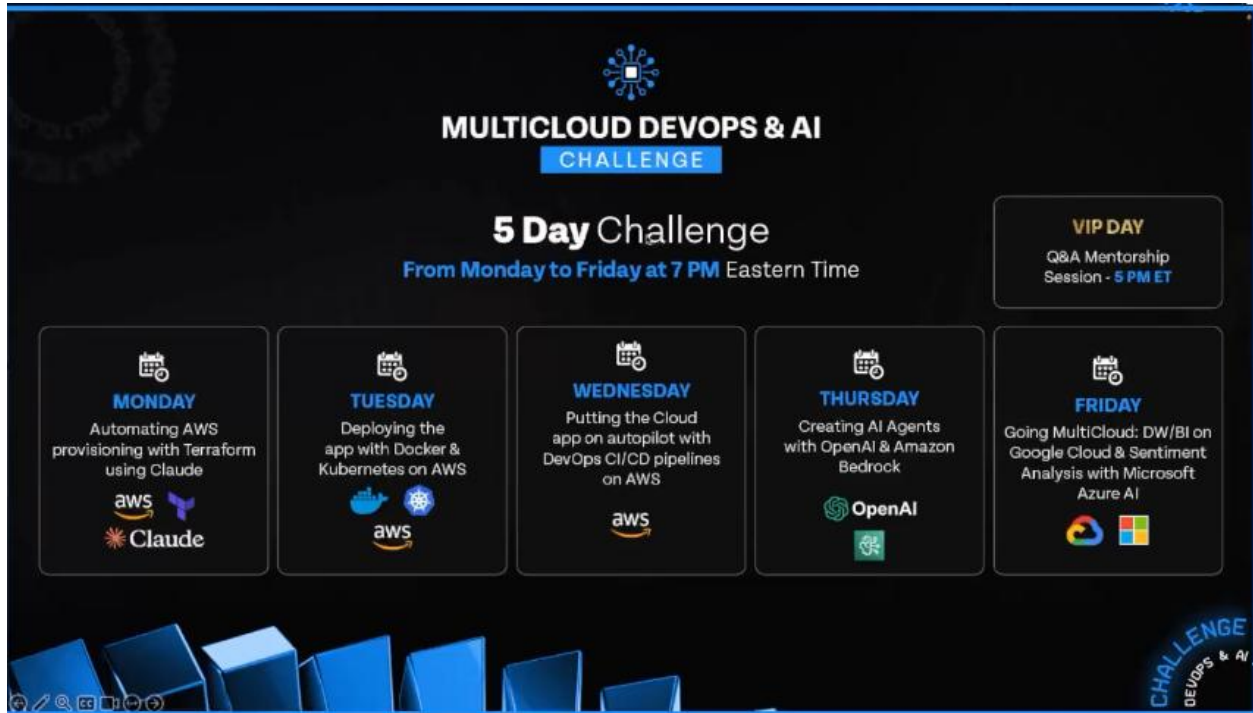


# MultiCloud, DevOps & AI Challenge - Day 1 Report



## AUTOMATING AWS provisioning with Terraform using Claude

### Introduction

The first day of the MultiCloud, DevOps & AI Challenge focused on leveraging AI assistance to streamline Terraform deployments. The goal was to use Claude AI to generate Terraform scripts, configure AWS infrastructure, and deploy resources efficiently. This report outlines the key steps undertaken, challenges faced, and outcomes achieved.

---

### Step 1: Using Claude to Generate Terraform Code

Claude AI was utilized to generate Terraform code for creating an AWS S3 bucket. The code was used later on. The following steps were performed:

NB: It is a good behaviour to create a rapport with my new friend, Claude. She is the best when it comes to code test or taste. She has a good taste.

1. Initiated a conversation with Claude AI.
2. Requested Terraform code for creating an S3 bucket with a unique name.
3. Claude provided the following Terraform script:

```
provider "aws" {
  region = "us-west-2" # Replace with your desired region
}

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

resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-bucket-name-${random_id.bucket_suffix.hex}"

  tags = {
    Name       = "My bucket"
    Environment = "Dev"
  }
}

resource "aws_s3_bucket_acl" "my_bucket_acl" {
  bucket = aws_s3_bucket.my_bucket.id
  acl    = "private"
}
```

The code was saved for later use in Terraform deployment.

---

## Step 2: Creating an IAM Role for EC2

This process was easy-peasy of course who doesn't have an I am. I am who I am

1. Logged into the AWS Management Console.
2. Navigated to IAM > Roles > Create Role.
3. Selected "AWS Service" as the trusted entity and chose "EC2" as the use case.
4. Attached the "AdministratorAccess" policy (not recommended for production use).
5. Named the role **EC2Admin** and created it.

---

## Step 3: Launching an EC2 Instance

1. Opened the EC2 dashboard and clicked "Launch Instance."

2. Selected **Amazon Linux 2 AMI**.
3. Chose **t2.micro** as the instance type.
4. Configured instance details:
  - o Default VPC, any available subnet
  - o Enabled auto-assign public IP
  - o Assigned IAM role **EC2Admin**
5. Created a security group allowing SSH access.
6. Added a tag: **Key=Name, Value=workstation**.
7. Launched the instance and associated a key pair.

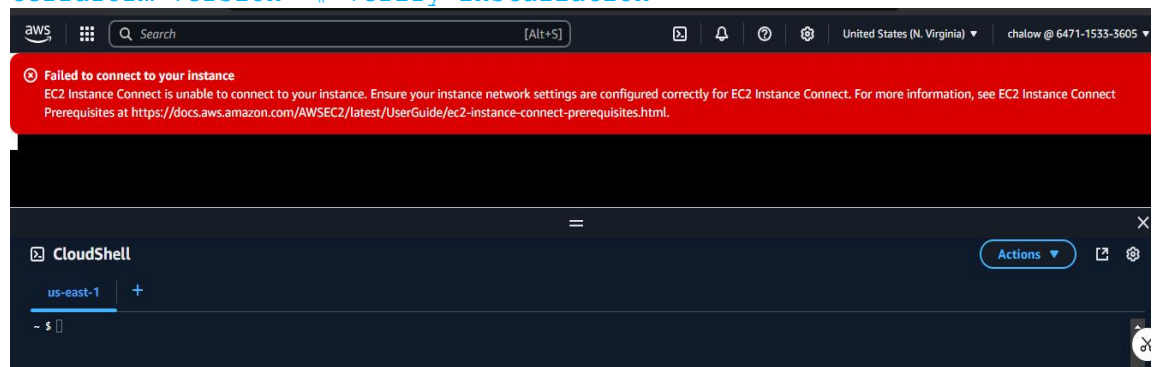
Are you still expecting screenshots HUREE VOOOIIILLAA, now we have our workstation ready to roll.

---

## Step 4: Connecting to EC2 and Installing Terraform

**I almost cried, cratching my head for quite some time. I encountered a lot of error, reason being I did not configure roles and policies well.**

1. Connected to the instance using EC2 Instance Connect.
2. Ran the following commands to install Terraform:
  1. `sudo yum update -y`
  2. `sudo yum install -y yum-utils`
  3. `sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo`
  4. `sudo yum -y install terraform`
  5. `terraform version` # Verify installation



I encountered an error while trying to connect to my EC2 instance using EC2 Instance Connect, which displayed the message: **"Failed to connect to your instance. Ensure your instance network settings are configured correctly."** To resolve this, I first checked the **security group settings** and confirmed that **inbound rules allowed SSH (port 22) from my IP address**. Next, I verified that the instance was in a **public subnet with an associated internet gateway** and had **Auto-assign Public IP enabled**. Additionally, I ensured that the **IAM role attached to the instance had EC2InstanceConnect permissions**. After making these adjustments, I was able to successfully connect to my instance.

```
~\###|  
~/V- -> https://aws.amazon.com/linux/amazon-linux-2023  
Last login: Tue Feb 25 13:19:50 2025 from 18.206.107.28  
[ec2-user@ip-172-31-29-158 ~]$ sudo yum update -y  
Last metadata expiration check: 0:16:28 ago on Tue Feb 25 13:07:37 2025.  
Dependencies resolved.  
Nothing to do.  
Complete!  
[ec2-user@ip-172-31-29-158 ~]$ sudo yum install -y yum-utils  
Last metadata expiration check: 0:17:42 ago on Tue Feb 25 13:07:37 2025.  
Package dnf-utils-4.3.0-13.amzn2023.0.5.noarch is already installed.  
Dependencies resolved.  
Nothing to do.  
Complete!  
[ec2-user@ip-172-31-29-158 ~]$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo  
Adding repo from: https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo  
[ec2-user@ip-172-31-29-158 ~]$ sudo yum -y install terraform  
Hashicorp Stable - x86_64                                23 MB/s | 1.6 MB    00:00  
Last metadata expiration check: 0:00:01 ago on Tue Feb 25 13:26:57 2025.  
Dependencies resolved.
```

Package	Architecture	Version	Repository	Size
=				

```
[CloudShell] Feedback [Alt+S]  
  
[ec2-user@ip-172-31-29-158 ~]$ terraform init  
Initializing the backend...  
Initializing provider plugins..  
- Finding latest version of hashicorp/random...  
- Finding latest version of hashicorp/aws...  
- Installing hashicorp/random v3.6.3...  
- Installed hashicorp/random v3.6.3 (signed by HashiCorp)  
- Installing hashicorp/aws v5.88.0...  
- Installed hashicorp/aws v5.88.0 (signed by HashiCorp)  
Terraform has created a lock file .terraform.lock.hcl to record the provider  
selections it made above. Include this file in your version control repository  
so that Terraform can guarantee to make the same selections by default when  
you run "terraform init" in the future.  
  
Terraform has been successfully initialized!
```

```
You may now begin working with Terraform. Try running "terraform plan" to see  
any changes that are required for your infrastructure. All Terraform commands  
should now work.  
  
If you ever set or change modules or backend configuration for Terraform,  
rerun this command to reinitialize your working directory. If you forget, other  
commands will detect it and remind you to do so if necessary.  
[ec2-user@ip-172-31-29-158 ~]$
```

**Hurreeeeeeee I am in**

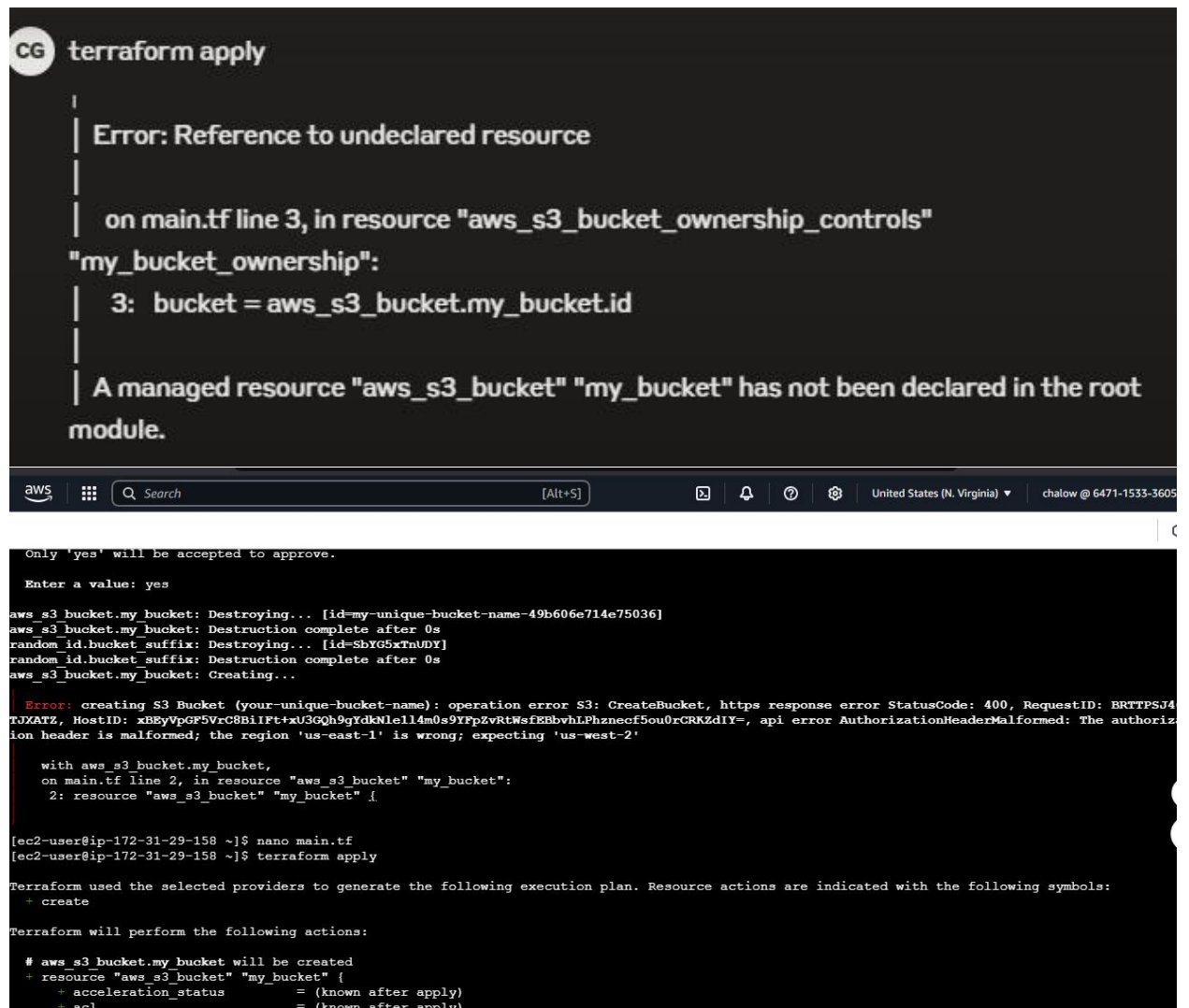
## Step 5: Applying Terraform Configuration

1. Created a new directory and navigated to it:
2. `mkdir terraform-project && cd terraform-project`
3. Opened `main.tf` and pasted the Terraform code generated by Claude.
4. Initialized Terraform:
5. `terraform init`
6. Reviewed and applied the plan:
7. `terraform plan`
8. `terraform apply` # Typed "yes" when prompted
9. Verified bucket creation using AWS CLI:

## 10. aws s3 ls

NB: Also encountered errors but its for our own good that's how we learn it was expecting US-WEST-2 but my instance was on US-EAST-1.

Abd also trying to reference a resource that has not been declared yet but again who is our friend she is called Claude. You got that right



```
CG terraform apply

Error: Reference to undeclared resource

on main.tf line 3, in resource "aws_s3_bucket_ownership_controls"
"my_bucket_ownership":
  3: bucket = aws_s3_bucket.my_bucket.id

A managed resource "aws_s3_bucket" "my_bucket" has not been declared in the root
module.

aws s3 bucket.my_bucket: Destroying... [id=my-unique-bucket-name-49b606e714e75036]
aws s3 bucket.my_bucket: Destruction complete after 0s
random_id.bucket_suffix: Destroying... [id=SbY65xTnUDY]
random_id.bucket_suffix: Destruction complete after 0s
aws_s3_bucket.my_bucket: Creating...

Error: creating S3 Bucket (your-unique-bucket-name): operation error S3: CreateBucket, https response error StatusCode: 400, RequestID: BRITPSJ4
TJXATZ, HostID: xBEyVpGF5VrC8BilFt+xU3GQh9gYdkNle1l4m0s9YfpZvRtWsfEBbvhlPhznecf5ou0rCRKZdIY-, api error AuthorizationHeaderMalformed: The authoriz
ion header is malformed; the region 'us-east-1' is wrong; expecting 'us-west-2'

with aws_s3_bucket.my_bucket,
on main.tf line 2, in resource "aws_s3_bucket" "my_bucket":
  2: resource "aws_s3_bucket" "my_bucket" {

[ec2-user@ip-172-31-29-158 ~]$ nano main.tf
[ec2-user@ip-172-31-29-158 ~]$ 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:

# aws_s3_bucket.my_bucket will be created
+ resource "aws_s3_bucket" "my_bucket" {
+   acceleration_status = (known after apply)
+   acl                  = (known after apply)
```

Let us see what Claude has for us

It looks like you're trying to run `terraform apply` but you've encountered an error. The error indicates that you're trying to reference a resource `aws_s3_bucket.my_bucket` that hasn't been declared yet.

In your configuration file (`main.tf`), you have a resource for `aws_s3_bucket_ownership_controls` that depends on an S3 bucket, but the bucket resource itself is missing.

To fix this, you need to add the S3 bucket resource declaration to your configuration before referencing it. Here's how you can do it:

And She gave us the code

```
hcl Copy

# First, declare the S3 bucket resource
resource "aws_s3_bucket" "my_bucket" {
  bucket = "your-unique-bucket-name" # Optional: specify a name or let AWS generate one
  # Add other bucket configurations as needed
}

# Then, declare the bucket ownership controls
resource "aws_s3_bucket_ownership_controls" "my_bucket_ownership" {
  bucket = aws_s3_bucket.my_bucket.id
  rule {
    object_ownership = "BucketOwnerPreferred" # Or another valid value
  }
}
```

In the bucket name it must be unique so I changed it. And I managed to solve the issue. A sip of my coffee



```
aws Search [Alt+S] United States (N. Virginia) chalow @ 6471-1533-3605

+ website (known after apply)
}

# aws s3 bucket ownership controls.my_bucket_ownership will be created
+ resource "aws_s3_bucket_ownership_controls" "my_bucket_ownership" {
+   bucket = (known after apply)
+   id     = (known after apply)

+   rule {
+     object_ownership = "BucketOwnerPreferred"
+   }
+ }

Plan: 2 to add, 0 to change, 0 to destroy.

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 s3 bucket.my_bucket: Creating...
aws s3 bucket.my_bucket: Creation complete after 3s [id=chalow-me]
aws s3 bucket_ownership_controls.my_bucket_ownership: Creating...
aws s3 bucket_ownership_controls.my_bucket_ownership: Creation complete after 0s [id=chalow-me]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
[ec2-user@ip-172-31-29-158 ~]$ aws s3 ls
2025-02-26 05:08:26 chalow-me
[ec2-user@ip-172-31-29-158 ~]$
```

## Step 6: Creating DynamoDB Tables

1. Modified `main.tf` to replace S3 configurations with the following DynamoDB table definitions:

```
Shell Copy

provider "aws" {
  region = "us-east-1"
}

# Tables DynamoDB
resource "aws_dynamodb_table" "cloudmart_products" {
  name         = "cloudmart-products"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "id"

  attribute {
    name = "id"
    type = "S"
  }
}

resource "aws_dynamodb_table" "cloudmart_orders" {
  name         = "cloudmart-orders"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "id"
}
```

```

    attribute {
      name = "id"
      type = "S"
    }
  }

resource "aws_dynamodb_table" "cloudmart_tickets" {
  name           = "cloudmart-tickets"
  billing_mode    = "PAY_PER_REQUEST"
  hash_key       = "id"

  attribute {
    name = "id"
    type = "S"
  }
}

```

Again policies issues, consequences? Errors

```

Only 'yes' will be accepted to approve.

Enter a value: yes

aws_dynamodb_table.cloudmart_products: Creating...
aws_s3_bucket.ownership_controls.my_bucket_ownership: Destroying... [id=challow-me]
aws_dynamodb_table.cloudmart_orders: Creating...
aws_dynamodb_table.cloudmart_tickets: Creating...
aws_s3_bucket.ownership_controls.my_bucket_ownership: Destruction complete after 1s
aws_s3_bucket.my_bucket: Destroying... [id=challow-me]
aws_s3_bucket.my_bucket: Destruction complete after 0s

Error: creating AWS DynamoDB Table (cloudmart-products): operation error DynamoDB: CreateTable, https response error StatusCode: 400, RequestID: BHM644JN4O6VDR3UCJMQ5SOPUFVV4KQNSO5ARMVJP66Q9ASUAAJG, api error AccessDeniedException: User: arn:aws:sts::647115333605:assumed-role/ec2s3/i-0763d40ede70acbaa is not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-west-2:647115333605:table/cloudmart-products because no identity-based policy allows the dynamodb:CreateTable action

with aws_dynamodb_table.cloudmart_products,
on main.tf line 7, in resource "aws_dynamodb_table" "cloudmart_products":
7: resource "aws_dynamodb_table" "cloudmart_products" {}

Error: creating AWS DynamoDB Table (cloudmart-orders): operation error DynamoDB: CreateTable, https response error StatusCode: 400, RequestID: 28H3R1ON29VFTNCNMRGU17NS53VV4KQNSO5ARMVJP66Q9ASUAAJG, api error AccessDeniedException: User: arn:aws:sts::647115333605:assumed-role/ec2s3/i-0763d40ede70acbaa is not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-west-2:647115333605:table/cloudmart-orders because no identity-based policy allows the dynamodb:CreateTable action

with aws_dynamodb_table.cloudmart_orders,
on main.tf line 18, in resource "aws_dynamodb_table" "cloudmart_orders":
18: resource "aws_dynamodb_table" "cloudmart_orders" {}

```

But luckily I have my Claude she is always there to help.....



Now you're facing a different issue - this is a permissions problem. Your error message shows that your AWS credentials (running under the assumed role "ec2s3") don't have permission to create DynamoDB tables.

The specific error is:

This means the IAM role "ec2s3" that your EC2 instance is using doesn't have the necessary permissions to create DynamoDB tables. You have a few options:

**1. Add DynamoDB permissions to your role:**

- Go to the AWS IAM console
- Find the role "ec2s3"
- Add the `AmazonDynamoDBFullAccess` policy (or a more limited custom policy that includes `dynamodb:CreateTable` )

**2. Use a different AWS credential** that has the required permissions

**3. Request permissions** from your AWS administrator if you don't have access to modify IAM roles

Since you appear to be running this from an EC2 instance (based on the instance ID in the error), the instance was launched with an IAM role that only has S3 permissions, not DynamoDB permissions. This typically requires administrator intervention to fix.

Would you like me to help you create a custom IAM policy that would grant just the DynamoDB permissions you need?

And VOILAAAA we are in.... the terminal

```
+ attribute {
+   name = "id"
+   type = "s"
+ }

+ point_in_time_recovery (known after apply)
+ server_side_encryption (known after apply)
+ ttl (known after apply)
}

Plan: 3 to add, 0 to change, 0 to destroy.

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_dynamodb_table.cloudmart_orders: Creating...
aws_dynamodb_table.cloudmart_tickets: Creating...
aws_dynamodb_table.cloudmart_products: Creating...
aws_dynamodb_table.cloudmart_products: Creation complete after 7s [id=cloudmart-products]
aws_dynamodb_table.cloudmart_orders: Creation complete after 8s [id=cloudmart-orders]
aws_dynamodb_table.cloudmart_tickets: Still creating... [10s elapsed]
aws_dynamodb_table.cloudmart_tickets: Creation complete after 14s [id=cloudmart-tickets]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
[ec2-user@ip-172-31-29-158 ~]$
```

2. Applied the updated Terraform configuration:
3. terraform apply
4. Confirmed successful deployment.

## Challenges and Lessons Learned

- **Challenge:** Setting up AWS IAM permissions correctly.
  - **Solution:** Used the **AdministratorAccess** policy temporarily but noted that least privilege policies should be implemented in production.
- **Challenge:** Ensuring Terraform installation was successful.
  - **Solution:** Verified installation using terraform version and reinstalled when necessary.
- **Lesson Learned:** Terraform simplifies infrastructure deployment significantly, and AI assistance can enhance efficiency in writing configurations.

---

## Conclusion

Day 1 of the MultiCloud, DevOps & AI Challenge was a success. Claude AI streamlined Terraform code generation, AWS resources were provisioned

efficiently, and both an S3 bucket and DynamoDB tables were created successfully. This experience reinforced the power of AI-driven DevOps automation.