Here's a step-by-step tutorial on how to create and manage an AWS DynamoDB table using Terraform.

Prerequisites

1. \*\*AWS Account\*\*: You need an AWS account.

2. \*\*AWS CLI\*\*: Ensure AWS CLI is installed and configured.

3. \*\*Terraform\*\*: Install Terraform on your machine.

4. \*\*IAM Role\*\*: Ensure your IAM role/user has permissions to manage DynamoDB.

Step 1: Install Terraform

If you haven't installed Terraform yet, you can download it from the [official Terraform website](https://www.terraform.io/downloads.html) and follow the installation instructions for your OS.

Step 2: Setup a Working Directory

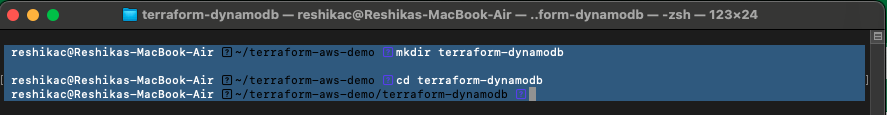
Create a directory for your Terraform project.

```bash

mkdir terraform-dynamodb

cd terraform-dynamodb

```



Step 3: Create a `main.tf` File

Create a `main.tf` file, which will define your infrastructure.

```bash

touch main.tf

```

Step 4: Define the Provider

In the `main.tf` file, start by defining the AWS provider:

```hcl

provider "aws" {

region = "us-west-2"

}

```

Replace `"us-west-2"` with your desired AWS region.

Step 5: Create a DynamoDB Table

Next, define the DynamoDB table resource:

```hcl

resource "aws\_dynamodb\_table" "example" {

name = "example-table"

billing\_mode = "PAY\_PER\_REQUEST"

hash\_key = "id"

attribute {

name = "id"

type = "S"

}

tags = {

Name = "example-table"

Environment = "dev"

}

}

```

main.tf

provider "aws" {

region = var.region

}

resource "aws\_dynamodb\_table" "table\_demo" {

name = "example-table"

billing\_mode = "PAY\_PER\_REQUEST"

hash\_key = "id"

attribute {

name = "id"

type = "S"

}

tags = {

Name = var.aws\_tags[1]

Environment = var.aws\_tags[0]

}

}

variables.tf

variable "region" {

description = "Region"

default = "us-west-2"

}

variable "aws\_tags" {

description = "tags"

type = list(string)

default = ["Dev", "example-table", "DynamoDB Demo"]

}

This configuration creates a DynamoDB table named `example-table` with a primary key attribute `id` of type `String`.

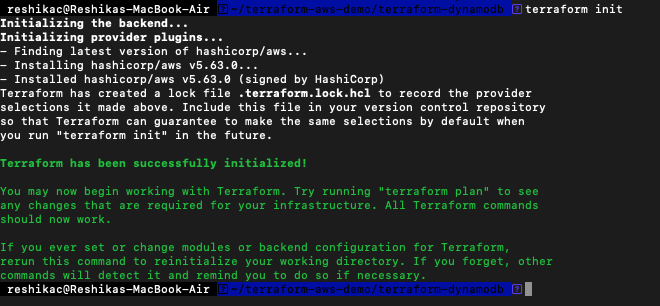
Step 6: Initialize Terraform

Before applying the configuration, initialize Terraform to download the required providers.

```bash

terraform init

```



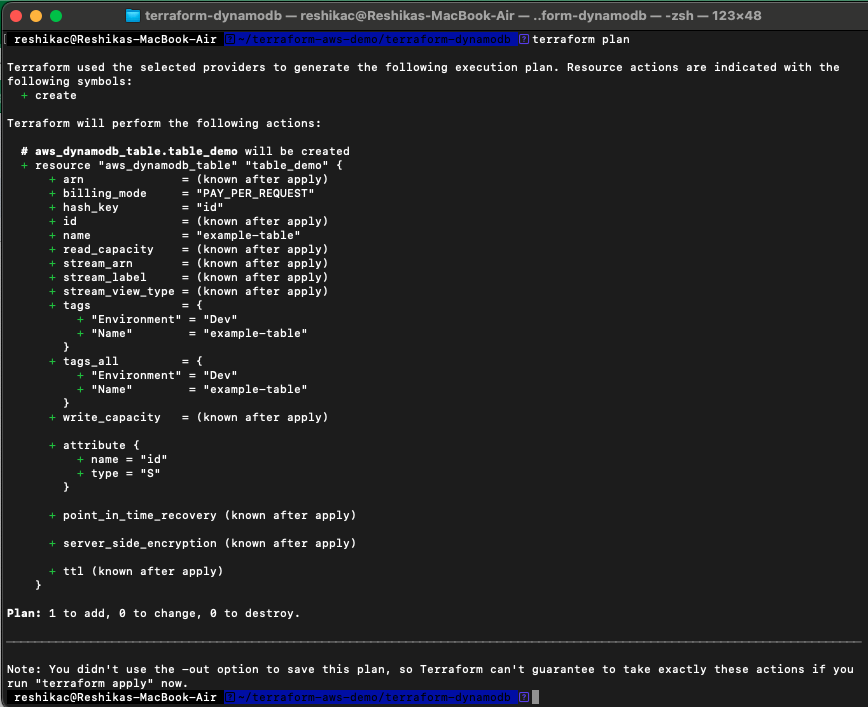
Step 7: Plan the Changes

Use the `terraform plan` command to see what Terraform will do when you apply the configuration.

```bash

terraform plan

```



This command helps you verify that the configuration is correct before making any changes.

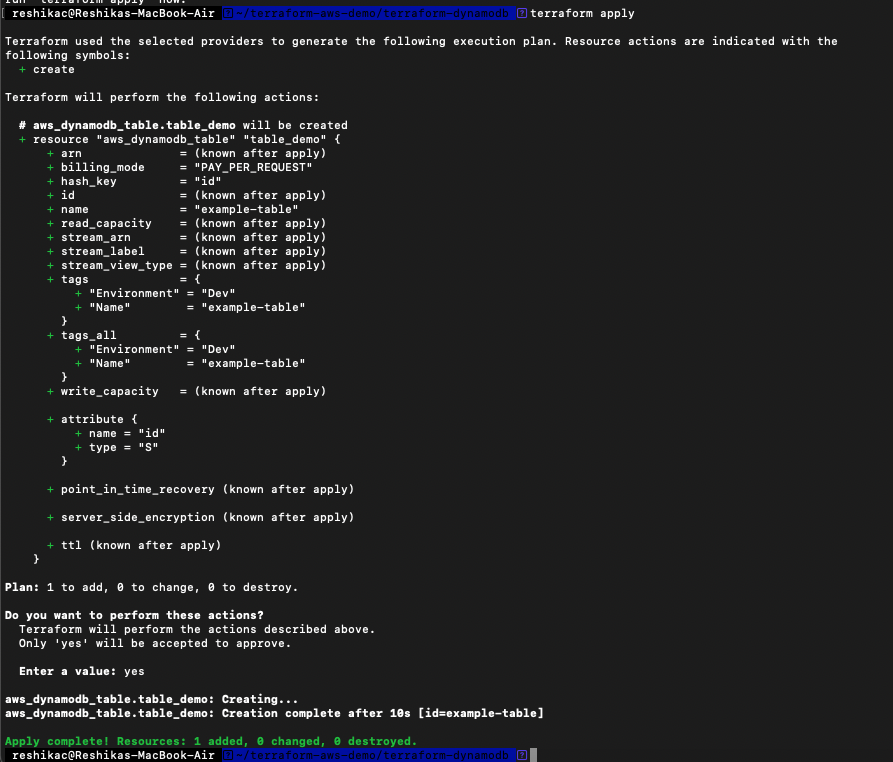
Step 8: Apply the Configuration

Apply the Terraform configuration to create the DynamoDB table.

```bash

terraform apply

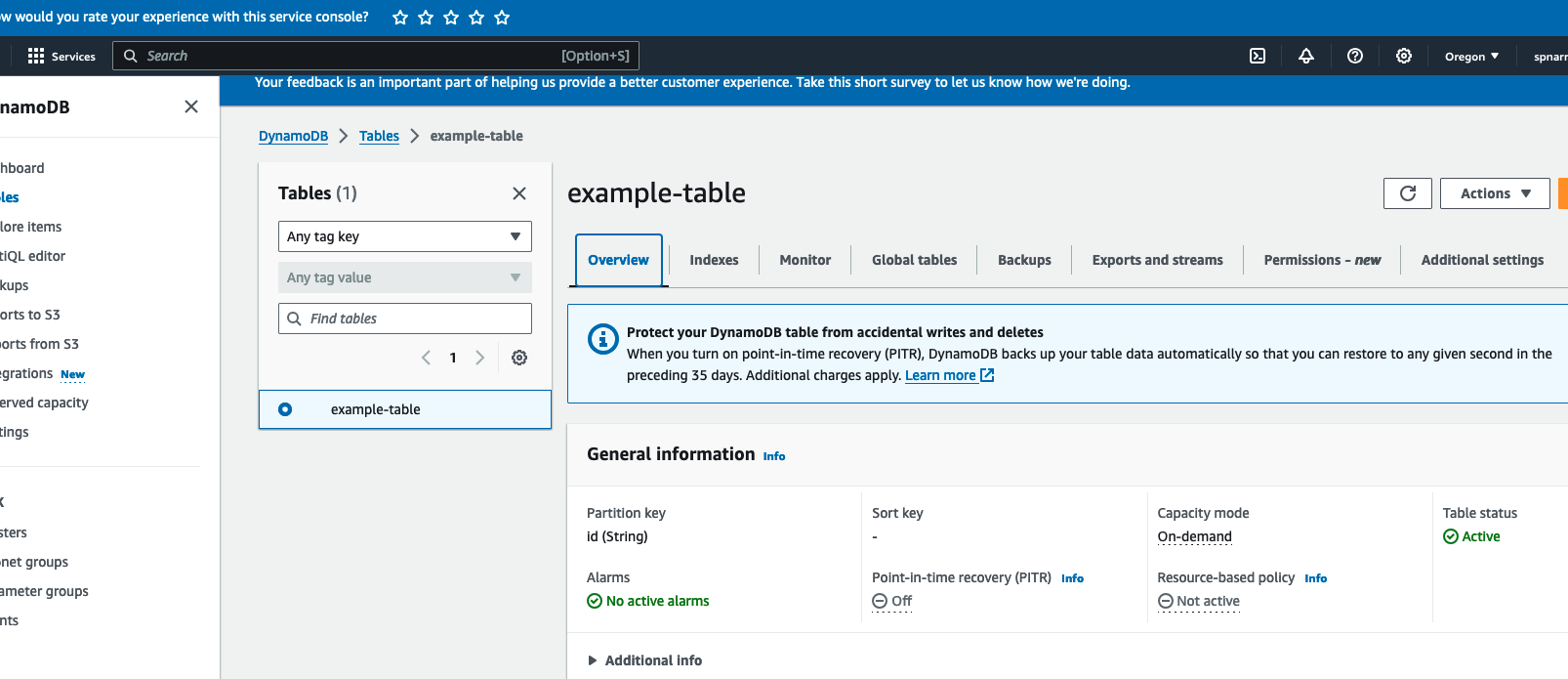
```



You'll be prompted to confirm. Type `yes` to proceed.

Step 9: Verify the Table

You can verify that the table was created by checking the AWS Management Console under DynamoDB or using the AWS CLI:



```bash

aws dynamodb list-tables --region us-west-2

```

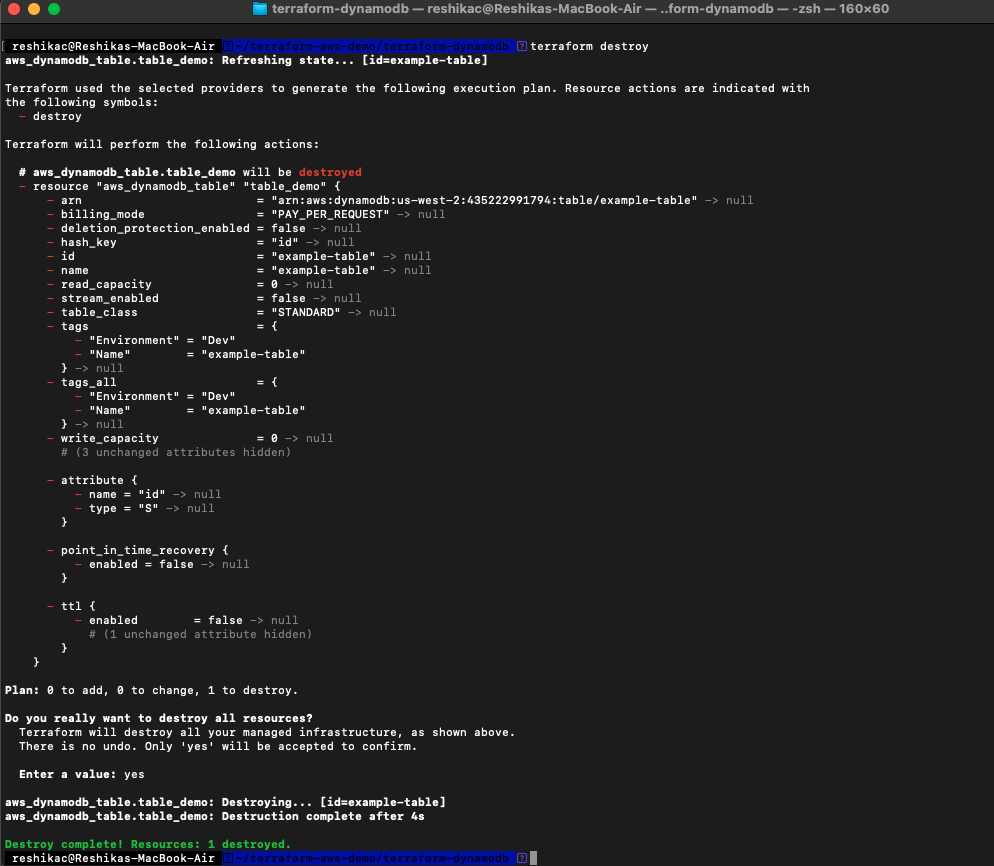
Step 10: Clean Up

If you want to remove the resources created by Terraform, use the `destroy` command:

```bash

terraform destroy

```



Confirm the destruction by typing `yes`.

Additional Configurations

You can further customize your DynamoDB table by adding secondary indexes, configuring stream settings, and more. Here’s an example to add a Global Secondary Index (GSI):

```hcl

resource "aws\_dynamodb\_table" "example" {

name = "example-table"

billing\_mode = "PAY\_PER\_REQUEST"

hash\_key = "id"

attribute {

name = "id"

type = "S"

}

attribute {

name = "createdAt"

type = "S"

}

global\_secondary\_index {

name = "createdAt-index"

hash\_key = "createdAt"

projection\_type = "ALL"

}

tags = {

Name = "example-table"

Environment = "dev"

}

}

```

This configuration adds a GSI named `createdAt-index` with `createdAt` as the hash key.

Conclusion

This tutorial covered the basics of setting up a DynamoDB table using Terraform. You can expand on this by integrating more AWS services or adding more complex configurations to your Terraform setup.