### 

[Installing Terraform](#_jia3mb9i2efv)

[What is Terraform?](#_a1ob1jfelqhq)

[Terraform Recap](#_16zjsbwf58kj)

[Activity 1 - Terraform Import](#_bojz0ndzm7xs)

[Terraform Walkthrough](#_pa6djdisyk8s)

[provider.tf](#_d6cf15ltofdt)

[main.tf](#_2myvhmpe6phu)

[Activity 2 - Hosting a Static Website on Cloudfront and S3](#_5xny6h9wyf5i)

[Terraform Walkthrough](#_g83tty4ul3bw)

[data.tf](#_8o57n4x0gxk7)

[providers.tf](#_vusjtld5853o)

[variables.tf](#_395fq2vie68c)

[main.tf](#_wc0yvqev72co)

[Activity 3 - Installing Git(Prep for Module 3)](#_y1quywqku9vl)

### 

## Installing Terraform

<https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli>

## What is Terraform?

Terraform is an open-source infrastructure as code (IaC) tool developed by HashiCorp. It allows you to define, provision, and manage infrastructure resources such as virtual machines, networks, storage, and other cloud-based services in a declarative way. Terraform uses a configuration language called HashiCorp Configuration Language (HCL) or JSON to describe the desired state of your infrastructure.

With Terraform, you can define your infrastructure as code, version it, and apply changes in a consistent and repeatable way. Terraform supports multiple cloud platforms such as Amazon Web Services, Microsoft Azure, Google Cloud Platform, and many others, making it possible to manage infrastructure across multiple clouds and on-premises environments.

Look at this link to see what kind of online resources you can build with Terraform:

<https://registry.terraform.io/browse/providers>

## Terraform Recap

We need to run terraform in the following steps:

1. **Initialize** - We need to initialize our Terraform environment by running the terraform init command. This allows you to download the necessary providers and modules that you have specified in your configuration.
2. **Plan** - After initializing the Terraform environment, we can create an execution plan with the terraform plan command. This plan shows us what changes Terraform will make to our infrastructure.
3. **Apply** - After we have reviewed the execution plan from the previous step and are satisfied with the proposed changes, we can apply the changes proposed by running the terraform apply command. Terraform will make the necessary changes to our infrastructure based on the plan.
4. Destroy - When we don’t need our cloud resources anymore, we can use the terraform destroy command to tear down all the resources created by Terraform.

[Basic CLI Features | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/cli/commands)

## Activity 1 - Terraform Import

Terraform can import existing infrastructure resources. This functionality lets you bring existing resources under Terraform management. Before you run terraform import you must manually write a resource configuration block for the resource. The resource block describes where Terraform should map the imported object.

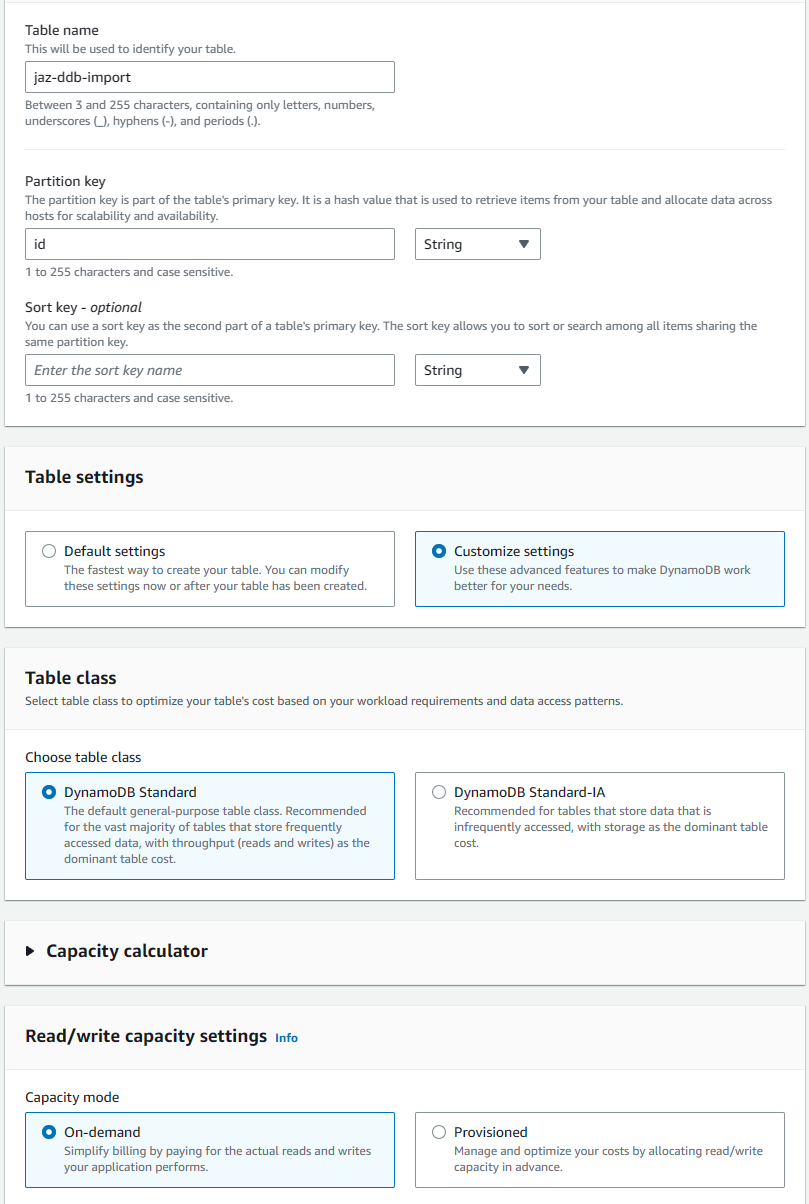
Create a DynamoDB table on the console with the following config:

Table name: <your table name>

Partition key: Id

Table class: DynamoDB standard

Capacity mode: on-demand



### Terraform Walkthrough

1. Create a new folder and create the following files in it:
   1. main.tf
   2. providers.tf

#### provider.tf

provider "aws" {

region = "ap-southeast-1"

}

#### main.tf

resource "aws\_dynamodb\_table" "personal\_table" {

name = "jaz-ddb-import" #Use your own table name here

billing\_mode = "PAY\_PER\_REQUEST"

hash\_key = "id"

attribute {

name = "id"

type = "S"

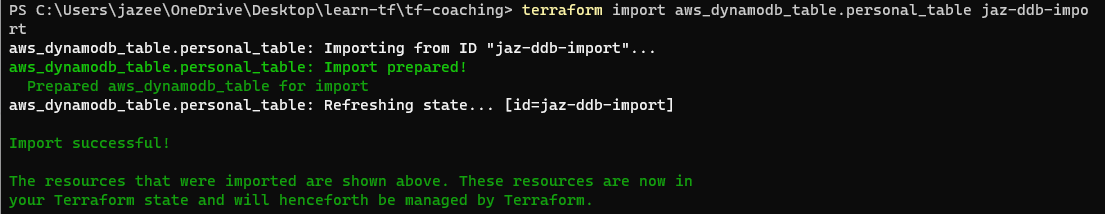
}

}

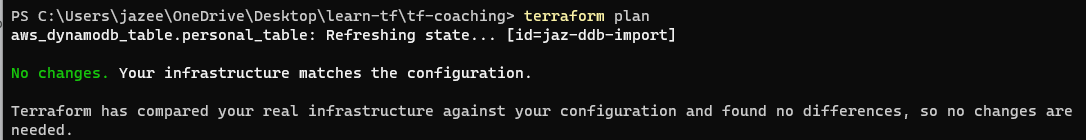
Run the following commands:

terraform init

terraform import aws\_dynamodb\_table.personal\_table jaz-ddb-import(this one replace with your table name)



terraform plan



## 

## Activity 2 - Hosting a Static Website on Cloudfront and S3

S3 bucket by default comes with the functionality of static web hosting. However integrating it with Cloudfront would also enable us to capitalize on the benefits of Edge Location, Web Application Firewall, Caching and AWS Certificate Manager.

In this Activity, We would be creating a s3 bucket to store our static web files and front it with a cloudfront distribution.Moreover, we would also add a bucket policy to ensure that connections to the s3 bucket can only be initiated through cloudfront. Meaning people can't access the s3 bucket directly(bypassing the cloudfront).

Ways to do it: [Use CloudFront to serve a static website hosted on Amazon S3 | AWS re:Post (repost.aws)](https://repost.aws/knowledge-center/cloudfront-serve-static-website)

### Terraform Walkthrough

1. Create a new folder and create the following files in it:
   1. data.tf
   2. main.tf
   3. providers.tf
   4. variables.tf

#### 

#### 

#### data.tf

data "aws\_cloudfront\_cache\_policy" "example" {

name = "Managed-CachingOptimized"

}

data "aws\_iam\_policy\_document" "default" {

statement {

actions = ["s3:GetObject"]

resources = ["${aws\_s3\_bucket.static\_web.arn}/\*"]

principals {

type = "Service"

identifiers = ["cloudfront.amazonaws.com"]

}

condition {

test = "StringEquals"

variable = "AWS:SourceArn"

values = [aws\_cloudfront\_distribution.s3\_distribution.arn]

}

}

}

#### providers.tf

provider "aws" {

region = "ap-southeast-1"

}

#### 

#### variables.tf

variable "bucket\_name" {

type = string

default = "jaz-cfstatic-1606-bkt" #Set your bucket name here that you want to create

}

variable "allowed\_methods" {

type = list(string)

default = ["GET", "HEAD"]

}

variable "cached\_methods" {

type = list(string)

default = ["GET", "HEAD"]

}

variable "viewer\_protocol\_policy" {

type = string

default = "allow-all"

}

#### main.tf

locals {

origin\_id = "s3origin"

}

resource "aws\_s3\_bucket" "static\_web" {

bucket = var.bucket\_name

}

resource "aws\_s3\_bucket\_policy" "allow\_access\_from\_cloudfront" {

bucket = aws\_s3\_bucket.static\_web.id

policy = data.aws\_iam\_policy\_document.default.json

}

resource "aws\_cloudfront\_distribution" "s3\_distribution" {

origin {

domain\_name = aws\_s3\_bucket.static\_web.bucket\_regional\_domain\_name

origin\_access\_control\_id = aws\_cloudfront\_origin\_access\_control.oac.id

origin\_id = local.origin\_id

}

enabled = true

comment = "Static Website using S3 and Cloudfront OAC"

default\_root\_object = "index.html"

default\_cache\_behavior {

cache\_policy\_id = data.aws\_cloudfront\_cache\_policy.example.id

allowed\_methods = var.allowed\_methods

cached\_methods = var.cached\_methods

target\_origin\_id = local.origin\_id

viewer\_protocol\_policy = var.viewer\_protocol\_policy

}

viewer\_certificate {

cloudfront\_default\_certificate = true

}

restrictions {

geo\_restriction {

restriction\_type = "none"

locations = []

}

}

}

resource "aws\_cloudfront\_origin\_access\_control" "oac" {

name = "${aws\_s3\_bucket.static\_web.id}-oac"

origin\_access\_control\_origin\_type = "s3"

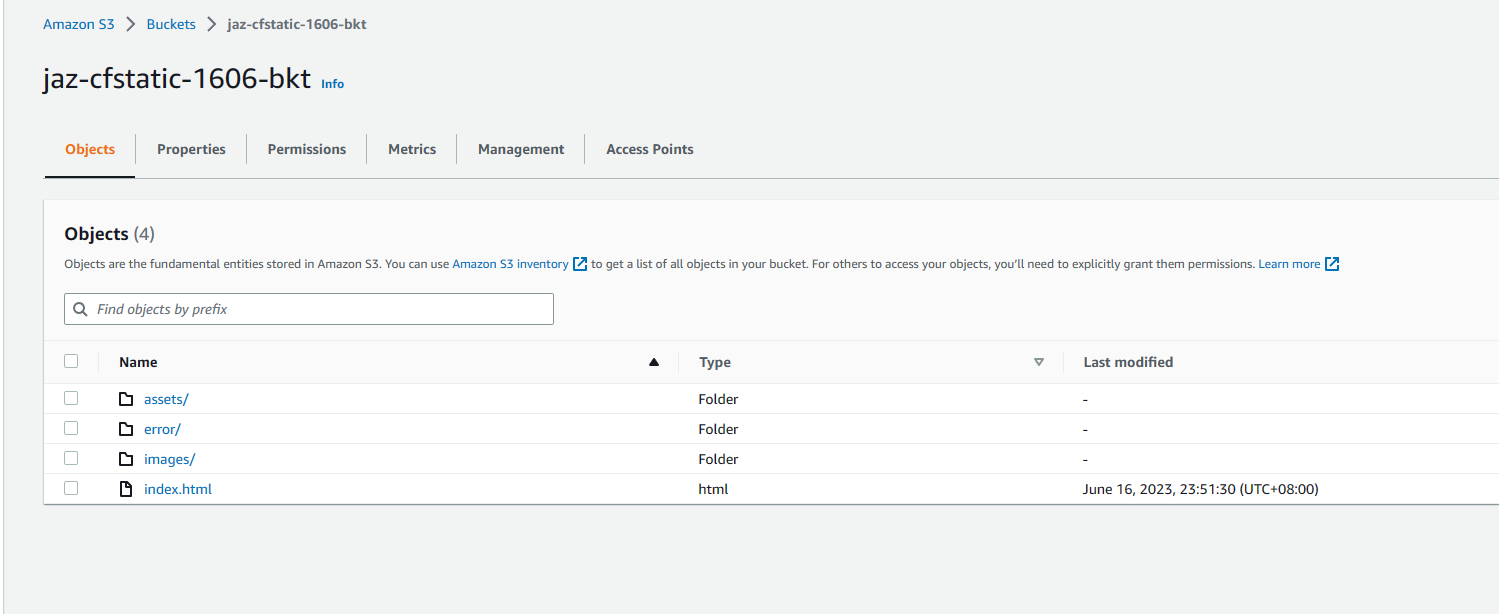
signing\_behavior = "always"

signing\_protocol = "sigv4"

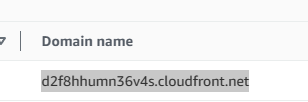
}

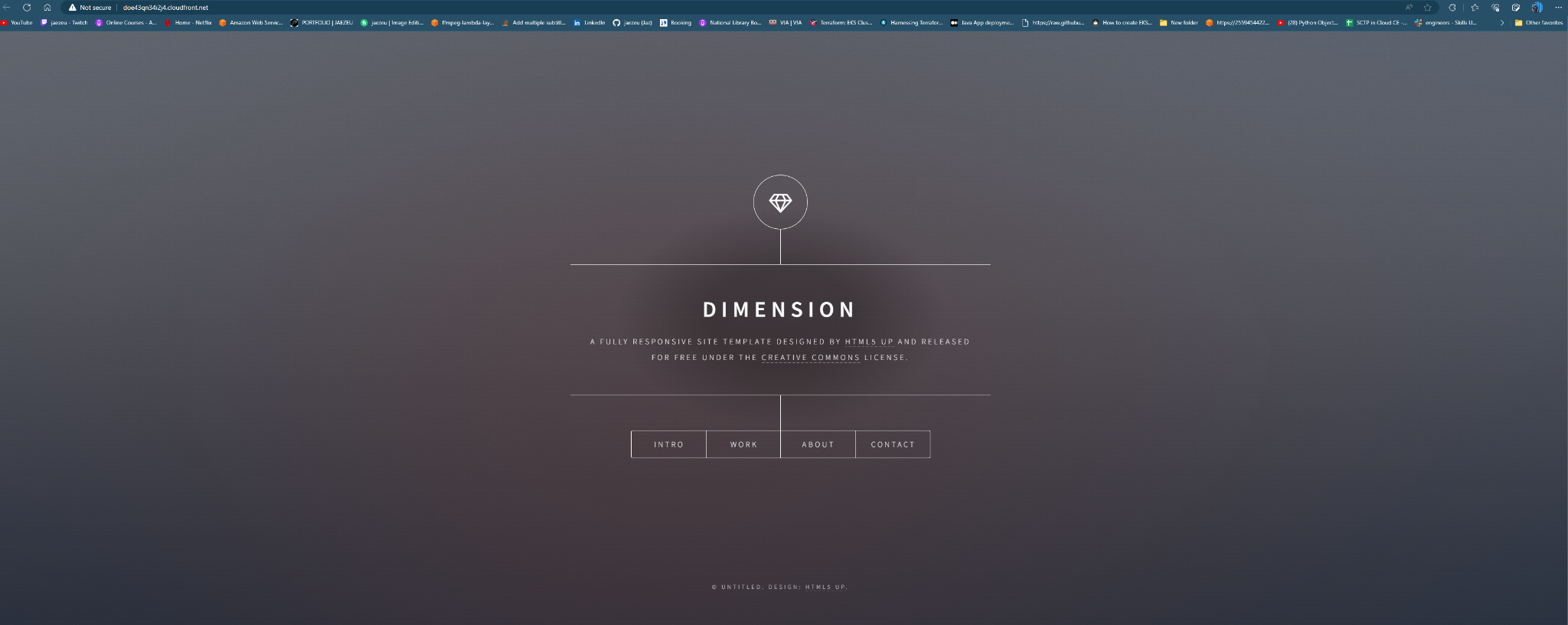
1. Go to the s3 bucket you created and upload the static web files into it:

Static web files: [jaezeu/cloudfront-static-web (github.com)](https://github.com/jaezeu/cloudfront-static-web/tree/main/static-website)



1. Go to the cloudfront distribution and copy the domain name and paste it in a browser:





## Activity 3 - Installing Git(Prep for Module 3)

[Git - Installing Git (git-scm.com)](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)

Mac: <https://git-scm.com/download/mac>

Windows: <https://git-scm.com/download/win>

Git verification in terminal/ powershell:

