

Práctica final de Terraform y AWS

Claudia Arrate Esteve

13 de noviembre de 2025

Resumen

En el siguiente documento se muestra paso por paso como se han realizado los dos ejercicios correspondientes a la Práctica final de Terraform y AWS del *II Talent Campus en Automatización + IA* de la empresa *PSS* impartido por Joel Rosental. Dicha práctica se divide en un **Parte II** y una **Parte III**.

1 Parte II

El siguiente apartado presenta los pasos de ejecución implementados para el despliegue de un sitio web estático sobre la infraestructura de Amazon Web Services (AWS) S3. El objetivo principal fue establecer una solución de hosting robusta y completamente automatizada. Para ello se usarán cinco pilares fundamentales: la creación del bucket, la gestión de permisos públicos, la aplicación de políticas de acceso, la configuración del hosting estático y la carga de los objetos.

1.1 Aprovisionamiento de Infraestructura con Terraform

Antes de empezar a crear el bucket se crearan manualmente los archivos *index.html* y *error.html* que se piden en los pasos 2 y 3. Estos archivos contienen un pequeño texto para indicarlo. El primero avisa de que estás en Terraform y el segundo muestra que la página no ha sido encontrada. Como último prerequisite añadiremos el documento *provider.tf*.

1.1.1 Creación y Configuración del Bucket S3

Como primer paso (pasos 1 y 4 de la hoja de enunciado) se crea el recurso principal, el bucket S3, el cual debe poseer un nombre único a nivel global. Adicionalmente, y de manera crítica para la exposición pública del sitio, se activan explícitamente todas las reglas dentro del **Public Access Block** de AWS, conforme a la normativa de acceso. El código que se muestra a continuación irá en el archivo *main.tf*.

Listing 1: Creación del Bucket y Desactivación del Bloqueo Público

```
1 # 1. Creacion del Bucket S3
2 resource "aws_s3_bucket" "mi_bucket_ejemplo" {
3   bucket = "claudia-ae-pss" #los nombres de buckets no aceptan
4                               mayusculas
5
6   tags = {
7     Name           = "MiBucketDesdeTerraform"
8     Environment    = "Dev"
9   }
10
11 # 2. Activacion del Public Access Block (Punto 4)
12 resource "aws_s3_bucket_public_access_block" "public_access_block" {
13   bucket = aws_s3_bucket.website_bucket.id
```

```

14   block_public_acls      = true
15   block_public_policy    = true
16   ignore_public_acls     = true
17   restrict_public_buckets = true
18 }
19
20 # Habilita el control de versiones (para poder recuperar objetos
   eliminados/modificados)
21 resource "aws_s3_bucket_versioning" "versioning_config" {
22   bucket = aws_s3_bucket.mi_bucket_ejemplo.id
23   versioning_configuration {
24     status = "Enabled"
25   }
26 }

```

1.1.2 Gestión de Políticas de Acceso (Punto 5)

Para garantizar la accesibilidad al contenido web por parte de cualquier usuario a través de Internet, se implementa una `aws_s3_bucket_policy`. Esta política otorga el permiso `s3:GetObject` (lectura) a todos los objetos dentro del bucket. El siguiente código se añadirá al archivo `main.tf`

Listing 2: Política de Bucket para Acceso Público de Lectura

```

1 data "aws_iam_policy_document" "website_policy" {
2   statement {
3     principals {
4       type       = "AWS"
5       identifiers = ["*"]
6     }
7     actions = ["s3:GetObject"]
8     resources = [
9       aws_s3_bucket.website_bucket.arn,
10      "${aws_s3_bucket.website_bucket.arn}/*",
11    ]
12  }
13 }
14 #Aplicar la politica al bucket
15 resource "aws_s3_bucket_policy" "website_policy" {
16   bucket      = aws_s3_bucket.website_bucket.id
17   depends_on = [aws_s3_bucket_public_access_block.public_access_block]
18   policy      = data.aws_iam_policy_document.website_policy.json
19 }

```

Además para que esto funcione es necesario realizar los siguientes cambios en el código que se tenía anteriormente.

Listing 3: Modificación de código

```

1 # 2. Activacion del Public Access Block
2 resource "aws_s3_bucket_public_access_block" "public_access_block" {
3   bucket = aws_s3_bucket.mi_bucket_ejemplo.id
4   block_public_acls      = true
5   block_public_policy    = false #<--
6   ignore_public_acls     = true
7   restrict_public_buckets = false #<--
8 }

```

1.1.3 Configuración del Hosting Estático (Punto 6)

Se habilita la funcionalidad de sitio web estático de S3. Para ello se hace uso del recurso `aws_s3_bucket_website_configuration`, definiendo `index.html` y `error.html` como documentos de índice y error, respectivamente.

Listing 4: Configuración del Hosting de Sitio Web Estático

```

1 resource "aws_s3_bucket_website_configuration" "website_config" {
2   bucket = aws_s3_bucket.mi_bucket_ejemplo.id
3   index_document {
4     suffix = "index.html"
5   }
6   error_document {
7     key = "error.html"
8   }
9 }

```

Además hay que crear el archivo `outputs.tf` con el siguiente contenido (punto 7).

```

1 #Parte 7
2 output "website_endpoint" {
3   description = "El endpoint p blico del sitio web est tico S3."
4   value       = aws_s3_bucket_website_configuration.website_config.
5     website_endpoint
6 }
7 output "website_url" {
8   description = "La URL completa del sitio web est tico S3 (usando la
9     regi n)."
10  value       = aws_s3_bucket.mi_bucket_ejemplo.website_endpoint
11 }

```

1.2 Gestión del Contenido y Pruebas

1.2.1 Carga de Archivos Mediante Terraform (Punto 8)

Inicialmente, se realizó una carga manual de los archivos `index.html` y `error.html` para la verificación funcional. Ahora se procede a la eliminación de los archivos manuales y a la utilización del recurso `aws_s3_object` para la carga programática.

Listing 5: Carga de Objetos de Contenido al Bucket

```

1 # Carga de index.html
2 resource "aws_s3_object" "index" {
3   bucket      = aws_s3_bucket.mi_bucket_ejemplo.id
4   key         = "index.html"
5   source      = "index.html" (Archivo local)
6   content_type = "text/html"
7   etag        = filemd5("index.html") #para mantener cambios locales
8 }
9
10 # Carga de error.html
11 resource "aws_s3_object" "error" {
12   bucket      = aws_s3_bucket.mi_bucket_ejemplo.id
13   key         = "error.html"
14   source      = "error.html"
15   content_type = "text/html"
16   etag        = filemd5("error.html") #para mantener cambios locales
17 }

```

1.2.2 Depurado de código (parte 9)

Para que el código sea más legible se añaden *tags* a los recursos para identificar de forma sencilla que se está haciendo en cada parte. Por ejemplo en la parte 8 añadimos a los recursos respectivamente:

```
1  tags = {  
2    Project      = "StaticWebsite"  
3    ManagedBy    = "Terraform"  
4    ContentRole  = "FilePage"  
5  }  
6  
7  tags = {  
8    Project      = "StaticWebsite"  
9    ManagedBy    = "Terraform"  
10   ContentRole  = "ErrorPage"  
11  }
```

1.3 Eliminación de recursos (Punto 10)

Finalmente, para cumplir con el requisito de gestión de costos, se recomienda la ejecución de la función `terraform destroy` al concluir las pruebas, garantizando la eliminación de todos los recursos aprovisionados.