Create a Static S3 Website Using Terraform

Introduction

- In this workshop we'll be learning about HashiCorp Terraform, an Infrastructure as Code (IaC) language that enables us to build and maintain infrastructure via code. We'll start by setting up our development environment and then build a static website hosted on an AWS S3 bucket.

- If you don’t want to use your own AWS account, check out this free lab environment from Cybr!

- https://cybr.com/courses/securing-amazon-s3/lessons/create-a-static-s3-website-using-terraform/

What is Infrastructure as Code (IaC)?

- Infrastructure as Code (IaC) enables creating and managing infrastructure like virtual machines, networks, and more using code. So, for example, instead of going into the AWS console and manually creating dozens of resources, you can write code to do this.

- By using IaC, your infrastructure now goes through the same software development lifecycle as any application code should. This means you can version control your infrastructure, automate deployments and updates, identify and remediate security issues prior to deployment, and more.

- HashiCorp's Terraform is arguably the most popular IaC tool, is open-source, and supports thousands of industry solutions like AWS, Azure, Kubernetes, Proxmox, Okta, and more. Several other IaC solutions do exist however including AWS CloudFormation, Pulumi, and OpenTofu.

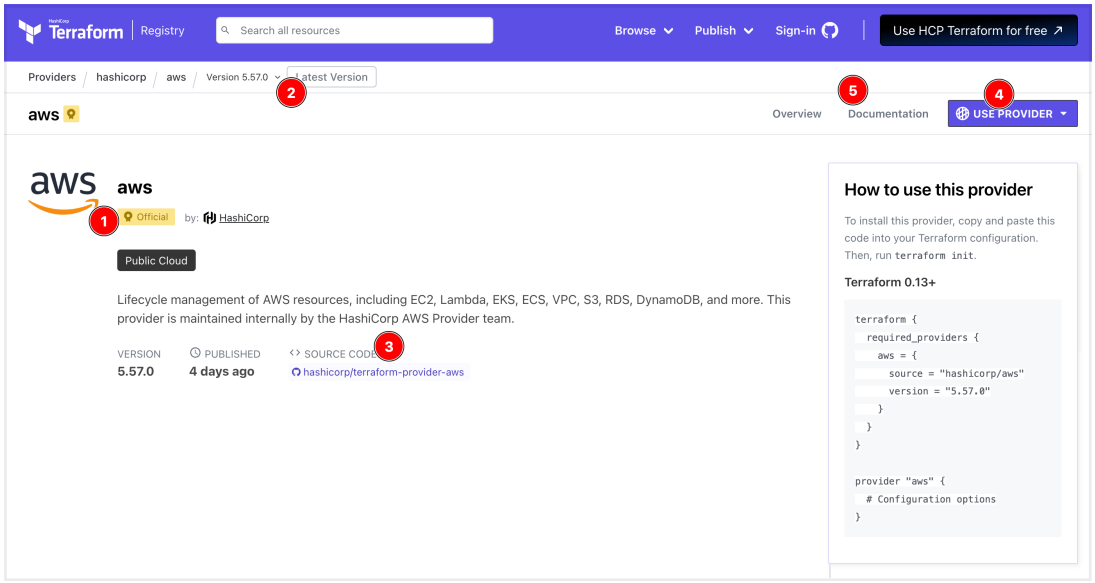
Terraform Providers

- To build infrastructure on AWS or other solutions, Terraform relies on Providers. These are essentially plugins that translate Terraform code into API calls for a service. So all you have to worry about is writing Terraform code and the provider will handle the rest.

- Let's take a look at the Providers available here and the differences

between the tiers (Offical, Partner, and Community). The key thing to remember is that *anyone* can create a provider so it's important that you trust the provider you're using.

- Looking at the AWS provider, we can see a few key things.



1. **Official**
    - This is the official AWS provider maintained by HashiCorp. It's the most trusted and most up-to-date provider.
2. **Version**
    - This is the version of the provider. It's important to keep your provider up-to-date to ensure you have the latest features and bug fixes.
3. **Source Code**
    - This is the link to the source code of the provider. This is useful if you want to see how the provider works or if you want to contribute to the provider.
4. **Provider**
    - Provides sample code for using the provider in your Terraform code.
5. **Documentation**
    - Provides documentation on the various AWS resources that can be created.

Installing the tools

Terraform

- See the official HashiCorp documentation for additional options.

AWS

- See the official AWS documentation for additional options.

Configuring the AWS CLI

- With your AWS access keys in hand, run the below command to configure

```
aws configure

    AWS Access Key ID [None]: <accessKeyId>
    AWS Secret Access Key [None]: <secretAccessKeyId>
    Default region name [None]: us-east-1
    Default output format [None]: json
```

- Confirm access. You should see a similar result below

```
aws sts get-caller-identity

    {
        "UserId": "AIDA7DH3JSLDF0NDND35RB",
        "Account": "123456789012",
        "Arn": "arn:aws:iam::123456789012:user/<userName>"
    }
```

Step 1: Creating a directory and file

- Make sure you have a clean directory to start in and create a `main.tf` file

```
mkdir terraform_lab

touch main.tf
```

Step 2: Define the Provider

- By referring to the documentation, we can learn how to use and add the AWS terraform provider to our terraform code.

- Add the below code into your `main.tf` file.

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

- Make sure to use us-east-1 if you're following along in the cyber lab

- Now we're going to initialize our provider which will download it to our local directory

```
terraform init

    Initializing the backend...

    Initializing provider plugins...
    - Finding latest version of hashicorp/aws...
    - Installing hashicorp/aws v5.60.0...
    - Installed hashicorp/aws v5.60.0 (signed by HashiCorp)
    [snip]
```

- Next we can view the hidden provider

```
ls -alh

    .terraform
    .terraform.lock.hcl
    main.tf
```

- This command will show us the version of the provider (5.60.0) although your version may be different

```
tree .terraform (OR) ls -R .terraform

    .terraform/providers/registry.terraform.io/hashicorp/aws:
5.60.0
```

Step 3: Creating the bucket

- AWS S3 is a simple resource that enables storing large amounts of data in a cost-effective manner. By default, S3 buckets are private but can be made public if needed. We'll start by creating a private bucket and then move on to making it public and hosting a static website.

- Copy the below code into your `main.tf` file and let's reference the documentation for the AWS S3 bucket resource here.

```
resource "aws_s3_bucket" "bucket1" {
  bucket = "<my-unique-bucket-name>"
}
```

- Make sure to replace `my-unique-bucket-name` with your own. It must be a bucket name that no one else in the world is using.

- After saving the file, let's learn how to validate our code's syntax

```
terraform validate

    Success! The configuration is valid.
```

- We should have no errors but as an example I'll add a fake argument and run the command again

```
resource "aws_s3_bucket" "bucket1" {
  bucket = "tylers-website-9827349723"
  fake_argument = "fake_value"
}
```

- Save the change and run the validate command again to see an error message

```
Error: Unsupported argument
│
│   on main.tf line 29, in resource "aws_s3_bucket" "bucket1":
│   29:   fake_argument = "fake_value"
│
│ An argument named "fake_argument" is not expected here.
```

- Make sure to undo adding that fake argument before moving on.

- Another handy command is format which will ensure our code is spaced appropriately

- To show a good example I'm going to add another argument to my code and make the spacing funky

```
resource "aws_s3_bucket" "bucket1" {
  bucket     =  "tylers-website-982734972349111234234"
  tags =     {
        "name"      = "value"
    }
}
```

- See how the spacing is all messed up?
- If we run terraform fmt it will fix it

```
terraform fmt
```

```
resource "aws_s3_bucket" "bucket1" {
  bucket = "tylers-website-982734972349111234234"
  tags = {
    "name" = "value"
  }
}
```

- Next we'll run terraform plan
- This is essentially a dry run, terraform will look at the code and show you what it's going to do

```
terraform plan
```

```
    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.bucket1 will be created
      + resource "aws_s3_bucket" "bucket1" {
          + acceleration_status         = (known after apply)
          + acl                         = (known after apply)
          + arn                         = (known after apply)
          + bucket                      = "tylers-
website-982734972349111234234"
          + bucket_domain_name          = (known after apply)
```

```
          + bucket_prefix               = (known after apply)
          + bucket_regional_domain_name = (known after apply)
          + force_destroy               = false
          + hosted_zone_id              = (known after apply)
          + id                          = (known after apply)
          + object_lock_enabled         = (known after apply)
          + policy                      = (known after apply)
          + region                      = (known after apply)
          + request_payer               = (known after apply)
          + tags_all                    = (known after apply)
          + website_domain              = (known after apply)
          + website_endpoint            = (known after apply)
        }

    Plan: 1 to add, 0 to change, 0 to destroy.
```

- When we're satisfied with that we'll run an apply which will actually deploy our code

```
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.bucket1 will be created
      + resource "aws_s3_bucket" "bucket1" {
          + acceleration_status         = (known after apply)
          + acl                         = (known after apply)
          + arn                         = (known after apply)
          + bucket                      = "tylers-
website-982734972349111234234"
          + bucket_domain_name          = (known after apply)
          + bucket_prefix               = (known after apply)
          + bucket_regional_domain_name = (known after apply)
          + force_destroy               = false
          + hosted_zone_id              = (known after apply)
          + id                          = (known after apply)
          + object_lock_enabled         = (known after apply)
          + policy                      = (known after apply)
```

```
          + region                     = (known after apply)
          + request_payer              = (known after apply)
          + tags_all                   = (known after apply)
          + website_domain             = (known after apply)
          + website_endpoint           = (known after apply)
        }

    Plan: 1 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.bucket: Creating...
    aws_s3_bucket.bucket: Creation complete after 2s
[id=tylers-website-982734972349111234234]
```

Step 4: Validating our bucket

- After it's created, we can use an AWS CLI command to view our bucket

```
aws s3 ls

    2024-07-30 22:07:45 tylers-website-982734972349111234234
```

Step 5: Terraform State

- One important thing about terraform is that it keeps track of resources it manages and stores this into a file called `terraform.tfstate`.

- We can view the state like so

```
terraform state list

    aws_s3_bucket.bucket1
```

- And view the properties of our bucket with

```
terraform state show aws_s3_bucket.bucket1

    # aws_s3_bucket.bucket1:
    resource "aws_s3_bucket" "bucket1" {
        arn                         = "arn:aws:s3:::tylers-
website-982734972349111234234"
        bucket                      = "tylers-
website-982734972349111234234"
        bucket_domain_name          = "tylers-
website-982734972349111234234.s3.amazonaws.com"
        bucket_regional_domain_name = "tylers-
website-982734972349111234234.s3.us-east-1.amazonaws.com"
        force_destroy               = false
        hosted_zone_id              = "Z3AQBSTGFYJSTF"
        id                          = "tylers-
website-982734972349111234234"
        object_lock_enabled         = false
        region                      = "us-east-1"
        request_payer               = "BucketOwner"
    [snip]
```

- If we list our directory we'll see this file as well

```
ls

    terraform.tfstate
```

- Since`terraform.tfstate` maintains the state of our resources we want to ensure it stays protected and we don't mess with it for the purposes of our lab.

- Refer to the official HashiCorp docs if you want to learn more about terraform state.

Step 6: Enable Public Access

- The next step is to enable public access for our bucket.

- We can check out the terraform documentation to learn how to do this.

- Let's add this new resource and set the arguments to false

```
resource "aws_s3_bucket_public_access_block"
"public_access_block" {
  bucket = aws_s3_bucket.bucket1.id

  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}
```

- Additionally, since we're accessing the bucket from our web browser, we're an anonymous user so we don't have access to it

- Because of this, we need to create a bucket policy that gives everyone read access

- There's a few different ways to define the policy but we'll do it this way and again there's documentation for this as well

```
resource "aws_s3_bucket_policy" "bucket_policy" {
  bucket = aws_s3_bucket.bucket1.id
  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect    = "Allow",
        Principal = "*",
        Action    = "s3:GetObject",
        Resource  = "${aws_s3_bucket.bucket1.arn}/*",
      },
    ],
  })
}
```

- Let's run our terraform commands again and deploy the changes

```
terraform validate

terraform fmt

terraform plan

terraform apply
```

Step 7: Configure our website

- In this section, we're going to create a simple HTML file, upload it to our bucket, and configure our bucket to be a website.

- We'll want to save the below HTML code into a file called `index.html` in the same directory as our terraform code.

```
<!DOCTYPE html>
<html>

<head>
    <title>Tyler's S3 Website</title>
</head>

<body>
    <h1>Tyler's S3 Website</h1>
</body>

</html>
```

- Next we're going to create a few more resources

- This resource is going to be the one that uploads our html file to the bucket

```
resource "aws_s3_object" "index" {
  bucket       = aws_s3_bucket.bucket1.id
  key          = "index.html"
  source       = "index.html"
  content_type = "text/html"
  source_hash  = filemd5("index.html")
}
```

- This one will configure the bucket to be a website

```
resource "aws_s3_bucket_website_configuration" "website_config"
{
  bucket = aws_s3_bucket.bucket1.id
  index_document {
    suffix = "index.html"
  }
```

```
}
```

- This resource will output the website endpoint we'll need to view in our browser

```
output "website_url" {
    value =
aws_s3_bucket_website_configuration.website_config.website_endp
oint
}
```

- Let's run our terraform commands again and make sure everything is good to go

```
terraform validate

terraform fmt

terraform plan

terraform apply
```

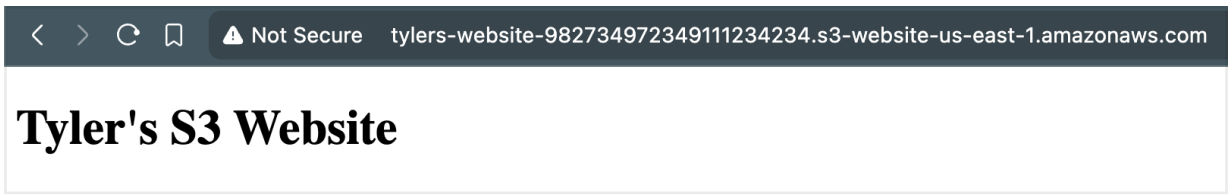
- When done, we should get the website endpoint as output

```
Outputs:

website_url = "tylers-website-982734972349111234234.s3-website-
us-east-1.amazonaws.com"
```

Step 8: Viewing the website

- Navigate to the website URL outputted by Terraform to see your website.



Step 9: Destroying the Infrastructure

- When you're done with your infrastructure, you can destroy it using terraform destroy. It's important to do this to avoid unnecessary

costs. For an S3 bucket, you pay for the storage and data transfer costs. See the pricing here for more information.

```
terraform destroy

    [snip]
    Plan: 0 to add, 0 to change, 5 to destroy.

    Changes to Outputs:
      - website_url = "tylers-website-982734972349111234234.s3-
website-us-east-1.amazonaws.com" -> null
    aws_s3_bucket_policy.bucket_policy: Destroying...
[id=tylers-website-982734972349111234234]
    aws_s3_bucket_public_access_block.public_access_block:
Destroying... [id=tylers-website-982734972349111234234]
    aws_s3_bucket_website_configuration.website_config:
Destroying... [id=tylers-website-982734972349111234234]
    aws_s3_object.index: Destroying... [id=index.html]
    aws_s3_object.index: Destruction complete after 1s
    aws_s3_bucket_website_configuration.website_config:
Destruction complete after 1s
    aws_s3_bucket_policy.bucket_policy: Destruction complete
after 1s
    aws_s3_bucket_public_access_block.public_access_block:
Destruction complete after 1s
    aws_s3_bucket.bucket1: Destroying... [id=tylers-
website-982734972349111234234]
    aws_s3_bucket.bucket1: Destruction complete after 1s

    Destroy complete! Resources: 5 destroyed.
```

Bonus

1. Update your website
    - This can be done by modifying the index file
    - Re-run terraform apply and your website will update
    - This is because of the hash function we configured in the resource

```
source_hash  = filemd5("index.html")
```

2. Generate and resolve drift
    - Sometimes resources can be modified outside of terraform e.g., via

the AWS console or CLI.
    - When this occurs, the next time terraform runs, it will detect the change and want to revert it.
    - Run the below command to make the S3 bucket private,

```
aws s3api put-public-access-block --bucket <your-bucket-name>
--public-access-block-configuration
"BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=t
rue,RestrictPublicBuckets=true"
```

- Now re-run terraform plan and we'll see this,

```
terraform plan

    [snip]
    Resource actions are indicated with the following symbols:
      ~ update in-place

    Terraform will perform the following actions:

      # aws_s3_bucket_public_access_block.public_access_block
will be updated in-place
      ~ resource "aws_s3_bucket_public_access_block"
"public_access_block" {
          ~ block_public_acls       = true -> false
          ~ block_public_policy     = true -> false
            id                      = "tylers-
website-982734972349111234234"
          ~ ignore_public_acls      = true -> false
          ~ restrict_public_buckets = true -> false
            # (1 unchanged attribute hidden)
        }

    Plan: 0 to add, 1 to change, 0 to destroy.
```

- Run terraform apply again to fix the drift

```
terraform apply

    [snip]
    Terraform used the selected providers to generate the
following execution plan. Resource actions are indicated with
the following symbols:
      ~ update in-place
```

```
    Terraform will perform the following actions:

      # aws_s3_bucket_public_access_block.public_access_block
will be updated in-place
      ~ resource "aws_s3_bucket_public_access_block"
"public_access_block" {
          ~ block_public_acls       = true -> false
          ~ block_public_policy     = true -> false
            id                      = "tylers-
website-982734972349111234234"
          ~ ignore_public_acls      = true -> false
          ~ restrict_public_buckets = true -> false
            # (1 unchanged attribute hidden)
        }

    Plan: 0 to add, 1 to change, 0 to destroy.
    aws_s3_bucket_public_access_block.public_access_block:
Modifying... [id=tylers-website-982734972349111234234]
    aws_s3_bucket_public_access_block.public_access_block:
Modifications complete after 1s [id=tylers-
website-982734972349111234234]

    Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

Improving the architecture

- S3 websites support custom domain names, see this AWS doc for more details
- CloudFront can be used to improve website performance and enable HTTPS to encrypt the website session, see this AWS doc for more details
- If we want users to access our website both from its domain name and subdomain name, e.g., mywebsite.com or www.mywebsite.com we'll need to create two S3 buckets, see this AWS doc for more details