

Xerris Bootcamp Series

Terraform Workshop



ACCOLITE DIGITAL
Transforming The Future, Now

Terraform 101

Simple AWS – S3 example

- Create a new folder called /terraform/s3.
- Create a file called main.tf
- main.tf acts as the **modules** entry point.
- Define the AWS Provider.
- Define your S3 Bucket.

Terraform Modules

- Contained within a folder
- It starts with main.tf
- Modules allow for component reuse across Terraform.



Terraform 101

Defining the required provider

- Each module needs to define the required module.
- The version number is subject to change as Terraform releases more versions.

```
terraform {  
  required_providers {  
    aws = {  
      source = "hashicorp/aws"  
      version = "~> 3.27"  
    }  
  }  
  required_version = ">= 0.14.9"  
}
```



Terraform 101

Define the S3 resource

- Ensure the bucket name is unique.
- Must be globally unique across all of AWS

```
resource "aws_s3_bucket" "my-bucket"  
{  
  bucket = "{globally unique bucket name}"  
  acl = "private"  
  tags = {  
    Name = "My bucket"  
    Environment = "Dev"  
  }  
}
```



Terraform 101

Deploying Your Infrastructure

terraform init

- Initializes your module by installing the appropriate cloud provider

terraform plan

- Looks at your 'current state' and create a plan to update to your 'desired state'.

terraform apply

- Applies your module to your profile.
- **terraform destroy**
- Tears down your 'stack' when you are done with it.



Terraform 101

Allows you to parameterize your terraform code

- `var.tf` – file to hold all your variables
 - `Terraform.tfvars` – a file that contains the values for your variables
 - Not usually checked into source control
 - Will be different between AWS environments (DEV, QA, PROD)
- `-var {var_name}={var_value}` – allows you to provide variables values to the terraform CLI.
- `-var-file {path to tfvar file}`

```
variable "site_bucket_name" {  
    type = string  
}
```

`terraform plan -var site_bucket_name=xerris-academy-static-site`



Terraform 101

Adding Bucket Policy

- Adding a bucket policy to allow for read-only access to our S3 static site bucket
- **IAM** – Identity Access Management
- How AWS secures resources.
- **Users** – defines users of AWS
- **Roles** – defines roles users can assume
- **Policies** – defines policies that can be attached to user or roles
- By default, All resources are denied access.



Securing your S3 Bucket

Creating a Policy

- Using the bucket_policy resource
- Define the action(s) you are going to allow: s3:GetObject
- Define the AWS resources to apply this policy to
- Define the principals to attach this policy.



Securing your S3 Bucket

Creating a Policy

```
data "aws_iam_policy_document" "bucket_policy" {
  statement {
    sid = "AllowReadFromAll"
    actions = [
      "s3:GetObject",
    ]
    resources = [
      "arn:aws:s3:::${var.site_bucket_name}/*",
    ]
    principals {
      type = "*"
      identifiers = ["*"]
    }
  }
}
```



Terraform TVARS file

Creating a Policy

- A **tfvar** file can provide values for all your variables
- Usually used when defining all variables for an environment (Dev, Stage, Prod)
- Could be checked into source control for CI/CD pipelines
- Should NEVER contain API keys, secrets or any other sensitive information



Terraform Backend State

Terraform tracks the state of your stack.

terraform.tfstate

- Contains the current state of your environment.
- Should not be stored within in source control
- Issues when in a multi-developer environment.

AWS S3 Backend

- *Terraform supports using AWS S3 as the state 'backend.'*

```
backend "s3" {  
    bucket = "xerris-academy-website-tfstate-greg"  
    key = "terraform.tfstate"  
    region = "us-west-2"  
}
```



Terraform Backend State

AWS S3 Backend

- *Create the S3 bucket for your state*
- *Grant permissions to your CI/CD account to this S3 bucket*
- *Use the **backend** resource as shown earlier*
- *Execute*
- *terraform init – initializes the bucket*
- *terraform plan – same as before*
- *terraform apply – same as before.*
- *Now your **tfstate** is within a shared S3 bucket*



Terraform Outputs

Terraform Outputs

- Allows you to output properties from the resources created during the **deploy** step.
- It can be the **ARN (Amazon Resource Name)** of any resource created.
- It Can be connection strings from AWS Aurora or AWS RDS databases created.

```
output bucket-arn {  
    value = aws_s3_bucket.my-first-s3-bucket.arn  
}
```

- *terraform plan -var site_bucket_name=xerris-academy-static-site*



Terraform Lab

Build a simple Terraform example with an S3 bucket

- *Define the provider*
- *Use environment variables for authentication*
- *Add a security policy to the bucket*
- *Create a vars file to capture the bucket-name*
- *Create a **dev.tfvars** file and invoke using that file from the command line*
- *Implement an **S3 backend** for the **tfstate***

