QATIP Intermediate
AWS Lab07
Managing AWS S3 Storage using Terraform

Contents

Lab Objectives

1. In this lab, you will use terraform to create and manage the following AWS resources...
    - S3 Bucket
    - Bucket Policy
    - File Uploads
    - Lifecycle Policy (to transition storage classes and delete objects after a set period)
    - Pre-Signed URL (to share secure access to objects)

Teaching Points

This lab focusses on using terraform to create and manage AWS storage resources, including creating S3 buckets, uploading data, storage management policies and Pre-Signed URLs.

Before you begin

1. Ensure you have completed Lab0 before attempting this lab
2. In the IDE terminal pane, enter the following command...

    ```
    cd /aws-tf-int/labs/07
    ```

3. This shifts your current working directory to /labs/07. Ensure all commands are executed in this directory
4. Close any open files and use the Explorer pane to navigate to and open the bonuslab

Solution

The solution to this lab is in folder /labs/solutions/07 Try to use this only as a last resort if you are struggling to complete the step-by-step processes.

Task 1: Create an S3 Bucket

Try It Yourself

1. Define an **S3 bucket** with Terraform.
2. The bucket name must be **globally unique**.
3. Set bucket versioning to **enabled**.
4. Deploy and verify the bucket in **AWS Console**.

Step-by-Step

Update main.tf as follows...

```
provider "aws" {
  region = "us-east-1"
}
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-s3-bucket-1234"
}
resource "aws_s3_bucket_versioning" "versioning_example" {
  bucket = aws_s3_bucket.my_bucket.id
  versioning_configuration {
    status = "Enabled"
  }
}
```

Save and apply...

```
terraform init
terraform plan
terraform apply -auto-approve
```

Verify the **S3 bucket** in **AWS Console**.

Task 2: Create an S3 Bucket Policy

Try It Yourself

1. Add an **S3 bucket policy** to enforce **public read restrictions**.
2. Deny public access to **all objects**.

Step-by-Step

1. Modify main.tf to add a **bucket policy**:

```
resource "aws_s3_bucket_policy" "bucket_policy" {
 bucket = aws_s3_bucket.my_bucket.id
 policy = <<POLICY
 {
  "Version": "2012-10-17",
  "Statement": [
   {
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::my-unique-s3-bucket-1234/*",
    "Condition": {
     "Bool": {
      "aws:SecureTransport": "false"
     }
    }
   }
  ]
 }
 POLICY
}
```

2. Apply the policy with `terraform apply -auto-approve`
3. Verify in the **AWS Console** under **S3 > Permissions > Bucket Policy**.

Task 3: Upload Files to S3

Try It Yourself

1. Upload **multiple files** to S3.
2. Assign **correct metadata** (e.g., MIME type).

Step-by-Step

1. Modify main.tf to upload multiple files dynamically...

```
resource "aws_s3_object" "files" {
  for_each = fileset("${path.module}/static_files", "**/*")
  bucket = aws_s3_bucket.my_bucket.id
  key = each.value
  source = "${path.module}/static_files/${each.value}"
  etag = filemd5("${path.module}/static_files/${each.value}")
  content_type  = lookup({
   ".jpg" = "image/jpeg",
   ".png" = "image/png",
   ".txt" = "text/plain"
  }, regex("\\.[^.]+$", each.value), "application/octet-stream")
 }
```

2. Apply using `terraform apply -auto-approve`
3. Verify in **AWS Console > S3 > Objects**.

Task 4: Implement an S3 Lifecycle Policy

Try It Yourself

1. Apply a **lifecycle policy** to:
   - Move objects to **Glacier after 30 days**.
     - Delete them **after 90 days**.

Step-by-Step

1. Modify main.tf...

```
resource "aws_s3_bucket_lifecycle_configuration" "lifecycle" {
  bucket = aws_s3_bucket.my_bucket.id

  rule {
   id = "storage-management-policy"
   status = "Enabled"

   transition {
    days        = 30
    storage_class = "GLACIER"
   }
```

```
      expiration {
        days = 90
      }
    }
  }
```

2. Apply using terraform apply -auto-approve
3. Verify in **AWS Console > S3 > Lifecycle Rules**.

Task 5: Generate a Pre-Signed S3 URL (Equivalent to SAS Token)

Try It Yourself

1. Generate a **Pre-Signed URL** for secure file access.
2. Allow **Read-Only access** for **1 hour**.

Step-by-Step

1. Add a **data block** to generate the Pre-Signed URL...

```
data "aws_s3_object" "example" {
  bucket = aws_s3_bucket.my_bucket.id
  key    = "example.txt"
}

data "aws_iam_policy_document" "signed_url_policy" {
  statement {
    actions   = ["s3:GetObject"]
    resources = [data.aws_s3_object.example.arn]
  }
}
```

2. Output the **Pre-Signed URL**...

```
output "pre_signed_url" {
  value = "aws s3 presign s3://${aws_s3_bucket.my_bucket.id}/example.txt --expires-in 3600"
  sensitive = true
}
```

3. Apply using **terraform apply -auto-approve**
4. Generate a **Pre-Signed URL**...

   terraform output pre_signed_url
5. Use the **URL** in a web browser to verify access.

Task 6: Using the Pre-Signed URL

1. Open a **browser**.
2. Paste the **Pre-Signed URL** from Terraform output.
3. If valid, the **file should download**.

Task 7: Lab Clean-Up

1. Remove all AWS resources using **terraform destroy -auto-approve**
2. Verify in **AWS Console**.