Exercise 2.0 Terraform S3 Backend Configuration (30 mins)

For this exercise we will use the AWS web console to create and configure and Amazon S3 bucket to be used as a Terraform Backend.

Note that it is not recommended to create the necessary AWS backend resources as part of the configuration you are deploying since a terraform destroy will also destroy your backend. These resources need to be created and managed independently. While we use the AWS console here, clearly these resources could also be created using the CLI.

# Create an S3 Bucket

* Open a browser in your vm and navigate to <https://console.aws.amazon.com>.
* Sign in using the credentials you have been using for the AWS CLI
* In the search box at the top type “S3” and choose the S3 service.

A screenshot of a computer

Description automatically generated

* A screenshot of a computer

  Description automatically generatedClick on “Create Bucket” and choose a bucket name (which must be unique) and a region.
* Scroll down and under “Bucket Versioning” select “Enable”
* A screenshot of a computer

  Description automatically generatedA screenshot of a computer

  Description automatically generatedScroll down to “Default Encryption” and click “Enable”. Choose “AWS Key Management Service key” as the encryption type and leave the other options in their default settings as shown. Click “Create Bucket” at the bottom

# Modify the Bucket Policy

* Click on your new bucket in the list of S3 buckets
* Look under the “Properties” tab of your bucket to find the ARN of your bucket and copy that to a text editor like Notepad.
* From a git bash command line run the aws command to get your user ARN and copy it to the same text editor:

$ aws sts get-caller-identity --output json

(Copy in a Git Bash shell can be achieved by selecting with the mouse and typing CTRL+INSERT to copy the text to the clipboard)

* A screenshot of a computer

  Description automatically generatedNow go to the “Permissions” tab of your S3 bucket, scroll down to the “Bucket Policy” box and click “Edit”
* Amend the policy to match the policy below, substituting your user ARN and S3 ARN, then click “Save Changes” at the bottom :

{

    "Version": "2012-10-17",

    "Statement": [

        {

            "Effect": "Allow",

            "Principal": {

                "AWS": "<your\_user\_arn>"

            },

            "Action": "s3:ListBucket",

            "Resource": "<your\_bucket\_arn>"

        },

        {

            "Effect": "Allow",

            "Principal": {

                "AWS": "<your\_user\_arn>"

            },

            "Action": [

                "s3:GetObject",

                "s3:PutObject"

            ],

            "Resource": "<your\_bucket\_arn>/\*"

        }

    ]

}

# Create a DynamoDB Table

* One of the big advantages of using and S3 bucket as a Terraform backend is that it means multiple team members can modify the state files without causing conflicts.
* If this is to work, we need to consider what would happen if two people try to modify the state file at the same time. For some backends (including S3 buckets) Terraform allows **state locking** – which prevents write operations on a state file while other write operations are happening.
* For an S3 bucket we need to supply a database to lock out state – we will use a DynamoDB table.
* A picture containing text, monitor, indoor, black

  Description automatically generatedSearch for dynamodb in your AWS console and click to create a new table
* A screenshot of a computer

  Description automatically generatedScroll down and find the orange “Create Table” button
* A picture containing text, indoor, computer, screenshot

  Description automatically generatedName the table anything you like, but set the **Partition key** string to LockID. Leave the remaining setting as they are and click the “create table” button on the bottom right.

# Hook up your Terraform configuration

* You can use any of your existing Terraform root modules for this – if you are unsure just use the terraform-configure-s3-backend configuration in the labs/student directory on the website.
* Update the terraform block in main.tf as follows (using your own bucket\_name, region and dynamodb table name):
  + Note that the region must be the region code of your bucket and dynamodb table, like “eu-west-2” for London, not “London”

terraform {

  backend "s3" {

    bucket         = "<your\_bucket\_name>"

    key            = "terraform.tfstate"

    region         = "<your\_aws\_region>"

    dynamodb\_table = "<your\_dynamo\_dbtable\_name>"

  }

  required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 3.27"

    }

  }

}

# Initialize the Terraform S3 Backend

* This is done as usual with terraform init:

$ terraform init

Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically

use this backend unless the backend configuration changes.

Initializing provider plugins...

- Finding hashicorp/aws versions matching "~> 3.27"...

- Installing hashicorp/aws v3.74.3...

- Installed hashicorp/aws v3.74.3 (signed by HashiCorp)

Terraform has created a lock file **.terraform.lock.hcl** to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

**Terraform has been successfully initialized!**

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform,

rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

# Run Terraform Apply

* This will create your EC2 instance and save the Terraform state to your S3 Bucket.
* Type “terraform apply” in your git bash command line. Remember to respond “yes” to the prompt to create resources.
* A picture containing text, screenshot, computer, indoor

  Description automatically generatedOnce creation is complete, go to your AWS Console and locate your S3 Terraform backend. Click on it to see that the terraform.tfstate file is saved in the bucket.
* You can click on the file and choose the Open button to see the file contents

A screenshot of a computer

Description automatically generated with medium confidence

# Cleanup Resources

* Run “terraform destroy” from the git bash command line, remembering to type “yes” at the prompt.
* Note that the terraform.tfstate file still exists in your S3 bucket. It will persist as it is created and managed independently of your terraform configuration.
* To remove your AWS resources you will need to delete them manually using the console:
  + For the S3 bucket, click the “Buckets” link on the left-hand navigation panel of the Amazon S3 Console page, select the radio button next to your S3 bucket and click the “Empty” button. Follow the instructions, then repeat the process above and click the “Delete” button.
  + For the DynamoDB, click the “Tables” link on the left-hand navigation panel of the DynamoDB Console page, select the check box next to your DynamoDB and click the “Delete” button.