We create chemistry

Cloud Infrastructure Engineer: Exercise

Table of Contents

Prerequisites

Already working Terraform in the instance you are working.

Terraform

init

We just need to execute the command `terraform init` to start the project, it will download the provider selected, in this case aws, and build the environment with some terraform configuration.

(back to top)

plan

The command `terraform plan` will promt 8 new adds:

1. Source bucket
2. Target bucket
3. Lambda function
4. IAM Policies for Lambda function
5. IAM Roles for Lambda function
6. IAM Roles and policy attachment
7. Lambda permission
8. Bucket notificatopm

(back to top)

Considerations

To skip credential validations we use the provider flags:

```
skip_credentials_validation = true
skip_requesting_account_id  = true
skip_metadata_api_check     = true
access_key                  = "mock_access_key"
secret_key                  = "mock_secret_key"
```

In order to allow our lambda function work together with both S3 buckets, I am using an Identity Access Management (IAM). To do that, I define the policy an the role by:

- `aws_iam_policy.lambda_policy`
- `aws_iam_role.s3_copy_function`

To append that to the lambda function with:

- `aws_iam_role_policy_attachment.lambda_iam_policy_execution`

Now, it is time to grant the S3 bucket to trigger the lambda function by:

- `aws_lambda_permission.allow_bucket`
    - The `statement_id` is an id used by aws to build a grant rule for the S3 bucket.
    - The `action` identifies the service we would like to use. call a Lambda function.
- `aws_lambda_function.s3_copy_function` should specify:
    - The `role` we are using
    - The `handler` pointing to tha main.py file where the lambda function code is
    - The `runtime` in this case, python 3.6
- `aws_s3_bucket_notification.bucket_notification` Using the event `s3:ObjectCreated:*` over the source S3 bucket the system is ready to listen every time a file is added on the source.
- Lambda function: already commented in code, it just retrieve the name and key for the new file in order to copy on the target in a loop triggered by the previously mentioned event.

(back to top)

CI/CD

Three phases need:

- Source: Any git source i.e. github
- Install: Make sure or install Terrafom in the agent
- Deploy: Using the credentials stored, run the Terraform project

In order to be able to create an end-to-end CI/CD flow, we need a DynamoDB table and a S3 bucket to save the terraform updates (terraform-state-lock).

(back to top)

Installation

1. Clone the repo

```
git clone https://github.com/txitxo0/basf.git
```

2. Perfom

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

(back to top)

Code

main.tf

```
variable "project_name" {}
variable "region_name" {}
variable "env" {}

data "archive_file" "lambda_function" {
  source_dir  = "${path.module}/lambda/"
  output_path = "${path.module}/lambda.zip"
  type        = "zip"
}


provider "aws" {
    region = var.region_name
```

```
    skip_credentials_validation = true
    skip_requesting_account_id  = true
    skip_metadata_api_check     = true
    access_key                  = "mock_access_key"
    secret_key                  = "mock_secret_key"
}

resource "aws_s3_bucket" "source_bucket" {
    bucket              = "s3-source-${var.project_name}-${var.region_name}"
    tags = {
        Name            = "S3 source bucket"
        Environment     = var.env
    }
    force_destroy = true
}

resource "aws_s3_bucket" "target_bucket" {
    bucket              = "s3-target-${var.project_name}-${var.region_name}"
    tags = {
        Name            = "S3_target_bucket"
        Environment     = var.env
    }
    force_destroy = true
}

resource "aws_iam_policy" "lambda_policy" {
  name        = "lambda-policy-${var.project_name}-${var.region_name}"

  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:ListBucket",
        "s3:GetObject",
        "s3:CopyObject",
        "s3:HeadObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::s3-source-${var.project_name}-${var.region_name}",
        "arn:aws:s3:::s3-source-${var.project_name}-${var.region_name}/*"
      ]
    },
    {
      "Action": [
        "s3:ListBucket",
        "s3:PutObject",
        "s3:PutObjectAcl",
        "s3:CopyObject",
        "s3:HeadObject"
      ],
      "Effect": "Allow",
```

```
      "Resource": [
        "arn:aws:s3:::s3-target-${var.project_name}-${var.region_name}",
        "arn:aws:s3:::s3-target-${var.project_name}-${var.region_name}/*"
      ]
    },
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
EOF
}

resource "aws_iam_role" "s3_copy_function" {
    name = "app-lambda-${var.project_name}-${var.region_name}"
    assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Effect": "Allow"
    }
  ]
}
EOF
}

resource "aws_iam_role_policy_attachment" "lambda_iam_policy_execution" {
  role = "${aws_iam_role.s3_copy_function.id}"
  policy_arn = "${aws_iam_policy.lambda_policy.arn}"
}

resource "aws_lambda_permission" "allow_bucket" {
    statement_id = "AllowExecutionFromS3Bucket"
    action = "lambda:InvokeFunction"
    function_name = "${aws_lambda_function.s3_copy_function.arn}"
    principal = "s3.amazonaws.com"
    source_arn = "${aws_s3_bucket.source_bucket.arn}"
}

resource "aws_lambda_function" "s3_copy_function" {
    filename = "lambda.zip"
    source_code_hash = data.archive_file.lambda_function.output_base64sha256
    function_name = "app-lambda-s3-copy-${var.project_name}-${var.region_name}"
```

```
    role = "${aws_iam_role.s3_copy_function.arn}"
    handler = "main.handler"
    runtime = "python3.6"

    environment {
        variables = {
            TARGET_BUCKET = "s3-target-${var.project_name}-${var.region_name}",
            REGION = "${var.region_name}"
        }
    }
}

resource "aws_s3_bucket_notification" "bucket_notification" {
    bucket = "${aws_s3_bucket.source_bucket.id}"
    lambda_function {
        lambda_function_arn = "${aws_lambda_function.s3_copy_function.arn}"
        events = ["s3:ObjectCreated:*"]
    }

    depends_on = [ aws_lambda_permission.allow_bucket ]
}
```

(back to top)

versions.tf

```
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
        }
        template ={
            source = "hashicorp/template"
        }
    }
    required_version = ">= 0.13"
}
```

(back to top)

terraform.tfvars

```
project_name="basf_exercise"
region_name="eu-west-1"
env="dev"
```

(back to top)

main.py

```
#!/usr/bin/env python3

import os
import logging
import boto3

LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)

# Reading environment variables
TARGET_BUCKET = os.environ.get('TARGET_BUCKET')
REGION = os.environ.get('REGION')

s3 = boto3.resource('s3', region_name=REGION)

def handler(event, context):
   LOGGER.info('Event structure: %s', event)
   LOGGER.info('TARGET_BUCKET: %s', TARGET_BUCKET)

# For every new event we have in Records
   for record in event['Records']:
       # Get the name and key element
       src_bucket = record['s3']['bucket']['name']
       src_key = record['s3']['object']['key']

       copy_source = {
           'Bucket': src_bucket,
           'Key': src_key
       }
       LOGGER.info('copy_source: %s', copy_source)

       # Set the s3 buckect target
       bucket = s3.Bucket(TARGET_BUCKET)
       # Copy the element
       bucket.copy(copy_source, src_key)

   return {
       'status': 'ok'
   }
```

(back to top)