Terraform

Favorite	☐
Archived	☐
Fleeting	☐
Area/Resource	
Project	
Type	
Review Date	
Image	
URL	
Created	@February 6, 2023 5:05 PM
Updated	@February 8, 2023 10:54 AM
Root Area	
Project Area	
Updated (short)	02/08/2023
Pulls	
Resource Pulls	
Project Archived	
URL Base	
🍳 Recipe Divider	🥗🥗🥗 RECIPE BOOK PROPERTIES 🥗🥗🥗
🍳 Recipe Tags	
📚 Book Divider	📚📚📚 BOOK TRACKER PROPERTIES 📚📚📚
📚 Author	
📚 Date Started	
📚 Date Finished	
📚 Book Status	
📚 Rating	

💡 Terraform is an open-source tool used for provisioning infrastructure resources. It is based on Infrastructure-as-Code, build change and manage your infrastructure in a safe, consistent and repeatable way by defining resource configurations that you can version, reuse and share.

Advantages:

- Infrastructure lifecycle management
- Version control commits
  - Collaborate on infra
- Useful for stack backed deplyoments
- State-based approach to track resource changes throughout deployments

https://www.youtube.com/watch?v=tomUWcQ0P3k

💡 To prevent you from building an app on the cloud using a the GUI, it allows you to provision resources and configure your infrastructure as code. It means you are going to be able to reproduce projects again in the future. It provides a way to build, change and version infrastructure safely. It represents your code in the hashicorp configuration language which is a declarative code.

- It can be though of as a blueprint showing the end state you wish your environment to reproduce.

Setup

1. Install terraform client
2. export GOOGLE_APPLICATION_CREDENTIALS with necessary permissions to provision resources

Service account permissions

⚙️ Terraform SA will need the permissions to provision the resources and services that you wish terraform to complete. In Production you would use the concept of least privilege

Create infrastructure with Terraform

⚙️ Terraform needs GOOGLE_APPLICATION_CREDENTIALS env_var set to function properly

Structure

⚙️ There are three files to use with terraform: main.tf, variables.tf and terraform-version.

terraform-version: What version of tf is installed and to be used

main.tf

```
terraform {
  required_version = ">= 1.0" -- compatible versions
  backend "local" {}  # Can change from "local" to "gcs" (for google) or "s3" (for aws), if you would like to preserve your tf-state online
  required_providers {
    google = {
      source  = "hashicorp/google"
    }
  }
}

provider "google" {
  project = var.project
  region = var.region
  // credentials = file(var.credentials)  # Use this if you do not want to set env-var GOOGLE_APPLICATION_CREDENTIALS
}

# Data Lake Bucket
# Ref: https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/storage_bucket
resource "google_storage_bucket" "data-lake-bucket" {
  name          = "${local.data_lake_bucket}_${var.project}" # Concatenating DL bucket & Project name for unique naming
  location      = var.region

  # Optional, but recommended settings:
  storage_class = var.storage_class
  uniform_bucket_level_access = true
  versioning {
    enabled     = true
  }

  lifecycle_rule {
    action {
      type = "Delete"
    }
    condition {
      age = 30  // days
    }
  }

  force_destroy = true
}

# DWH
# Ref: https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/bigquery_dataset
resource "google_bigquery_dataset" "dataset" {
  dataset_id = var.BQ_DATASET
  project    = var.project
  location   = var.region
}
```

Backend

💡 The backend is where terraform will store it's current state. The state is the representation of the current terraform configurations and infrastructure. Each time tf is changed or updated the state file is updated to reflect the changes. It can be stored locally or on a file storage cloud provider such as GCS. Terraform uses a state file so that it knows what actions to take in subsequent runs. It can be stored as a JSON file.

Required providers

💡 Can be thought of as plugins that can connect to different cloud providers so that terraform can interact with them and provision resources.

Provider block

💡 This block provides terraform with the necessary credentials to connect to the google cloud or cloud account. In our case our credentials are stored in a ENV variable.

Resource block

💡 The resource block is used to define which resources you want to provision, it's name and configurations, and how tf should manage it. It tells tf what changes it should make to the infrastructure.

variables.tf

💡 The variables file serves to abstract configuration values from the main configuration file. By using variables you can make terraform more reusable, maintainable and flexible. The variables file defines a set of variables that can be used in the main file to parameterize it.

```
locals {
  data_lake_bucket = "dtc_data_lake"
}
```

```
variable "project" {
  description = "Your GCP Project ID"
}

variable "region" {
  description = "Region for GCP resources. Choose as per your location: https://cloud.google.com/about/locations"
  default = "europe-west6"
  type = string
}

variable "storage_class" {
  description = "Storage class type for your bucket. Check official docs for more info."
  default = "STANDARD"
}

variable "BQ_DATASET" {
  description = "BigQuery Dataset that raw data (from GCS) will be written to"
  type = string
  default = "trips_data_all"
}
```

- [x] ~~What is the difference between local and variable?~~

Locals

💡 Can be seen as constants.

Variables

💡 Generally passed at runtime, more variable and less constant.

Execution

Commands:

1. terraform init: Initialize & Install
2. terraform plan: Match changes against the previous state
    a. Detects changes from previous state
3. terraform apply: apply changes to cloud
    a. Finds a way to match the new configurations and applies them to the cloud
4. terraform destroy: remove your stack from cloud