FGTC 1

For AI Applications

Description

FGTC, an abbreviation for 'From the Ground To the Cloud' is a project founded to produce highly structured, easy-to-manage IaC templates that create efficient, reliable, and scalable infrastructure.

FGTC 1, an IaC-Template-Solution for AI applications enabling faster deployments, optimizing costs, enhancing scalability, ensuring security best practices and compliance requirements. The template empowers AI companies with a reliable Cloud Infrastructure that comprises the appropriate computational resources for AI applications.

General Concepts

Cloud: a short form for "cloud computing" It refers to the delivery of computing services such as servers, storage, databases, networking, software, analytics, and intelligence over the Internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale.

AWS: an abbreviation of Amazon Web Services. It's a cloud computing platform provided by Amazon that offers a wide range of services

IaC: an abbreviation of Infrastructure as Code. It's a practice of managing and provisioning Cloud resources through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

Terraform: an Infrastructure as Code (IaC) tool provided by HashiCorp that makes it easier to automate, version, and manage your infrastructure.

IaC template: an easy to manage and modify set of configuration files and folders that form the structure of the product

modular design: breaking down code into reusable, composable modules simplifying management and allowing you to combine components in flexible ways.

Vault: a tool for securely storing and managing secrets provided by HashiCorp. It provides a centralized location for storing sensitive information like passwords, API keys, and encryption keys.

General Approach

Architecture

our approach is aiming to make a decomposable structure in order to make it easy to manage and modify the template, so we are adopting what we call a Modular Design a flexible yet reliable architecture that:

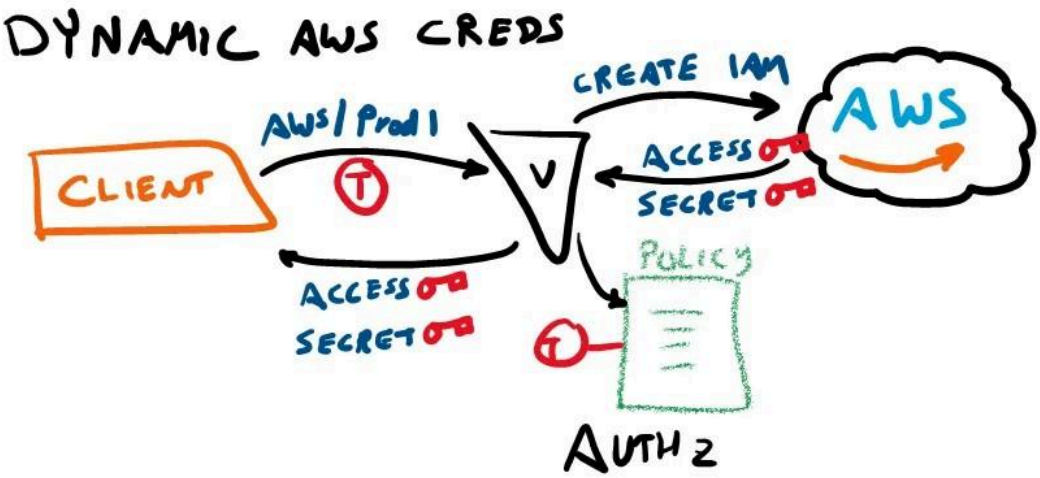
- Encapsulate distinct components into their own modules.
- Make modules as self-contained and decoupled as possible.
- Use input variables and outputs to connect modules.

Security

In order to avoid hardcoding credentials directly in the Terraform configurations. We need either to encrypt credentials or to use a credential provider to handle the task.

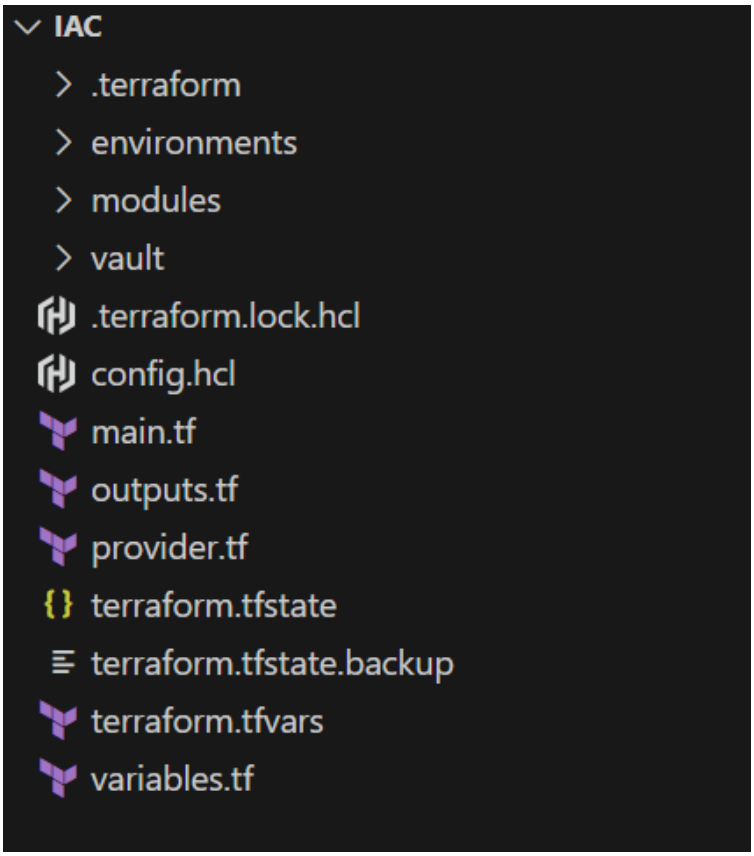
In the pursuit of securing our API keys we chose to work with Vault. Vault offers a High level of encryption with industry standard algorithms. It provides a centralized repository for storing all your sensitive information, which also maintains a clear and organized structure making it easier to manage and track secrets.

Vault helps avoid hardcoding credentials directly in your Terraform configurations by dynamically generating temporary AWS credentials.



Product structure

Template Directory



NOTE: Auto-generated files are files that are generated by Terraform automatically which we are not going to discuss. Files like ".terraform.lock.hcl", "terraform.tfstate.backup", "terraform.tfstate" and files under the ".terraform" directory.

environment/

This directory contains configuration files for different environments (e.g., development, staging, production).

modules/

This directory is the library that contains and organizes our modules, it serves as a central repository for reusable infrastructure components.

Those are the modules that we will be using in our infrastructure. Here is the VPC which could be distributed in multiple regions and availability zones. The VPC has an Internet Gateway, public subnets and a private subnet for each public subnet. Each public subnet contains a nat-gateway that controls the traffic from & to the corresponding private subnet. The Internet Gateway contains a routing table that routes traffic to the public subnets and Each nat-gateway contains a routing table that routes traffic to the private subnets. And we are implementing the AWS System Manager (SSM) in order to connect to the EC2 instances (including those in private subnets) via a secure session.

The subnet module leverages the VPC module's output to configure its network settings. Similarly, the router module utilizes both the VPC and subnet module's outputs to establish routing. Finally, the nat-gw module relies on the subnet module's output to define appropriate Network Address Translation (NAT) rules for outbound traffic.

The computational workflow leverages a distributed architecture, with EC2 instances deployed across the Private Subnets on multiple Availability Zones (AZs) for redundancy and fault tolerance. These instances, serving as the compute nodes, are seamlessly integrated with a shared FSx for Lustre file system, providing high-performance parallel file access for data-intensive workloads. Additionally, the S3 bucket serves as a scalable and durable object storage layer, offering flexible data storage and retrieval capabilities. To ensure secure and controlled access, IAM policies and security groups are meticulously configured to restrict access to specific resources and enforce granular permissions, thereby safeguarding the integrity and confidentiality of the system and its data.

Each module contain those files:

main.tf

Contains the code that define that specific module

variables.tf

Contains the variables that have been used in the main.tf file

output.tf

Defines the desired output of that module that will be used later as an input for an other module

main.tf

The principal main.tf file in the root directory is a central configuration file in Terraform, used to define the infrastructure you want to provision. It acts as the starting point for Terraform to understand your infrastructure needs.

variables.tf

The variables.tf file in the root directory provides all the necessary variables for the main.tf file, which were imported automatically from the terraform.tfvars.

terraform.tfvars

In the terraform.tfvars (in root directory) file you **MUST INPUT THE VARIABLES** desired values. Variables are defined as key=value pairs:

- address= "" # The address of the vault provider in the provider.tf file
- backend = "" #backend attribute for the vault_aws_access_credentials in the provider.tf file
- role = "" #role attribute for the vault_aws_access_credentials in the provider.tf file
- region = ""
  # NOTE: EC2p5 (instance_type) instances does only exist in those regions [Europe (Stockholm) — eu-north-1, Asia Pacific (Tokyo) — ap-northeast-1, US West (Oregon) — us-west-2, US East (Ohio) — us-east-2, US East (N. Virginia) — us-east-1]
  # NOTE: EC2pe5 (instance_type) instances does only exists in the "US East (Ohio) — us-east-2" region
- environment = "" # Environment on which we are deploying the Infrastructure.
- vpc_cidr_block = "" # The CIDR block for your VPC

- vpc_name = "" # The name of your VPC
- availability_zones = [] # changing the number of AZ will result in an error
- instance_number = #Specify your desired number (of type number not string) of instances as integer
- instance_type = "" # customize to the instance type you want to use and keep in mind the region dependency.
- ami_id = "" # Replace with your AMI_Id (the template image you want to clone from).

  # NOTE s3_life_cycle is Enabled, that mean that the data will be deleted after 1 year
- storage_users_ip_addresses = [] # ADD users machines IP addresses that need to Access the S3 bucket and FSx for Luster (full access CRUD)
- fsx_storage_capacity = # Set the desired amount (of type number not string) of storage for the fsx for Luster

  NOTE: The AWS account credentials will be entered when setting up Vault.

output.tf

The output.tf file in the root directory represents the output of the whole template that will be displayed on the CLI after applying the configuration.

provider.tf

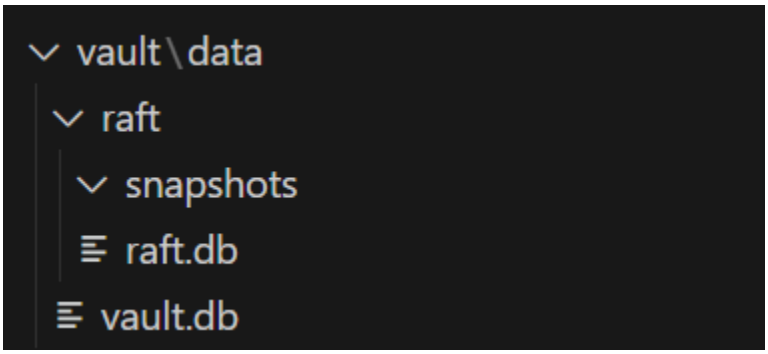
This file exists in the root directory and contains the provider required details (e.g aws, terraform, vault etc...).

config.hcl

This file resides in the root directory and contains the Vault configuration used by Terraform to access AWS temporary credentials managed by Vault.
This configuration specifies the directory where vault stores Data, Vault server listener and api_address.

vault/data/

This directory is where the vault stores its data. The directory that was specified in the config.hcl file



NOTE: YOU NEED TO ADD THIS DIRECTORY UNDER THE ROOT (at the level of config.hcl file). It is no included in the Iac Template

Cloud Architecture