**AWS CloudFormation vs Azure ARM vs Terraform(Iac)**

**TABLE OF CONTENT**

**1. Introduction**

**2. Configuration Language**

**3. Modularity**

**4. Planning**

**5.Credentials**

**6.Access Management**

**7.Cleaning Up**

**8.Conclusion**

**9. About CloudThat**

**10. FAQs**

**Introduction:-**

Infrastructure as code (IaC) is the process of managing IT infrastructure using configuring infrastructure through code instead of manually. A manual process needs operators and system administrators to configure any changes to the infrastructure.

Using IaC, DevOps teams can ably store the infrastructure configuration code and application code in a central repository. IaC ensures consistent and secure deployment. By avoiding manual error-prone configuration and deployment, security standards and policies are easy to maintain. And, Now DevOps engineers can able to improve scalability and improve productivity through faster deployments using Iac.

Well, DevOps engineers can choose from various tools when implementing IaC. Three of the most popular tools for Amazon Aws are CloudFormation, Microsoft Azure are Azure Resource Manager (ARM) templates and HashiCorp Terraform. Proprietary offerings like ARM templates and Aws CloudFormation allows infrastructure configuration exclusively on their respective cloud providers. Furthermore, HashiCorp’s Terraform supports multiple cloud providers.

**Configuration Language:-**

Azure ARM templates can only be written in JSON while AWS CloudFormation supports both JSON and YAML. On the other hand, HashiCorp Terraform configuration files uses either of two formats: Terraform domain-specific language (HashiCorp Configuration Language format [HCL]), which is the recommended approach, or JSON format if the files need to be machine-readable.

Terraform configuration files are convinient to document and maintain, so tracking down errors or security bugs is less difficult.

**Modularity:-**

**Modules** allow us to reuse configuration files in other deployment environments. Modules are components that can be reused across multiple CloudFormation templates and is used just like a native CloudFormation resource. **CloudFormation** does not have native support for modules. It allows you to use something called nested stacks as modules. Given an example, you can have a standard configuration of how you wish to provision an S3 bucket in your organization. So, you create a standard CloudFormation template that creates S3 buckets. When if an end-user wants to create the S3 bucket they can use this CloudFormation template as a nested stack and create a standard S3 bucket.

There is also a unknown service of AWS, the AWS Service Catalog that can help you with the modularity of your AWS CloudFormation. Service Catalog is an AWS service that is designed specifically for organizations that want to limit the scope of AWS Services to meet compliance, security, cost, or performance requirements. And estimate what? AWS Service Catalog uses CloudFormation templates in the backends.

Modularize the configuration code for **ARM templates** by nesting templates by breaking your template into multiple template files .You can modularize ARM templates using “nested ARM templates” which are (relatively) flexible also — you can bring in nested templates from files, urls etc. — but the file needs to be accessible by Azure Resource Manager during deployment (which means you cannot have the nested template just locally — it has to be stored somewhere ARM can fetch it, e.g. in the Storage account)..

A **Terraform module** allows you to create logical theorization on the top of some resource set. By way of explanation, a module allows you to group resources together and reuse this group later, possibly many times.A Terraform module is a set of Terraform configuration files(.tf files) in a single directory. Even a simple configuration consisting of a single directory with one or more .tf files is a module. So in this sense, every Terraform configuration is bit of a module.

**root module:**

When you run Terraform commands directly from a directory, it is considered the root module.

**Child module:**

A module that is called by another configuration is mentioned to as a child module.

Diagram

Description automatically generated

**Planning:-**

With usage of terraform plan command, you can perform dry runs of your code, letting you know what the deployment changes. It’s kind of like a preview before making the actual changes.

ARM provides the similar functionality using the what-if operation. This operation highlights the changes that will occur to the existing resource if you deploy the template.

**Cleaning Up:-**

Usage of running the terraform destroy command to destroy resources in Terraform. This command is helpful for terminated resources.

An Azure ARM template doesn’t have an explicit destroy command. You need to use the Azure CLI or the Azure console to manually remove the resources. Removing these unwanted resources helps keep your infrastructure secure by shrinking your attack surface.

In Aws CoudFormation ,You can delete the CloudFormation stack (and therefore all resources contained within it) through either cli or console.