Terraform

Deploying Infrastructure with Terraform

Challenges that are solved by terraform

1. Terraform allows us to create reusable code that can deploy identical set of infrastructure in repeatable fashion
2. Terraform reads to the configurations(Hardening Rules) and deploys them in any of the cloud services like AWS,Azure etc.
3. Terraform supports thousands of providers(AWS, OCI, Azure, GCP, etc.)
4. Once we learn the terraform core concepts we can write code to create and manage infrastructure across all the providers

Infrastructure as Code (IAC)

1. There are two ways in which we can create and manage the infrastructure
    a. Manual Approach: This wastes lot of time and need human intervention every time and not recommended
    b. Automation Approach:
        i. Depending on the type of task the tools for automation changes.
        ii. There are wide variety of tools used for automation like Python, Terraform, Ansible, Cloud Formation etc.
        iii. Example: Lets say we need to create a set of resources(Virtual Machines, Database, S3, AWS Users) must be created with exactly same configurations in Dev, QA and Prod.
        iv. So to deploy the above given entire infrastructure we use a IAC tool in multiple set of environments

Basics of Infrastructure as Code (IAC)

1. Infrastructure as code is the managing and provisioning of the infrastructure through code instead of manual process

Benefits of Infrastructure as Code (IAC)

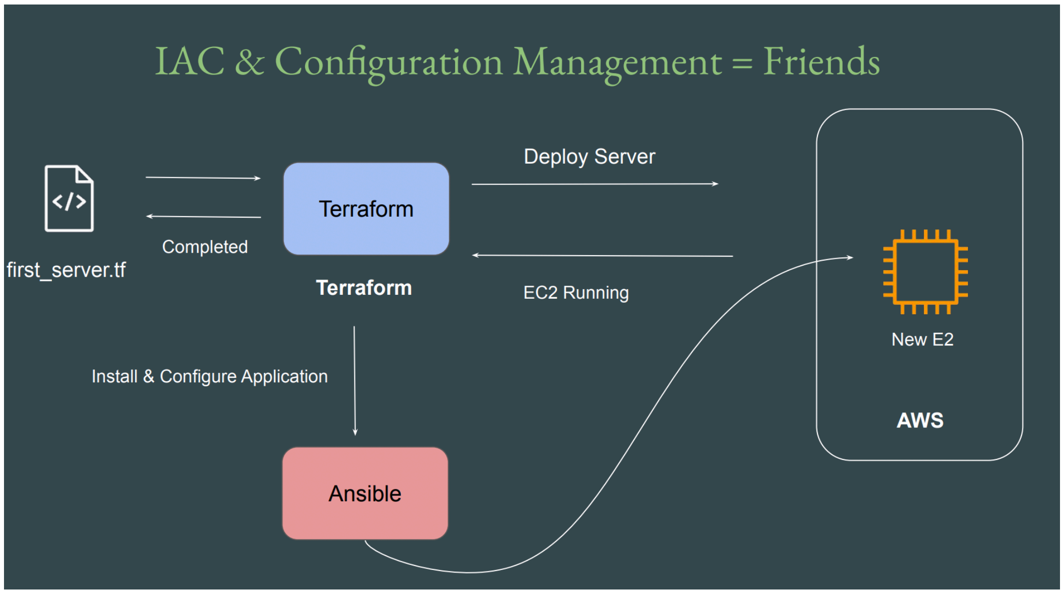
1. Speed of Infrastructure management. We can deploy more than 100 services all at once in a repeatable fashion
2. Low risk of human error
3. Version control
4. Easy Collaboration between teams

IAC Tools

1. There are multiple IAC tools used to deploy infrastructure as code
    a. Terraform
    b. Cloud Formation
    c. Heat
    d. Ansible etc.

2. Infrastructure as Code divided into two parts
    a. Infrastructure Orchestration (Terraform, Cloud formation): Infrastructure Orchestration is primarily used to create and mange infrastructure environments (we deal with raw infrastructure)
    b. Configuration Management (Ansible, Chef): Configuration Management tools are primarily used to maintain the desired configuration of the system(inside servers)

3. If we want to add IAC solution it is recommended to use the one that is part of the Infrastructure Orchestration category



Which IAC tool to choose?

1. IAC tools are chosen based on the below points
    a. Is your infrastructure going to be vendor specific in longer time? Like AWS
    b. Are you planning to have multi cloud/hybrid cloud base infrastructure?
    c. How well does it integrate with Configuration Management tools
    d. Price and Support

Basics of Authentication and Authorization:

1. Authentication is the process verifying who a user
2. Authorization is the process of verifying what they have access to
3. Terraform needs access credentials with relevant permission to create and manage the environment

4. In terraform we will also associate the username and password of the specific provider in which you want to create the resource
5. Terraform will go ahead and authenticate through the provider and after the authorization the respective resource is created
6. Depending on the provider the types of credentials changes(Different for AWS, Azure, GCP, etc.)

Launching virtual machine through Terraform

Below are the configurations that user has to consider before launching the VM

1. Before launching the VM we need to select the region to launch the VM in Terraform
2. Before creating VM we need to define the specifications related to CPU, memory, storage and OS
3. From terraform plan we can see what all changes we are making with terraform apply we can create the resources

Providers and Resources

Providers:

1. A Provider is a plugin that lets terraform to manage the external API's
2. When we run terraform init, plugins required to the provider are automatically downloaded and saved locally to .terraform directory
3. Whenever we define new provider like AWS,Azure etc. we have to do terraform init so that it will download the necessary packages

Resource:

1. A resource block declares a resource of a given type ("aws_instance") with a given local name("myec2")

2. Every time we need to create a new services we create a resource block in terraform file that will create different services
3. Resource type and Name together serve as an identifier for a given resource and so must be unique

Note: We can only use the resource that are supported by a specific provider

The core concepts, standard syntax remains similar across all the providers

Provider Maintainers

1. There are 3 primary types of providers in terraform
    a. Official a. Owned and Maintained by hashicorp
    b. Partner a. Owned and Maintained by Technology company that maintains direct partnership with hashicorp
    c. Official a. Owned and Maintained by Individual Contributors

Provider Namespaces

1. Namespaces are used to help users identify the organisation and publisher responsible for the integration

- Terraform requires explicit source information for any providers that are not hashicorp maintained , using a new syntax in the request_providers nested block inside the terraform configuration block

Terraform Destroy

Approach-1:

1. Terraform destroy allows us to destroy all the resources that are created within the folder

Approach-2:

1. Terraform destroy with **-target** flag allows us to destroy specific resource

We can also destroy the resource without using the terraform destroy we can do that by commenting the script that we don't want and we can do terraform plan that will assume that commented code is not existing and plans whatever there in the script and destroys whatever additional things present. These are multiple ways of destroying resources

State Files

How does terraform know whether the resource is deleted or how it will add new resource when we re-add the same destroyed

1. Terraform stores the state of the infrastructure that is being created from the TF files
2. This state allows terraform to map the real world resource to your existing configuration
3. Whenever we create a resource(like ec2) terraform also stores the details of the instance in a file called "terraform state file"
4. From this state file terraform tracks the information of the infrastructure related data
5. So if we destroy any of the resource the associated state file is also removed
6. So this is how terraform knows the respective resource is being destroyed
7. The state file contains the information of the resources that are currently live

Desired State & Current State

Desired State:

1. Terraform primary function is to create, destroy and modify the infrastructure resource to match the desired state described in a terraform configuration

Current State:

1. Current state is the actual state of resource that is currently deployed

Note: It is not always considered that Desired State = Current State

- Terraform tries to ensure that the deployed infrastructure is based on desired state
- If there is a difference between the 2 states terraform plan presents a description of the changes necessary to achieve the desired state

Provider Versioning

1. Provider are released separately from Terraform itself. They have different set of version numbers
2. The version of provider plugin keeps on updating at regular intervals irrespective primarily by the terraform version itself
3. Sometimes it is difficult to update the version in production environment because of we have resources/ dependencies based on older version when it automatically gets updated it may crash the whole system

Explicitly Setting Provider Version

1. During Terraform init if the version is not specified explicitly, the most recent provider will be downloaded during the initialisation
2. For production use, you should constrain the acceptable version via configuration, to ensure the new versions with breaking changes will not be automatically installed
3. When we mention the tilda (~>)version and we do the terraform init it will initialise a lock file. This lock file will have the exact version of the resource at that period of time if you change the version to any other series it will generate the error because we have specified version in the lock file

Dependency Lock file

1. Terraform dependency lock file allows us to lock to the specific version of the provider
2. If a particular provider already have a selection recorded in the lock file, Terraform will always reselect that version for installation, even if a newer version is available.
3. You can override the behaviour by adding the upgrade option when you run the terraform init

Terraform Refresh

1. The terraform refresh command will check the latest state of your infrastructure and update the state file accordingly
2. You shouldn’t typically need to use the command, because terraform automatically performs the same refreshing action as a part of terraform plan/apply
3. Terraform refresh command is deprecated in newer versions of terraform

AWS Provider- Authentication Configuration

1. Sometimes we accidentally push our credentials in our code to GitHub and make them public. So to prevent this we came up with couple of approaches

Better Approach: We can store the credentials in the config file and push the changes such that the credentials are not exposed and helps in storing them securely. But there is a drawback with this approach. Lets say we have 10 people working on the same resource as we can hardcode one config file it will be difficult to put 10 different config files while a large team working on a same resource

AWS CLI

1. AWS CLI allows customers to manage AWS resources directly from CLI

2. When you configure Access/Secret key in AWS CLI, the location in which this credentials are stored is the same default location that terraform searches credentials from which is $HOME/.aws/.config and $HOME/.aws/.credential