Terraform

IAC (INFRASTRUCTURE AS CODE)

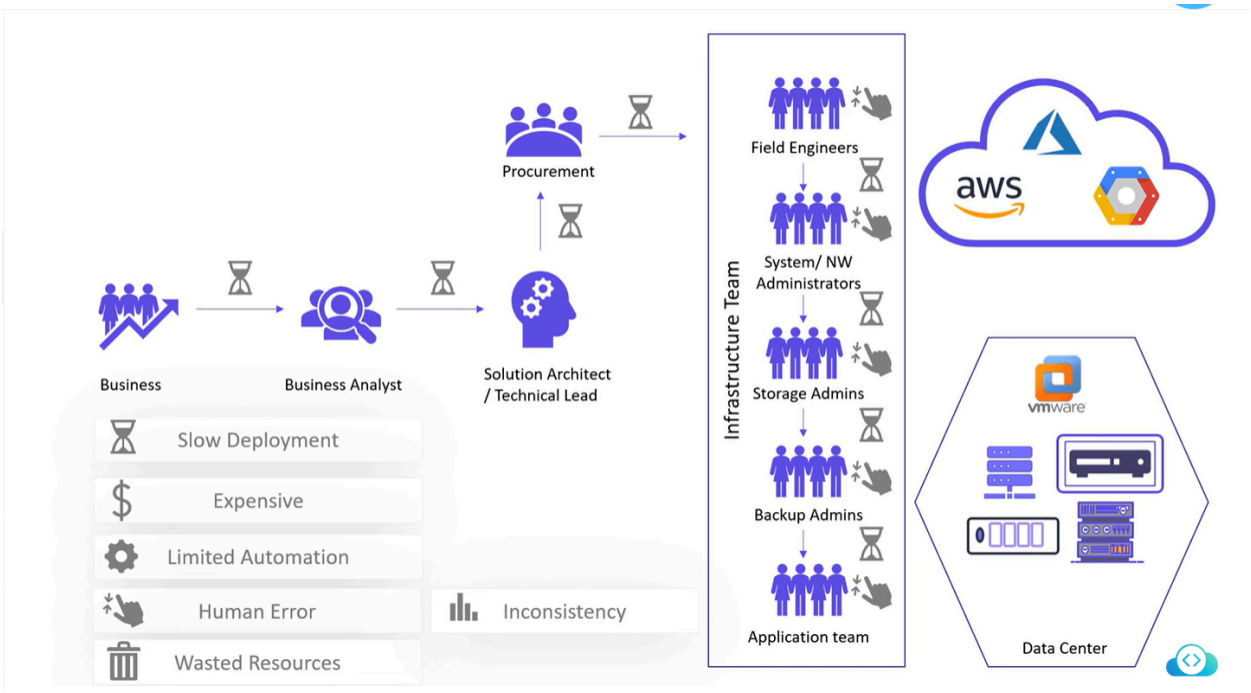
Infrastructure as Code (IAC) is the process of managing and provisioning computing infrastructure through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. This allows for more consistent and reliable infrastructure management.

IAC is a practice in which infrastructure is managed and provisioned using code and software development techniques. Such as version control and Continues integration.

The approach allows for consistent and repeatable infrastructure deployment., Reduce human errors and improves collaboration among teams.

TERAFORM:

Terraform is an open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON. Terraform supports a wide variety of cloud providers as well as custom in-house solutions.



Terraform Installation:

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/s
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https
sudo apt update && sudo apt install terraform
```

Popular IAC Tools

1. Terraform
2. Aws CloudFormation
3. Azure Resource Manager
4. .Ansible
5. Chef
6. Puppet

7. OpenStack --- Heat Template.

***What is Terraform and Why Terraform and also explain Below Commands*:**

1. Terraform init
2. Terraform Plan
3. Terraform Apply.

Terraform is an open-source infrastructure as code tool created by HashiCorp. It allow you to define and provisioned data center infrastructure using\
a high level configuration language known as "HashiCorp Configuration Language(HCL)". or Optionally Json. With this you can define your infrastructure in configuration file such as virtual Machines, Storage & Networking. This approch allows for virsion control & collaboration.

***Why Terraform*:**

Even through all cloud provider like Azure, Aws, GCP have their own IAC Services like, CloudFormation, Resource Manager and more. We are preferring terraform for below reasons.

1. Multi-Cloud Support: Terraform works with Aws, Azure, GCP and more.
2. State Management: Terraform Maintain state in files (terraform tf state), allowing tracking of changes.
3. Modular & Reusable Code: Terraform allows you to create module that can be reused across different environments.
4. Cost Effective: Terraform is Open-Source and free to use.

Terraform Commands

i) **Terraform init:** Terraform init Initialize a terraform working directory.

a) Download and install the necessary terraform provider (Aws, Azure, Gcp).
b) Set up the terraform backed.
c) Checks for any requirefd module and download them.

ii) **Terraform Plan:** Terraform plan generate a execution plan that shows what changes terraform will make before applying them.

a) Identifies resources to be created, modified, Destoyed.
b) Helps provent accidental changes before applying them.
c) Does not make actual -- just Preview.

iii) **Terraform apply**: Terraform plan applied the changes described in the execution plan to create, modify or delete infrastructure.(Ask for configuration before applying changes -- unless auto-approve is used.)

Provides: Provides are plugin which helps terraform codes to interact with clouds like Aws, Azure ..

Example:

```
provider "aws"
{
alias = "us-east-1"
region = "us-east-2"
}
```

***Terraform Script to create resources on multiple region*:**

```
provider "aws"
{
alias = ""us-east-1"
region = "us-east-1"
}

provider "aws"
{
alias = "us-west-1"
region = "us-west-1"
}

resource "aws_instance" "test-2"
```

```
{
ami = " ami-1020"
instance_type = "micro"
provider = "aws.us-east-1"
}

resource "aws_instance" "test-2"
{
ami = " ami-1020"
instance_type = "micro"
provider = "aws.us-west-1"

}
```

Terraform Script with multiple cloud provider.

```
provider "aws"
{
region = "us-east-1"
}

provider "azurerm"
{
subscription_id = "12345"
Client_id = "146"
Client_secret = "3455"
tenant_id = "456"
}

resource "aws-instance" "test-1"

{
ami = "ami-23435"
instance_type = "t2.micro"
```

```
}

resource "azurerm_virtual_machine" "test-2"
{
name = test-vm
location = "east-us"
size = standard-AI"
}
```

Variable : Terraform variable allows you to define reusable and configuration values that can be passed to module, reducing hardcoding and improving flexibility.

```
i) Input variable (Var) → Used to be pass values dynamically.
ii)Local variable (Local) → Used withing a module for better readability.
iii) Output Variable → Used to return values after execution.
```

Declaring a Variable

```
Input Variable
variable "instance-type"

{
description = "Ec2 Instance"
type = "string"
default = "t2.micro"
}
```

Using the variable

```
resource "aws_instance" "example"
{
instance-type = var.instance-type"
}
```

local variable : Used to simplify complex expressing and improve code readab ility.

Example:

```
locals
{
common_tag = {
owner = "Prabhat"
Project = "Terraform"
}
}

resource "aws_instance" "example"

{
tags = local.common_tag

}
```

Output Variable : Used to return values after terraform execution.

example:

```
output "instance_public_ip"
```

```
{
description = " Publuic IP of the Instance "
value = "aws.instance-example_public_ip"

}
```

Conditional Operator :

Conditional expression in terraform are used to define conditional logic within your configuration. They allows you to make decision or set values based on conditions.

Conditional expression are used to control weather resources are created or configured based on the evaluation of a condition.

```
Syntax :

condition?true_val : falsed_val
```

```
i) Condition :  Is an expression that evaluate to either "true" or "false".

ii) true_val : is the value that is returned if the condition is "true".

iii) falsed_val : is the value that is returened if the condition id "false".
```

Example:

```
variable "production_subnet_cidr"
{
description = "CIDR block for Prod"
type = string
default = "10.0.1.0/24"
```

```
}

variable "development_subnet_cidr"
{
description = "CIDR block for dev"
type = string
default = "10.0.1.0/24"
}

resource "aws_security_group" "example"
{
name = "example-eg"
description = "Example Security group "
}

ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_block = var.environments == "production"?[var.production_subnet_cidr] :
[var.development_subnet_cidr]
}

}
```

Terraform Module:

Module in terraform are a way to encapsulate and organize your infrastructure code making it more reusable and maintainable.
They help you create logical grouping of resources and you can use them to define reusable components that can be easily shared and managed.

What is module:

Module are containers for multiple resource that are used together.

a) A module can call other module which lets you break your infrastructure into more manageable components.

Why use module:

a) Reusability : Define a module once and use it many times.
b) Encapsulation: Hide the complexity the underlying resources.
c) Consistency: Ensure that configuration are applied consistently across your infrastructure.

Example:

```
main.tfstate

provider "aws"
{
region = "us-east-1"

}

module "ec2_instance"
{
source = "location_of_module"
}
```

Module Folder Structure.

```
|--- Variable.tf
|--- terraform.tfvar
|--- Output.tf
```

Terraform State

The state file in terraform is a crucial components that keep track of the resources managed by terraform.

What is state file.

i) State file (terraform.tfstate) is a JSON file that maps real-world resources to your configuration.
ii) It maintains the relationship between your configuration file and the actual infrastructure.

Why it's Important.

i) Tracking Resource Changes : It helps terraform understand what resource exits so it can plan updates accurately.
ii) Performance : Terraform uses the state file to improves performance by "" metadata about resource.
iii) Collaboration: It allows multiple users to work on the same infrastructure.

Managing State.

i) Backend: Use remote backends (Aws S3, Azure Blob storage and terraform cloud) to store state file securely and enable collaboration.
ii) Locking: Implement state locking to prevent concurrent operations that cloud corrupt the state file.

Provisioner: Provisioner in terraform allow you to execute script or run commands on infrastructure resource after they created or updated.

They are tipically used for task like bootstraping , configuration management and

However, their use is discouraged in some cases becauses they can make infras

Common Type of Provisioner:

1. Local-exec: Execute a command locally on the machine running terraform.

2. Remote-exec: Execute a command on a remote resource via SSH or WinRM.
3. File: Uploads files or directory to remote resource.

How to use:

```
resource "aws_instance" "example"
{
ami = "123"
instance_type = "micro"
provisioner "local-exec"

{
  command = " echo Helow world"
}
}
```

Terraform Workspace :

Terraform workspace are a feature that allows you to manage multiple instance of your infrastructure configuration within the same working directory.

Each workspace has its own state data. which mean you create and manage seperate set of resorces without interacting with each other.

Important Commands:

1. To create new workspace:
   cmd : terraform workspace new dev/stage/prod
2. To list all available workspace:
   cmd: terraform workspace list
3. To Switch between workspace:
   cmd: terraform workspace select <workspace name>

4. Delete workspace
   cmd: terraform workspace delete <workspace name>

Important Points:

i) Workspace are useful for creating parallel, distinct copies of infrastructure.
ii)Each workspace has it's own state file, which keep tracking of the resources managed by terraform.

Scenario Question:

Let's Suppose we have infrastructure provisioned with some other tools & technologies and now we want tech upgrade and want to use "terraform" for infrastructure
provisioning. How we will replicate with terraform without disturbing the current Setup.

Answer: By Using terraform import concept we can prepare terraform state file for current infrastructure and from the next time for any change or updated we can use terraform.

Scenario Question:

Let's suppose we have infrastructure provisioned with terraform and we have all setup & state file.
But some of us has made some changes on any infrastructure segment manually from console. which we are using or created by terraform and which is not shared with others.

In that case out terraform state file will be not updated and if we execute we will face multiple issue. How we can avoid this scenario.

Answer: Here we can use terraform refresh concept. Where we can schedule a "cron job" which will monitor out infrastructure and of found any changes it will notify us or update "terraform state file" accordingly. By Using the concept we can avoid.

- ---------- END ----------