Why Use Terraform as Infra-As-Code Tool?

Infrastructure as code (IaC) tools allow you to manage infrastructure with configuration files rather than through a graphical user interface. IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.

Terraform is HashiCorp's infrastructure as code tool. It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle. Using Terraform has several advantages over manually managing your infrastructure:

- Terraform can manage infrastructure on multiple cloud platforms.
- The human-readable configuration language helps you write infrastructure code quickly.
- Terraform's state allows you to track resource changes throughout your deployments.
- You can commit your configurations to version control to safely collaborate on infrastructure.

Why to Use Terraform As Infra-As-Code Tool ???

Infrastructure as code (IaC) tools allow you to manage infrastructure with configuration files rather than through a graphical user interface. IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.

Terraform is HashiCorp's infrastructure as code tool. It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle. Using Terraform has several advantages over manually managing your infrastructure:

- Terraform can manage infrastructure on multiple cloud platforms.
- The human-readable configuration language helps you write infrastructure code quickly.
- Terraform's state allows you to track resource changes throughout your deployments.
- You can commit your configurations to version control to safely collaborate on infrastructure.

```
provider "aws" {
region = "var.AWS_REGION"
shared_credentials_file = "<Your AWS Credentials File path>"
}
```

Variables File:- Terraform variables lets us customize aspects of Terraform modules without altering the module's own source code. This allows us to share modules across different Terraform configurations, reusing same data at multiple places.

When you declare variables in the root terraform module of your configuration, you can set their values using CLI options and environment variables. When you declare them in child modules, the calling module should pass values in the module block.

Versions File:- It's always best practice to maintain a version file where you specific version based on which your stack is testing and live on production.

```
terraform {
required_version = ">= 0.12"
}
```

Scenario 1:-Create 5 IAM users and a Developer group, and align all users as part of this Developer Group

Let's create a variable to type a list to pass our usernames for whom the IAM user profile needs to be created in AWS.

```
variable "usernames" {
  type = list(string)
  default = ["mawein","sakali","heavenly","marion","jerry"]
}
```

Resource

- aws_iam_user:- This resource is used to create an AWS IAM user.

Arguments

- **name**: - This is a mandatory argument to define user name as part of resource creation.
- **count:** - Variable to take the length of the user list and save it.
- **element**: - It's an intrinsic function of terraform to retrieve a single element from a list.

```
resource "aws_iam_user" "userlist" {
  count = "${length(var.username)}"
  name = "${element(var.username,count.index )}"
}
```

Resource

- aws_iam_group: - This resource is used to create an AWS IAM group.
- **Arguments**
- name: - This is a mandatory argument to define group name as part of resource creation

```
resource "aws_iam_group" "dev_group" {
  name = "Developer"
}
```

Resource

- aws_iam_user_group_membership:- This resource is used to associate AWS IAM users to single or multiple groups.

Arguments

- name: - This is a mandatory argument to define this group membership association.

- user: - This is a mandatory argument to provide a list of users to be associated with the group.
- groups: - This is a mandatory argument to provide a list of groups to be associated.
- count: - Variable to take the length of the user list and save it.
- element: - It's an intrinsic function of terraform to retrieve a single element from a list.

```
resource "aws_iam_user_group_membership" "user_group_membership" {
  count  = length(var.username)
  user   = element(var.username, count.index)
  groups = [aws_iam_group.dev_group.name, ]
}
```

Scenario 2: -Create IAM Policies and assign this to the Developer group.

Resource

aws_iam_policy: - This resource is used to create IAM policy and define JSON policy within it.

Arguments

- name: - This is an optional argument to define the IAM policy name.
- description: - This is an optional argument to provide more details about the IAM policy.
- policy: - This is a mandatory argument to JSON policy document.

```
resource "aws_iam_policy" "dev_group_policy" {
  name        = "dev-policy"
  description = "My test policy"
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
     {
```

```
      Action = [
        "ec2:Describe*",
        "ec2:Get*",
      ]
      Effect   = "Allow"
      Resource = "*"
    },
  ]
 })
}
```

Resource

- aws_iam_group_policy_attachment:- This resource is used to attach the AWS IAM policy to the group.

Arguments

- group: - This is a mandatory argument to provide the name of the group to which the policy needs to be attached.
- policy_arn: - This is a mandatory argument to provide AWS IAM policy arn which needs to be associated with the group.

```
resource "aws_iam_group_policy_attachment" "custom_policy" {
  group      = aws_iam_group.dev_group.name
  policy_arn = aws_iam_policy.dev_group_policy.arn
}
```

Output File

Output values make information about your infrastructure available on the command line, and can expose information for other Terraform configurations to use. Output values are similar to return values in programming languages.

```
output "user_arn" {
  description = "Provide the IAM user names which are created as part of this resource"
  value = aws_iam_user.userlist.*.arn
}
output "dev-group-id" {
  value       = aws_iam_group.dev_group.id
  description = "A reference to the created IAM group"
}
```

The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style

terraform fmt

Initialize the working directory by running the command below. The initialization includes installing the plugins and providers necessary to work with resources.

terraform init

Create an execution plan based on your Terraform configurations.

terraform plan

Execute the execution plan that the terraform plan command proposed.

terraform apply -auto-approve

Cross verify stack using AWS Management Console

make sure to destroy terraform stack with terraform destroy command.