TERRAFORM

Notes:

1. **Sysops** means it's only adminstarating infrastructure, **sysops** plus **CI,CD** is called **Devops**.
2. **IAC:** Provisioning infrastructure using descriptive languages is called **Infrastructure as code(IAC)**
3. **IAC** is done by **Cloudformation templates**(follows **Json/Yaml**), and Terraform.
4. **Cloudformation templates** only for **AWS.**
5. **Terraform is** also supports Microsoft azure, Google Cloud,Openstack.
6. Compared to **CT** terraform is better because it is more durable, more flexible, more reusable.
7. **Chef, ansible** is also used for **IAC,** but those two are not recommendable for **IAC.(**Not idempotent in terms of IAC**)**
8. **Terraform is** Open source.
9. Terraform files contains extension with **.tf**
10. Terraform is **idempotent** means it did't give any changes if we apply same command more times.
11. Terraform will not support if u want to execute particular file in same folder, for that we want to separate folders and apply.
12. Latest version is **0.11.5,** but mine is **Terraform v0.9.9 and 0.10.8**
13. The syntax of Terraform configurations is called **HashiCorp Configuration Language (HCL)**. It is meant to strike a balance between human readable and editable as well as being machine-friendly. For machine-friendliness, Terraform can also read **JSON** configurations.
14. **We** use descriptive languages for infrastructure provisioning
15. Terraform official site is www.terraform.io
16. Terraform aws plugins version **1.10.0**
17. **.terraform** folder contains aws plugins.
18.

Benefits of IAC:

1. Automation, we can easily provision similar environments.
2. Saves time money and efforts.
3. Code is version controlled, infrastructure related defects can be traced out and fixed easily, we also can roll back to previous versions when needed.

Tools available for IAC:

1. **Cloud formation templates**(Only works in aws), supports json and yml, but not more dynamic and less modular compared to Terraform.
2. **Terraform:** From HashiCorp supports multiple providers like aws,azure,google cloud etc… Its more dynamic, more modular and more reusable. It is simple and more readable.
3. Cheff
4. Puppet
5. Ansible
6. SDK's(Python,.net, java)

TERRAFORM:

Installation:

1. Download terraform from https://releases.hashicorp.com/terraform/0.11.5/terraform_0.11.5_windows_amd64.zip?_ga=2.191727288.1552344817.1521695899-793763513.1519731792

2. Unzip above file and set **.exe** file path in environmental variables.
3. After installation we have to give permissions to access aws console. For that two ways are there
4. Configure aws cli with credentials.

Terraform tfstate file:

- This is a json file which contains all the information about resources provisioned via terraform.
- It is completely for terraform, we do not update or delete this file.
- When you ran first time the command "*terraform apply",* it creates "*terraform.tfstate*" file.
- From next time onwards terraform updates its "*terraform.tfstate"* file.
- When you ran the command "*terraform apply"* it compares the present data with old data, if any changes present terraform creates new resources or terminate the resources, otherwise it shows already up to date.

```
"version": 3,
"terraform_version": "0.10.8",
"serial": 1,
"lineage": "67ba6e6d-b980-46f1-b5f7-4944e912031a",
"modules": [
    {
        "path": [
            "root"
        ],
        "outputs": {},
        "resources": {
```

Deleting the resource:

- When we comment the particular resource in file it will destroy particular resource but **terraform destroy** will destroy all resources.

Keywords:

1. Provider
2. Variable
3. Resource
4. Tags
5. Data source

Terraform variable types:

1. String
2. Map
3. List

Commands:

1. First we want to mention provider.
2. **terraform init** → according to provider terraform downloads plugins to initialize(aws provider version **1.11.0**.
3. **terraform apply** → To apply changes.
4. **terraform destroy** → To destroy entire project.
5. If u want to destroy particular resource in project, u should comment that resource and apply.

6. **terraform plan** → without applying if u want to know changes u can use this command. This is called **dry run**.
7. It is best practice always see **terraform plan** before u apply.
8. Variable syntax is → **"${var.region}"**
9. **terraform fmt** → it allines all code with nice format.
10. When we didn't explicitly mention any data in any resource terraform will take default values(Example vpc tenancy,ec2 security groups)
11. **vpc_id = "${aws_vpc.main.id}"** → **interpolation** example, where **main** is Logical name of vpc, **Id** is attribute.
12. Terraform supports lists.
13. Terraform compares with **terraform.tfstate** file value of file first, if it finds changes compared to previous value then it will take action on terraform apply.
14. **Name = "Subnet-${count.index + 1}"** → for tagging
15. **count="${length(aws_subnet.main.*.id)}"** → To count number of subnets
16. subnet_id=**"${element(aws_subnet.main.*.id,count.index)}"**
17. user_data=**"${file("scripts/userdata.sh")}"**

Importing a resource:

1. **terraform import resource_name.logical _name id**

Before importing u must be add resource to the configuration

Terraform remote state: by using this multiple developers can collaborate same project simultaneously. Terraform keeps state file locally but when multiple developers collaborate state file must exist in remote. It supports s3, terraform enterprise and console.

Example:

```
terraform {
  backend "s3" {
    bucket = "kanna-bucket"
    key    = "terraform.tfstate"
    region = "us-west-2"
  }
}
```

After this we want to initialize one more time by using terraform init.

Terraform Modules:

1. Modules in Terraform are self-contained packages of Terraform configurations that are managed as a group. Modules are used to create reusable components in Terraform as well as for basic code organization.

   Example:

```
module "ec2" {
  source  = "hashicorp/consul/aws"
  count = 3
  instance_type="t2.micro"
  ami = "ami-36989668"
}
```