Date

" TERRAFORM "

→ It will ~~hav~~ help create infrastructure through code instead of manually doing it on cloud providers website

→ It is used for "Infrastructure as Code" (IaC)

```
//main.tf
provider "google" {

}
resource "google_compute_instance" {

}
```

→ We we run "terraform init" all the required packages which are needed for interacting with cloud service provider

→ In a ".terraform" folder all dependencies will get installed

Date

→ terraform plan

↳ This will tell that what resources will be added, changed or destroyed.

↳ Response : Plan: 1 to add, 0 to change, 0 to destroy.

→ terraform apply

↳ This command will actually apply changes to the cloud service provider like create an EC2, VPC, or ELB.

```
// aws-ec2-instance.tf
# configure the AWS Provider
provider "aws" {
    region      = "us-east-1"
    access_key  = "xxxxx"   } will get this from aws console
    secret_key  = "xxxx"    }

# Create an EC2 instance          ↗ user defined
resource "aws_instance" "my_ec2_name" {
    ami = "ami-0767046d1677be5a"  → image id
    instance_type = "t2.micro"
}
```

Date

→ terraform destroy

↳ It will delete all the resources on AWS

→ All the secret keys are stored in "terraform.tfvars"

// terraform.tfvars → provide values

aws_secret_key = "xxx"

aws_access_key = "xxxx"

defines inputs

→ Then make a "variable.tf" → names of variable must be same.

// variable.tf

variable "aws_secret_key" {

type = string

sensitive = true

description = "AWS Secret key"

}

→ when "plan" cmd is run terraform loads the ~~terraform.tfvars~~

→ If we do any change to .tfvars then we need to run plan & apply cmd

variable "aws_access_key" {

type = string

sensitive = true

}

Page No.

Date

// main.tf

```
provider "aws" {                    → referencing variables
    access_key = var.aws_access_key
    secret_key = var.aws_secret_key
}
```

- **Terraform Variables** → useful for storing values that may change b/w environments like diff values for test & dev env.

```
variable "<your variable nam>" {
    description = "instance type t2-micro"
    type        = string
    default     = "t2.micro"
}
```

simple values variable

→ [string, number, bool, list, map, set ...]

→ collection variable

```
variable "instance_cant" {
    description = "___"
    type        = number
    default     = 2
}

variable "enable_public_ip" {
    description = "___"
    type = bool
    default = true
}
```

Page No.

variable "instance_type" { }
↳ we can also leave that empty

Date

// main.tf

```
resource "aws_instance" "ec2_example" {
    instance_typ = var.instance_type
    count        = var.instance_count
    associate_public_ip_address = var.enable_public_ip
    ami = "_____"
```

→ List variable

```
variable "user_names" {
    description = "IAM usernames"
    type        = list (string)
    default     = ["user1", "user2", "user3"]
}
```

→ map variable

```
variable "project_enviornment" {
    description = "_____"
    type        = map (string)
    default     = {
      project = "project-alpha",
      enviornment = "dev"
    }
}
```

Page No.

Date

// main.tf

```
resource "aws_instance" "ec2_example" {
    =
    tags = var.project_environment
}

resource "aws_iam_user" "example" {
    count = length(var.usernames)        ← like a for loop which iterates over array and assign all user values.
    name = var.usernames[count.index]
                ↳ looping using count.
```

→ **output variables**

↳ use to extract info about a resource created by terraform

↳ like extracting of IP of a new ec2 instance

//output.tf → also it can be written in main.tf

```
output "instance_ip" {
    value = aws_instance.example.public_ip
}
```

use the cmd `terraform output instance_ip`

>> 52.4.222.33

add sensitive = true (so that it will give a masked output)

resource "aws_instance" "ec2_example" {
count = 3 → this will have 3 IP Addr

}

output "fetched_fromaws" {
value = ["${aws_instance.ec2example.*.public_dns}"]

}

Date

→ Can also use these "output values" in the configuration files.

resource "aws_security_group_rule" "example" {

...

cidr_blocks = [output.instance_ip]

}

(using values of IP instance IP in security group

→ The terraform.tfvars variables are loaded automatically but if we want to have multiple varible files then to load those files we need to run cmd.

↳ terraform apply -var_file = new_vars.tfvars

→ Variables can also be provided values during running "plan" command. This value will have higher priority.

terraform plan -var = "instance_type = t2.micro"

Page No.

Date

- Terraform Locals

↳ Agr kisi large config file mein koi value har jaga use krni hai to we use "locals"

```
// main.tf
local {
    staging_env = ${ "staging" or "${var.enviornment}" }
}

resource "aws_vpc" "staging-vpc" {
    cidr_block = "10.5.0.0/16"
    tags = {
        Name = "${local.staging-env}-vpc-tag"
    }
```

age jake isko ~~staging-2, staging-3~~ krna hai to har jaga change nhi krna parega

- Looping

→ for each loop

↳ Only works on set or map because they contain unique values

```
variable "user_name" {
    default = ["user1", "user2", "user3"]
}
```

Date

```
resource "aws_iam_user" "example" {
    for_each = var.user_names
    name = each.value
```

→ for loops

```
resource "aws_iam_user" "example" {
    value = [ for name in var.user_name : name ]
```

• File Provisioning

↳ We can also transfer files from our local machine to remote EC2 instance using file provisioning in aws.

• Data Source

```
resource "aws_instance" "ec2_example" {
    ami = "ami- ____"
    instance_type = "t2.micro"
    tags = {
      Name = "Terraform EC2"
    }
}
```

Page No.

Date

```
data "aws_instance" "myaws_instance_data" {

    filter {
        name = "tag:Name"
        values = ["Terrabrm EC2"]
    }
    depends_on=[
        "aws_instance.ec2_example"
    ]
}

output "fetched_info_from_aws" {
    value = data.aws_instance.myawst_instance_data.
                                                public_ip
}
```

Date

- Modules

↳ It is just like functions. We write modules for different things then import them into "main.tf" file.

```
main.tf
module-1/
    main.tf
    variables.tf
    output.tf
module-2/
    main.tf
    variables.tf
terraform.tfvars
```

• each module has it's own main.tf

```
// main.tf        → calling modules
module "my_module_1" {
    source = "./module-1"        ↗ input var
    web_instance_type = "t2.large"
}                    → this will also be defined in module-1/main.tf

output "public_ip_ec2" {
    value = module.module-1.public_ip_ec2
}
```

Page No.

Date

```
module "my_web_server" {
    source = "./module-2"
}
```

- Running a Sepcific Resource

terraform apply target = aws_intance.ec2_example

terraform apply target = module.module-1.aws_instance.ec2-example