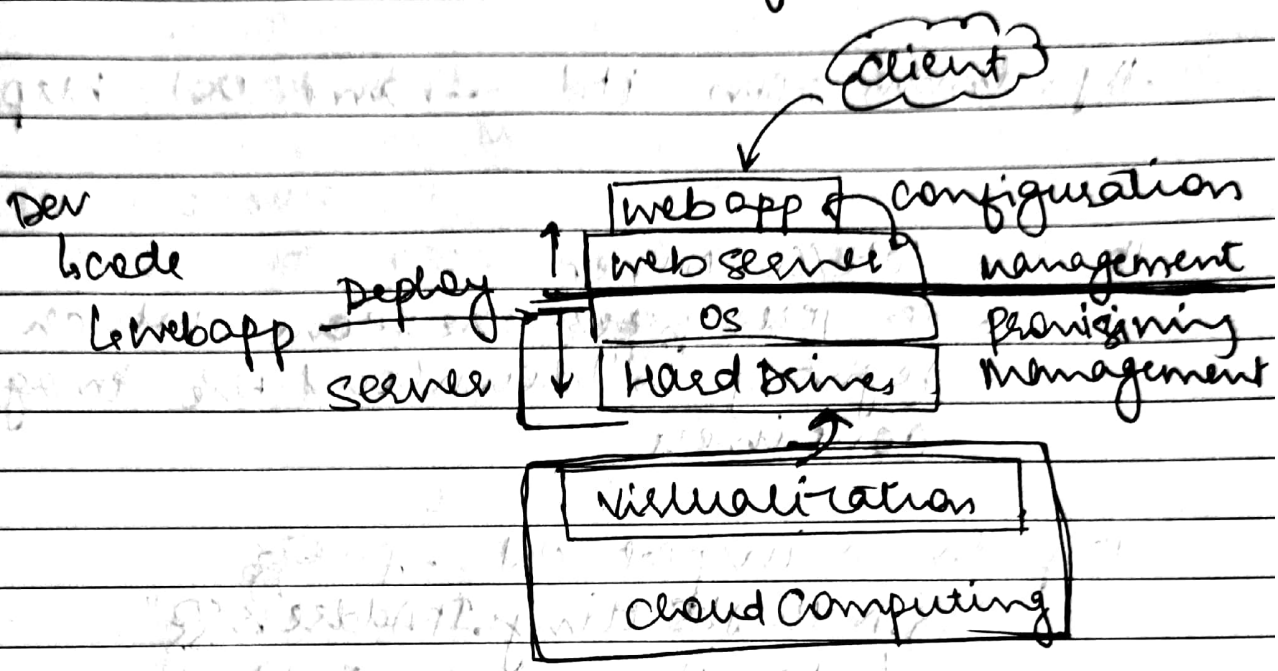
PAGE NO. :
DATE : 4 May 21

Session: 158 Terraform



- Terraform is a Provisioning Management tool.
  - Infrastructure As A Code (IAAC)

PAGE NO.:
DATE: 5May21

session: 159 Terraform

win-cmd

#terraform version
cd Documents
mkdir terraform-training-ws
mkdir basic
notepad "first.tf"

provider "aws" {}
↳ automatically downloads the aws plugin

terraform init
↳ initializes the project & downloads the plugin.

# first.tf
provider "aws" {
access_key =
secret_key =
region = "ap-south-1"
}
resource "ec2" instance" {
image id = " amazon ami "
instance type = "t2.micro"

tag to the resource

resource "aws_instance" "os1" {
ami = " ami- xxxx"
instance_type = " t2.micro"
tags = {

PAGE NO. :
DATE : / /

name = "os 4"
}
}

IAM
↳ Users
↳ Add User
↳ Username : u2
Access type : Programmatic access
↳ Attach existing policy :
Admin
>> Create

cmd
terraform init
terraform plan
↳ checks the code & gives the plan

terraform apply

session: 160 Terraform

PAGE NO.:
DATE: 6 May 21

notepad "ec2.tf"

```
variable "x" {
    type = string
    default = "linux world"
}
```

Defining a variable

```
output "myvalue" {
    value = "${var.x}"
}
```

Printing a variable

terraform apply
↳ runs all the .tf files in the current dir.

mkdir ec2
notepad "ec2storage.tf"

aws configure list-profiles
↳ myprimal

can create using create cmd

```
provider "aws" {
    region = "ap-south-1"
    profile = "myprimal"
}
```

PAGE NO. :
DATE : / /

- Resource name is a variable that stores all the information about that resource.

For eg: resource "aws_instance" "os1" {
=
}
↑ variable.

- Whenever create a new dir we need to download the plugin. (every new dir)

# output "o1" {
value = aws_instance.os1
}
↳ prints all the info of os1

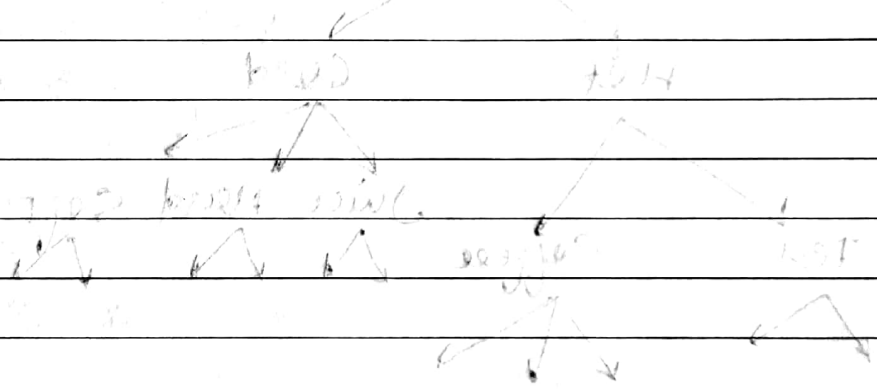
aws_instance.os1.public_ip
↳ only prints pub ip
↳ o1 = 13. — — . —

# resource "aws_ebs_volume" "st1" {
availability_zone = "us-west-2a" ✗
size = 10
tags = {
name = "world"
}
}

↓
aws_instance.os1.availability_zone

PAGE NO. :
DATE : / /

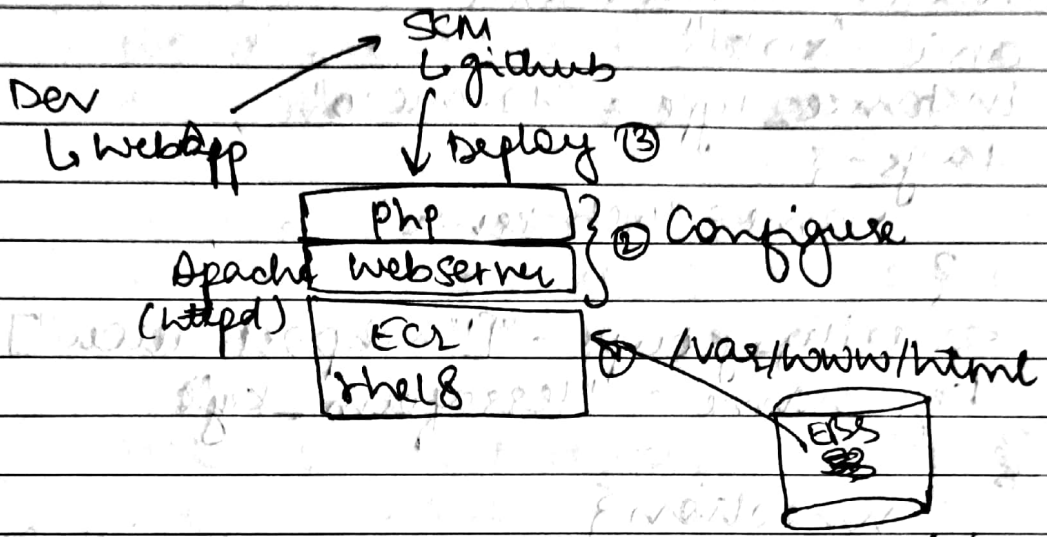
```
# output "02" {
    value = aws_ebs_volume_st1
}

# resource "aws_volume_attachment" "ebs_att" {
    device_name = "/dev/sdh"
    volume_id = aws_ebs_volume.st1.id
    instance_id = aws_instance.os1.id
    device_name = "/dev/sdh"
}

# output "03" {
    value = aws_volume_attachment.ebs_att
}
```



PAGE NO. :
DATE 11 May 21

Session: 163 Terraform

- It is an automation tool




```
# cd terraform-ws
# mkdir prov
# notepad "web.tf"
```

```
provider "aws" {
    region = ap-west-2   ap-south-1
    profile = default
}
```

```
# terraform init
```

PAGE NO.:
DATE: / /

web.tf → resource for creating a sg, key, ec2

```
resource "aws_instance" "webos1" {
  ami = "xxx"
  instance_type = "t2.micro"
  tags = {
    Name = "web server"
  }
  security_groups = ["webport-allow"]
  key_name = "terraform_key"

  connection {
    type = "ssh"
    user = "ec2-user"
    private_key = "<path>\terraform-key.pem"
    private_key = file("<path>/key")
    host = aws_instance.webos1.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo yum install httpd -y",
      "sudo yum install php -y",
      "sudo systemctl start httpd"
    ]
  }
```

path with forward slash (/)

HERE THE ROLE OF ANSIBLE COMES TO PLAY

terraform apply

PAGE NO.
DATE : / /

- Terraform works on the concept of current state & desired state.
So if ec2 is already launched it won't run the script for configuration. Here, ansible comes to play.

- To run it we can either delete the resource (pre-existed) from aws or create a new resource in .tf file.

```
resource "null_resource" "nulre1" {
  connection{
    =
  }
  provisioner{
    =
  }
}
```

terraform init
Le to download plugins for null resource.

terraform apply

web.tf

```
resource "aws_ebs_volume" "ebs" {
  availability_zone = aws_instance.webs1.availability_zone
  size = 1
  tags = {
    Name = "web server HD"
  }
}
```

PAGE NO. :
DATE : / /

```
resource "aws_volume_attachment" "ebs_att" {
  device_name = "/dev/sdc"
  volume_id = "aws_ebs_volume.ebs.id
  instance_id = aws_instance.webos1.id
}

resource "null_resource" "n2" {
  connection {
    type = "ssh"
    user = "ec2-user"
    private_key = file("mykey.pem")
    host = aws_instance.webos1.public_ip
  }
  provisioner "remote-exec" {
    inline = [
      "sudo mkfs.ext4 /dev/sdc",
      "sudo mount /dev/sdc /var/www/html",
      "sudo yum install git -y",
      "sudo git clone ____"
    ]
  }
}
```

terraform apply -auto-approve

```
resource "null_resource" {
  provisioner "local-exec" {
    command = "chrome http://<ip>/hw.html"
  }
}
```

terraform destroy ← destroys all the infra

PAGE NO.:
DATE 13May21

session: 166 Terraform

mkdir aws-wc1

- Create multiple files ; split the code for better management.

notepad ++ "a.tf"

- In one single dir we can create multiple files & when we run "apply" and they will run all the files.

"ec2.tf"

```
resource "aws_instance" "os1" {
    ===
}
```

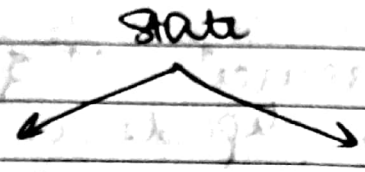
"block.tf"

```
resource "aws_ebs_volume" " " {
    ==
}
```

"attach_block.tf"

```
resource "aws_volume_attachment" " " {
    ==
}
```

PAGE NO.:
DATE: / /



(Idempotence)

- .tfstate
  - current state is stored here.
  - It is created when we run tf apply cmd

"output.tf"

```
output "myip" {
  value = aws_instance.os1.public_ip
}
```

- They retrive the values from .tfstate file

- .tfstate file is always updated when we run tf apply cmd.

terraform refresh
- it goes to the cloud and updates .tfstate file

EITHER DO 100% MANUAL OR 100% AUTOMATE IT.

PAGE NO. :
DATE : / /

"apache.tf"

```
resource "null_resource" "" {
    connect {
    }
    provisioner {
    }
}
```

depends_on = [
aws_volume-attach.ebs
]

terraform validate

Le checks the syntax of the code

PAGE NO. :
DATE 18 May 24

Session 169 Terraform

mkdir google

GCP

project new
↳ iwteraprojectl

Enable GCE API

main.tf

provider "google" {
project = "iwteraprojectl" ← projid
region = "asia-south1"
credentials = "iwteraprojectl-key.json"
}

security
IAM
↳ service Account
↳ <key xox>
↳ keys
↳ Add key
↳ Create new key
↳ JSON
> Create

#terraform init

#main.tf

```
resource "google_compute_instance" "os1" {
    name = "os1"
    machine_type = "e2-medium"
    zone = "asia-south1-c"
    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-9"
        }
    }
    network_interface {
        network = "default"
    }
}
```

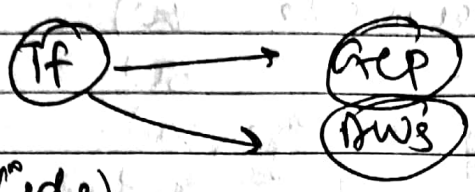
#gcloud init

Create project name

PAGE NO. :
DATE : 19/May/24

Session: 170 Terraform

Multicloud strategy



- Terraform is a framework but understands data types.

vscode ("ide")

#mkdir multi-cloud

provider.tf

```
provider "aws" {
    region  = "ap-south-1"
    profile = "default"
}
provider "google" {
    project
    region
}
```

#variables.tf

```
variable "x" {
    default = "hello"      #If we leave it empty
    type = "string"        #By default string
}

output "o1" {
    value = var.x
}
```

the value they will auto prompt

~~main.tf~~ aws_main.tf

```
resource "aws_instance" "web" {
    ami = "ami-xxxx"
    instance_type = "t2.micro"
}
```

hardcoded

dynamic = var.x

PAGE NO. :
DATE : / /

variable "x" {
default = "t2.micro"
type = string
}

terraform apply -var="x=t2.medium"

terraform.tfvars ← fixed name
↳ config file
x = "t2.large"
y = "az=1"
Only used to create key-value pairs.

-var-file="x.tfvars"
↳ If we have custom file name.

variable "y" {
type = bool
}

condition? true_value : false_value

output "os" {
value = var.y ? "linux" : "jack"
[yes/true] [no/false]
}

PAGE NO. :
DATE : / /

gcp_main.tf

```
resource "google_compute_instance" "os" {
    name = "os1"
}
```

# variable "istest" {
    type = bool
}

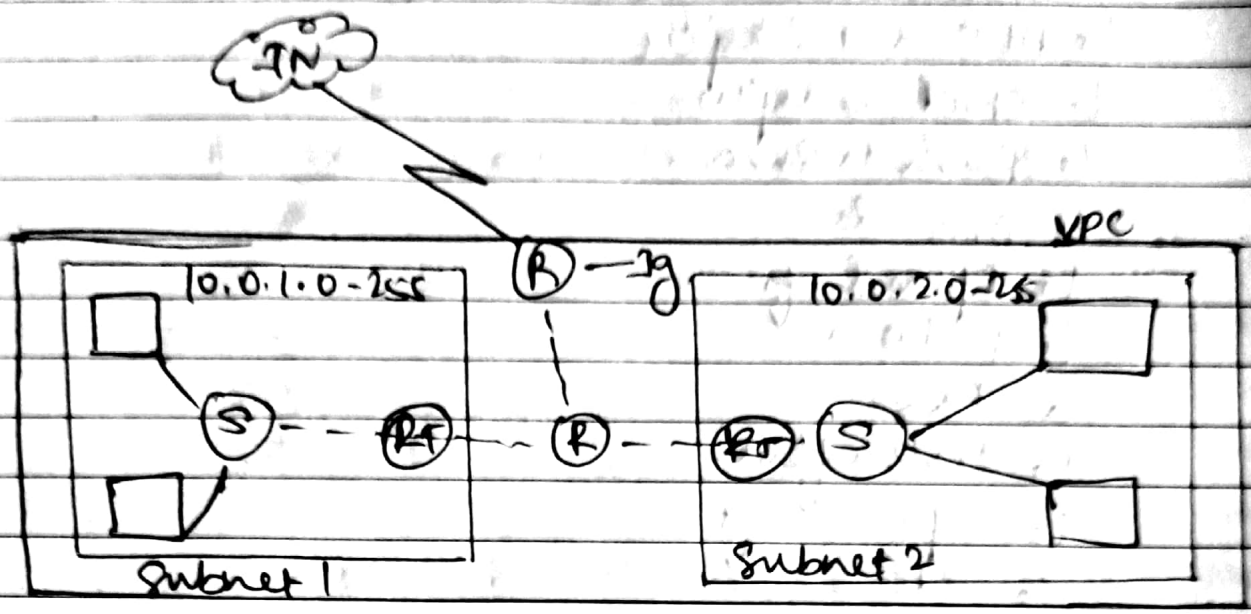
```
resource "aws_____" {
    count = var.istest ? 0 : 1
}

resource "gcp" {
    count = var.istest ? 1 : 0
}

var "abc" {
    default = ["a", "b", "c"]
    type = list
}
output "prabc" {
    value = var.abc [0]   [1]   [2]
}                   a       b     c
```

PAGE NO. :
DATE : / /

```
var "types" {
    type = map
    default = {
        us-east-1 = "t2.nano"
        ap-south-1 = "t2.micro"
        us-west-1 = "t2.medium"
    }
}

output "o3" {
    value = var.types
}                    ↳ all $ print

        or

value = var.types["us-east-1"]
                    ↳ t2.nano
```

Session: 172 Terraform

PAGE NO.:
DATE 25May21



I. create vpc
II. igw (Internet gateway)
III. Attach igw with vpc
IV. create 2 subnets
  ↳ use count=2
    but also give unique names
    ↳ ∴ use loops
V. Routing Table
  ↳ i/p: 0.0.0.0/0 to go to igw
VI. Add RT to the subnets

mkdir vpc
providers.tf

```
provider "aws" {
    region = var.aws_region
    profile = "default"
}
```

PAGE NO. :
DATE : / /

variables.tf

```
variable "aws_region" {
    default = "ap-south-1"
}

variable "vpc_cidr" {
    default = "10.0.0.0/16"
}
variable "subnets_cidr" {
    default = ["10.0.1.0/24", "10.0.2.0/24"]
    type = list
}
variable "azs" {
    default = ["ap-south-1a", "ap-south-1b"]
    type = list
}
```

vpc.tf

```
resource "aws_vpc" "main" {
    cidr_block = var.vpc_cidr
    tags = {
        Name = "myvpc"
    }
}

resource "aws_internet_gateway" "gw" {
    vpc_id = aws_vpc.main.id
    tags = {
        Name = "myigw"
    }
}
```

PAGE NO. :
DATE : / /

```
resource "aws_subnet" "main" {
    vpc_id = aws_vpc.main.id
#   cidr_block = var.subnets_cidr[0]
    tags = {
        Name = "subnet_${count.index+1}"
    }
#   availability_zone = var.azs[0]
    count = length(var.subnets_cidr)

    cidr_block = element(var.subnet_cidr,
                         count.index)
    availability_zone = element(var.azs,
                         count.index)
    map_public_ip_on_launch = true

}

resource "aws_route_table" "rt" {
    vpc_id = aws_vpc.main.id
    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.
                     igw.id

    tags = {
        Name = "public route"
    }
    }
}
```

PAGE NO. :
DATE : / /

misc functions
f-lf

```
output "01" {
    value = length(["a", "b", "c"])
}
```

```
output "02" {
    value = index(["a", "b", "c"], "b")
}
```

```
output "03" {
    value = element(["a", "b", "c"], 2)
}
```

or

```
element(["a", "b", "c"], count.index)
```

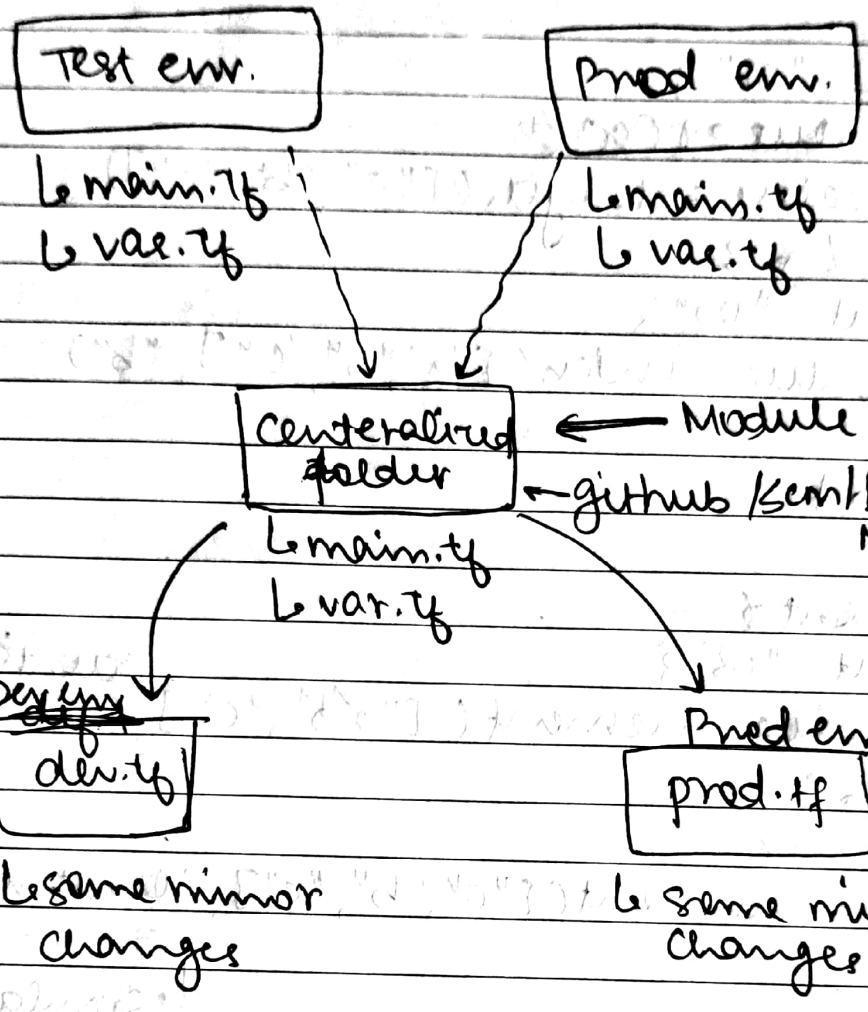
similar to k++ in long like C

```
resource "aws_route_table_association" "*" {
    count = length(var.subnets_cidr)
    subnet_id = element(aws_subnet.main.*.id, count.index)

    route_table_id = aws_route_table.rt.id
}
```

PAGE NO.:
DATE: June 21

Session: 174 Terraform

Module



- Here, the centralized folder is known as module.

```
# mkdir mym
# cd mym
# mkdir modules
# cd modules
# notepad main.tf
provider "aws" {
    region =
    profile = 2
}
```

terraform
| mym
| module, dev, prod
| ec2

PAGE NO. :
DATE : / /

resource "aws_instance" {
    type = "t2.micro"
}

var.mytype

notepad var.tf
variable "mytype" {} ← empty var.
~~default = "t2.micro"~~

terraform apply
↳ enter value (var.mytype): t2.micro

# mkdir dev
# cd dev
# notepad dev.tf
module "my_dev_module" {
    source = "../modules/ec2"
    mytype = "t2.~~micro~~ small"
}

terraform init
terraform apply

• Now, terraform has its own registry that contains various modules.

registry.terraform.io
↳ Browse Modules

PAGE NO. :
DATE : / /

mkdir /mym/vpc
cd vpc
provider "aws" {....}
module "vpc" {
source = "terraform-aws-modules/vpc/aws"
cidr = 10.0.1.0/24"
..... (from website)
}
# terraform init
# terraform apply

#terraform module source (use)

PAGE NO. :
DATE : 2 June 21

Session: 175 Terraform

- TF w/ K8S
- TF fn's.

#minikube start

#terraform console
- gives the terraform cli/console for live code testing

mkdir f
cd f
notepad e.tf

```
variable "region"{
    default= "ap-south-1"
}
variable "ami"{
    type = map
    default = {
        "us-east-1" = "ami-122333"
        "us-west-1" = "ami-23434"
        "ap-south-1" = "ami-11111"
    }
}

output "o1"{
    value = lookup( var.ami, var.region,
                    "ami-44444")
}
```

PAGE NO. :
DATE : / /

kubectl get pods
↳ No resources.

terraform
↳ registry
↳ k8s
↳ document

mkdir/cd kube
k.tf

```
provider "kubernetes" {
    config_path = "~/.kube/config"
    config_context = "minikube"
                ↳ namespace
}
```

terraform init

mkdir sg
sg.tf

```
resource "aws_security_group" "sg1" {
    name = "mysg"
    ingress {
        description = "test"
        from_port = 80
        to_port = 80
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
```

We can create this block dynamic

PAGE NO :
DATE : / /

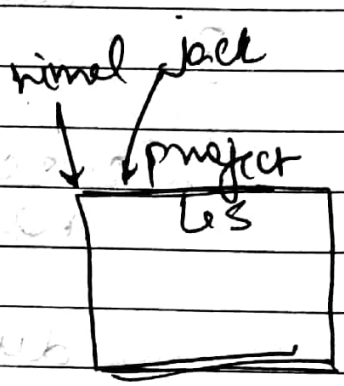
```
variable "sgports" {
    type = list
    default = [80, 81, 8080, 8081]
}

resource "aws_security_group" "sg" {
    name = "mysg"

    dynamic "ingress" {
        content {
            for_each = var.sgports
            from_port = ingress.value
            to_port = ingress.value
            protocol = "tcp"
            cidr_blocks = ["0.0.0.0/0"]
        }
    }
}
```

PAGE NO.:
DATE: 8 June 21

Session: 176 Terraform

Remote State Management

```
# mkdir pd
# s.tf
provider "aws" {
    region = "ap-south-1"
    profile = "default"
}
resource "aws_instance" {
    ==
}
```




```
# AWS
  S3
    Create bucket
      bname:
```

```
# mkdir s3
# s3.tf
provider "aws" {
    ==
}
resource "aws_s3_bucket" "b" {
    bucket = "my_tf_test_bucket"
    acl = "public" (optional for pub)
    tags = {
      Name = "tf"
    }
    versioning {
      enabled = true
    }
```

PAGE NO. :
DATE : / /

```
lifecycle {
    prevent_destroy : true
}
```
↳ won't be able to destroy the resource using terraform destroy cmd.

```
}
```

terraform init / apply

```
# cd ..
# mkdir <dir> && cd <dir>
# s.tf
provider "aws" {
    ≡
}
terraform {
    backend "s3" {
        bucket = "my-tf-test-bucket"
        key = "my.tfstate"
        region = "ap-south-1"
    }
}
```

creates a file my.tfstate

terraform state file (use)
↳ <backend>
↳ s3

```
# terraform init
# e.tf
resource "aws_instance" {
    ≡
}
```
↳ stores the state file in s3 but still not solves the remote state problem.

mkdir dy.

dy.tf → provider {} db

resource "aws_dynamodb_table" "tf" {

name = "tf-lock"

read_capacity = 5

write_capacity = 5

hash_key = "LockID" ← fixed

~~range_key =~~

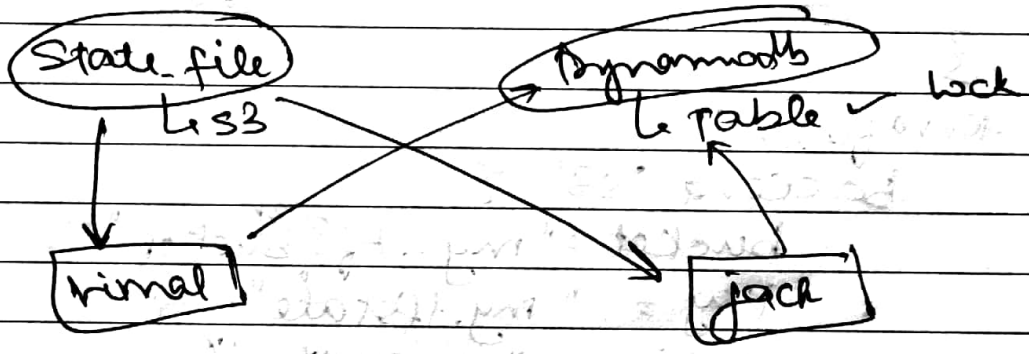
attribute {

name = "LockID"

type = "S"

}

}



PAGE NO.:
DATE: 4/June/21

Session: 17 Terraform

↳ Workspace

mkdir/cd w

# w.tf → provider "aws" {}
resource "aws_instance" {

}

variable "type" {
type = map
default = {
dev = "t2.micro",
test = "t2.small",
prod = "t2.large"
}
}

/*
=
=
*/
↑
multi line comment

- Till now, we have worked only in default workspace

terraform workspace list
↳ *default
↳ current ws.

- terraform.workspace
↳ pre-created var for displaying ws.

terraform workspace new dev
↳ creates / switches to new ws.

# w.tf
instance_type = lookup(var.type,
terraform.workspace)

PAGE NO. :
DATE : / /

terraform workspace select dev
↳ selects the dev ws.

<use> terraform regex func
↳ regex(pattern, string) #syntax
↳ replace(string, substring, replacement)