

H.S.R.A.

session: 158

Terraform

client  
http://localhost:8000

Dev

Code

Deploy

Webapp

Logs of web server

web app &amp; configuration

web server

management

OS Env

Hard Drives

provisioning management

Virtualization

Cloud Computing

• Terraform is a Provisioning Management tool.

• Infrastructure as Code (Iaas)

• Configuration Management

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

## session: 159 Terraform

win-cmd

```
# terraform version  
# cd Documents  
# mkdir terraform-training-aws  
# 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 = {}

Name = "OS 4"

3

# IAM

## ↳ Users

## Add User

Le Username: U2

Access type: Programmatic access

to attack existing policy:

Admire

• >> Create

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

# terraform apply

```
# notepad "ec2.tf"
```

variable "x" {

type = string

default = "Linux word"

}

Defining a variable

```
# notepad "output.tf"
```

output "myvalue" {

value =

("\${var.x}")

}

Printing a variable

```
# terraform apply
```

↳ runs all the .tf files in the

current directory

```
# mkdir ec2storage
```

```
# notepad "ec2storage.tf"
```

```
# aws configure list-profiles
```

↳ mynimal

using  
create cmd

```
provider "aws" {
```

region = "ap-south-1"

profile = "mynimal"

↳ mynimal profile

83

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

for ex: resource "aws\_instance" "01" {

↳ points to the object  
↳  $\{ \}$  which is nothing but variable.

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

# output "01" {

value = aws\_instance.01

↳ points all the info of 01

aws\_instance.01.public\_ip

↳ only prints pub ip

# resource "aws\_ebs\_volume" "st1" {

availability\_zone = "us-west-2a"

size = 10

tags = {

aws\_instance.01.

Name = "vol1" availability\_zone

}

{}?

# Output "02" {

    value = aws\_ebs\_volume\_stl

{

# resource "aws\_volume\_attachment" "ebs\_att"

    device\_name = "/dev/sdh"

    volume\_id = aws\_ebs\_volume\_stl.id

    instance\_id = aws\_instance.os1.id

    device\_name = "/dev/sdh"

{

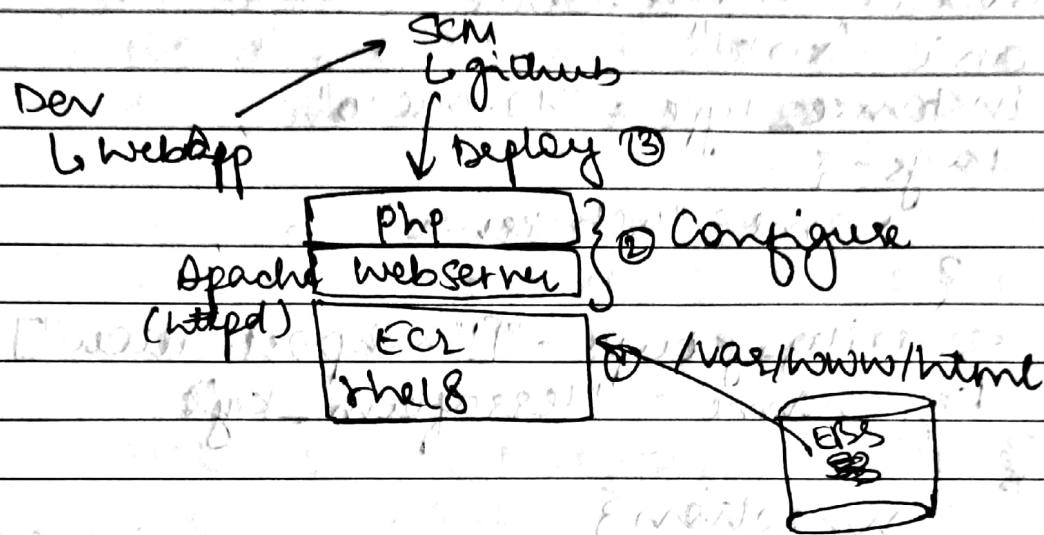
# Output "03" {

    value = aws\_volume\_attachment.ebs\_att

{

## Session: 163    Terraform

- It is an automation tool



I. partition

II. format

III. mount

```

    "partition": "3", "format": "ext4",
    "# cd /var/www/html
    # mkfs -t ext4 /dev/xvda3
    # mount /dev/xvda3 /var/www/html
    
```

provider "aws" {

region = "ap-west-1"

profile = "default"

# terraform init

# web.tf      resource for creating a sg, key pair

```
resource "aws_instance" "web01" {
  ami = "xox"
  instance_type = "t2.micro"
  tags = {
    Name = "web server"
  }
  security_groups = ["webport-allow"]
  key_name = "terraform-key"
}
```

connection {

```
  type = "ssh"
  user = "ec2-user"
  private_key = file "<path>/key"
  host = aws_instance.web01
  public_ip
```

HERE THE  
ROLE OF  
ANSIBLE  
COMES  
TO  
PLAY

provisioner "remote-exec" {

```
  inline = [
    "sudo yum install httpd -y",
    "sudo yum install php -y",
    "sudo systemctl start httpd"]
```

# terraform apply

- 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 (~~expired~~) or create a new resource in .tf file.

resource "null-resource" "nullrel" {  
connections {

—

288

2020-21 Year 18

promises

~~100~~

30-16 Metres above ground

# terraform init

To download plugins for null

# ~~Perseform~~ apply

[www.english-test.net](http://www.english-test.net)

resource "aws\_ebs\_volume" "ebs" {

Availability zone: AWS instance websites.

$\sin \theta = 1$

tags = {

Name = "Web Server HD"

resource "aws\_volume\_attachment" "webs\_att" {  
 device\_name = "/dev/xvdc"  
 volume\_id = aws\_ebs\_volume.ebs.id  
 instance\_id = aws\_instance.webos1.id}

resource "null\_resource" "n2" {  
 connection {

type = "userdata-local-exec"  
 provisioner {  
 inline = [

- "sudo mkfs.ext4 /dev/xvdc",
- "sudo mount /dev/xvdc /var/www/html"
- ]
- "sudo yum install git -y",
- "git clone https://github.com/username/repo.git"

# terraform apply -auto-approve

resource "null\_resource" {

connection { type = "user-data" }

provisioner "local-exec" {

command = "chromium http://ip:port"

# terraform destroy  $\leftarrow$  destroys all the info

## Session: 166      Terraform

```
# mkdir aws-ws
```

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

```
# notepad # "a.tf"
```

- In one single WS we can create multiple files & when we run `tf apply` and they will run all the files.

```
# "ec2.tf"
```

```
resource "aws_instance" "os1" {
```

=====

}

```
# "block.tf"
```

```
resource "aws_ebs_volume" "u" {
```

=====

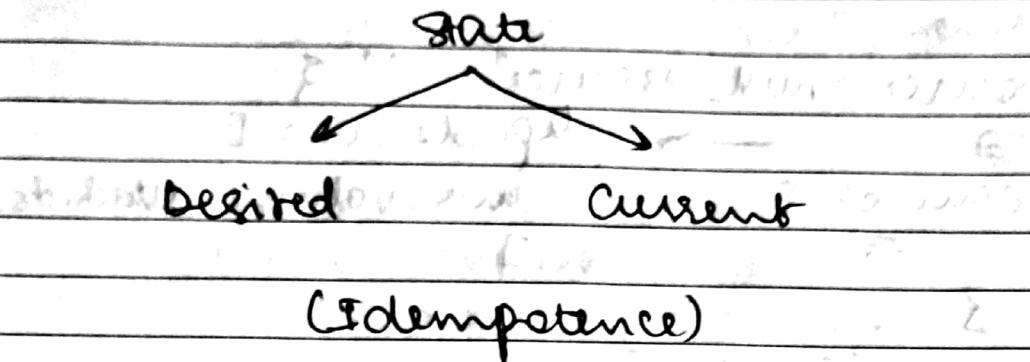
}

```
# "attach_block.tf"
```

```
resource "aws_volume_attachment" "u" {
```

=====

}



### **.tfstate**

↳ current state is stored here.

↳ It is created when we run  
`tf apply` cmd

```

# "output.tf"
output "myip" {
  value = aws_instance.osd1.public_ip
}
  
```

↳ They retrieve the values from  
.tfstate file

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

# terraform refresh

↳ it goes to the cloud and  
~~tfstate~~ updates .tfstate file

EITHER DO 100% MANUAL OR 100% AUTOMATE  
IT.

# "apache.tf"

resource "null\_resource" "aws\_volume\_attach" {

depends\_on = [

connect {

aws\_volume\_attach.ebs

= ]

3

(can't fragment)

provisioner {

= ]

# terraform validate

↳ checks the syntax of the code

↳ gives warning

↳ highlights errors in code

↳ makes code more readable

↳ finds bugs

↳ checks if resources are available

↳ finds conflicts between resources

TERMINAL OUTPUT FROM TERRAFORM

## Session 169      Terraform

~~20,000,00,00,00,00,00~~

# mkdir google

GCP

# project new

↳ ~~interap~~ project

# Enable GCE API

# main  
tf

```
provider "google" {
  project = "interaproject"
  region  = "asia-southeast1"
}
```

Credentials  
~~key~~ = "interaproject-key.json"

# security

# IAM

↳ Service Account

- ↳ <key>
- ↳ Create
- ↳ 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 = {}

initial\_params = {}

image = "debian-cloud/debian-9"

}

network\_interface = {}

nat\_ip = network\_interface["default"]

# gcloud init & set connection

Set project name to os1

gcloud config set project os1

gcloud config set compute/region asia-south1

gcloud config set compute/zone asia-south1-c

gcloud auth activate-service-account os1@os1.iam.gserviceaccount.com

gcloud auth activate-service-account os1@os1.iam.gserviceaccount.com

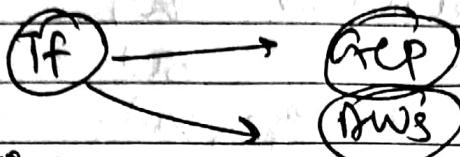
gcloud auth activate-service-account os1@os1.iam.gserviceaccount.com

gcloud auth activate-service-account os1@os1.iam.gserviceaccount.com

Session: 170

Terraform

## MultiCloud Strategy



vscode (IDE)

# mkdir multi-cloud

# provider.tf

- Terraform is a framework but understands data types.

provider "aws" {

region = "ap-south-1"

profile = "default"

}

provider "google" {

project

{

region

# variables.tf

variable "x" {

they will auto prompt

default = "hello" # If we leave it empty

type = "string" # By default string

{

output "o1" {

value = var.x

{

# ~~main~~ aws\_main.tf

resource "aws\_instance" "web" {

ami = "ami-0000"

instance\_type = "t2.micro" # hardcoded

{

instance\_type = "t2.micro" # dynamic

= var.x

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"

← solid opt

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? "win10" : "jack"

}

true      false

false

# gcp\_main.tf

resource "googlecomputeinstance" "os" {  
name = "081"

# variable "isfirst" {

type = bool

resource "aws" {

count = var.isfirst? 0 : 1

resource "gcp" {

=

count = var.isfirst? 1 : 0

}

var "abc" {

default = ["a", "b", "c"]

type = list

output "prob" {

value = var.abc [0] | 0.1 | 0.2

3 | a | b | c

was "types"?

type = map

$$\text{defaut} = \{$$

US-east-1 = ? . name

ap-south-1 = " +2. micro

Wg + West - 1 = " + 2. medium

~~Log U Log -~~

bequit 3 mit diesen mittwoch und freitag

anper "03" 3

value = var. types

3 Is all & printed

or

value = var, types ["us-east-1"]

add punctuation after Ld2.name

Position of legend

卷之三

99,000 ft. Mississippian 4000 ft.

geographies begin to move around us.

Page 10 of 10

different methods for the same

*2010-11*

at least

*1930-1931* *1931-1932* *1932-1933* *1933-1934* *1934-1935*

the original + first three sets of data, I found

11.08.00 4

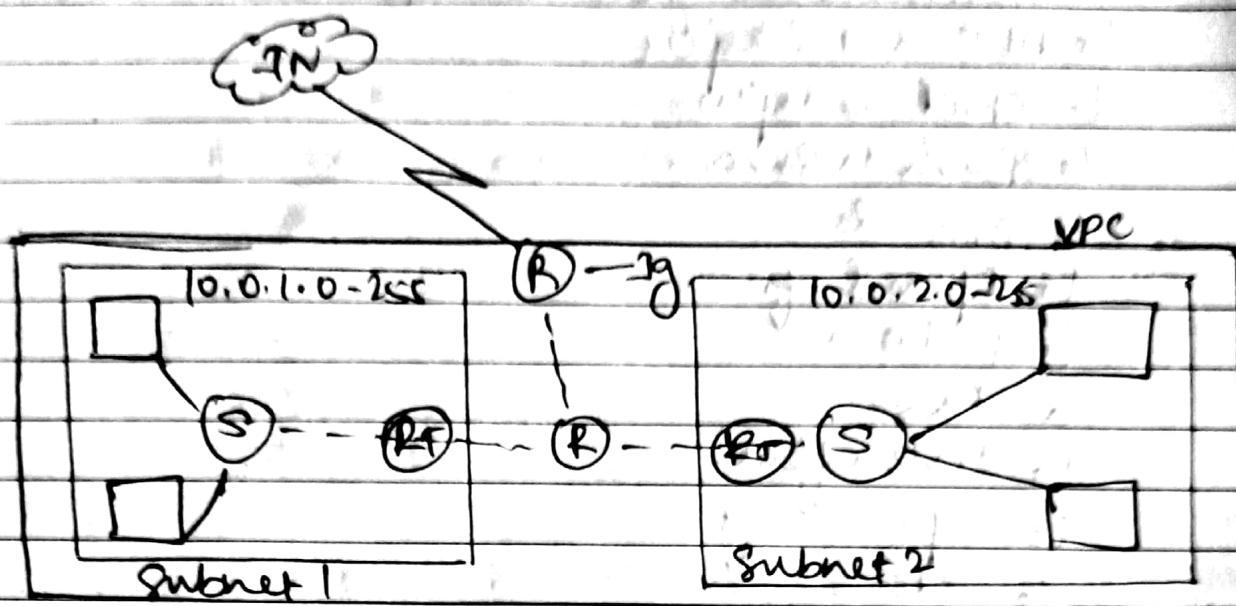
10. *Leucosia* *leucostoma* *leucostoma* *leucostoma*

083216-38 083216-38 083216-38

Figure 1. The relationship between the number of species and the number of individuals in each species.

Session 172

Terraform



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

With: 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"

3

```

# variable's tfvars file
variable "aws_region" {
  default = "ap-south-1"
}

variable "vpc_cids" {
  default = ["10.0.0.0/16"]
}

variable "subnet_cids" {
  default = ["10.0.1.0/24", "10.0.2.0/24"]
  type = list
}

variable "vrs" {
  default = ["ap-south-1a", "ap-south-1b"]
  type = list
}

# vpc.tf file
resource "aws_vpc" "main" {
  cidr_block = var.vpc_cids
  tags = {
    Name = "myvpc"
  }
}

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

```

resource "aws\_subnet" "main" {

  vpc\_id = aws\_vpc.main.id

  #cidr-block = var.subnets.cidr[0]

  tags = []

  name = "subnet-\$count.index"

}

  #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"

}

}

  #aws\_route\_table\_association {

    route\_table\_id = rt.id

    subnet\_id = var.subnets.id

# midie functions

# f-if

output "01" {

value = length

value = length(["a", "b", "c"])

{ L + 3 }

output "02" {

value = index(["a", "b", "c"], "b")

{ }

element

output "03" {

value = element(["a", "b", "c"], 2)

{ L C . body }

or

value = element(["a", "b", "c"], count\_index)

similar to

remove if self &amp; first not eq k+im

. python lang like C

bring with support

resource "aws\_route\_table\_association": "r" {

Count = length(var. subnets\_id)

subnet\_id = element(~~aws~~ aws\_subnet.

main.\*.id, count\_index)

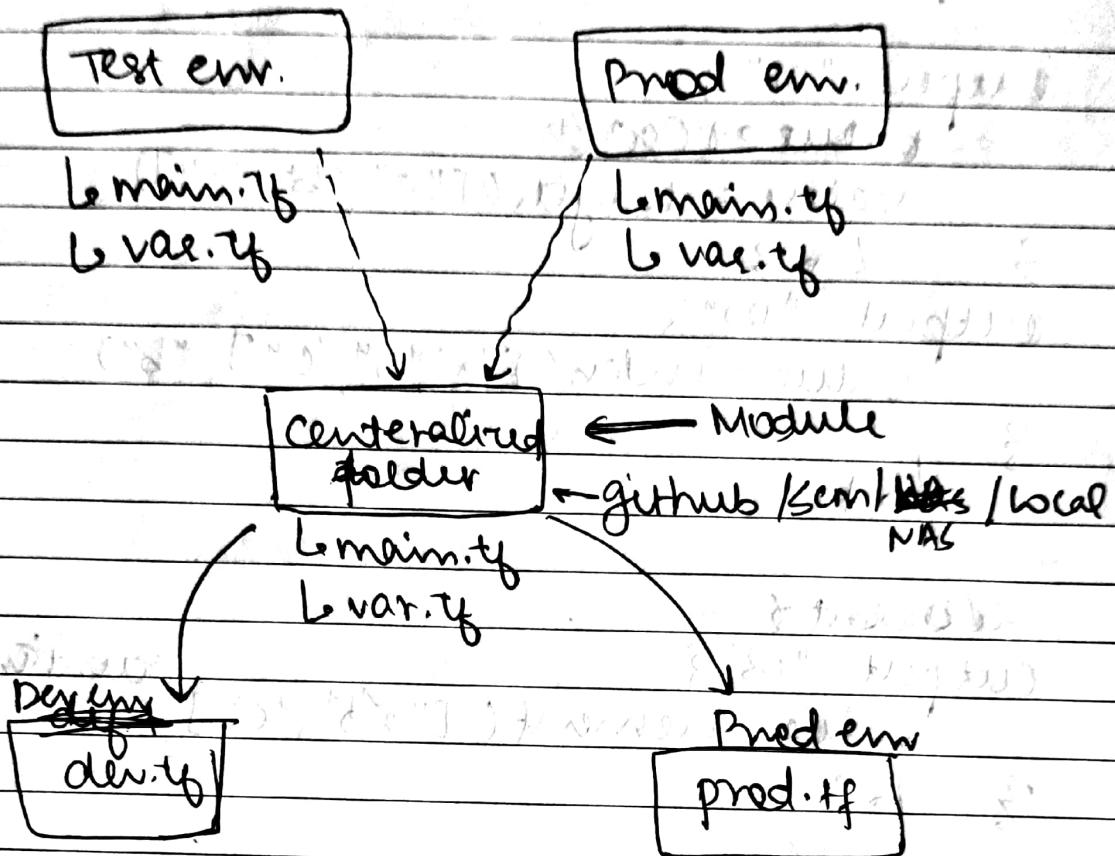
route\_table\_id = aws\_route\_table\_id

{ }

Session : 174

Terraform

Module



↳ some minor changes ↳ some minor changes

• Here, the centralized folder is known as module.

terraform

```

# mkdir mym | mym module
# cd mym | mym provider
# mkdir modules | mym module dev/prod
# cd modules | mym module ec2
# notepad main.tf
provider "aws"
  region = ?
  profile = ?
  
```

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"

# terraform init  
# terraform apply

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

# registry.terraform.io  
↳ Browse Modules

# mkdir /myml/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)

Session: 175 Terraform

↳ TF w/ K8s

↳ TF fns.

# minikube start

# terraform console

↳ gives the terraform cli/console 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", default = "ami-44444")

}

Output o1

Output o1

Output o1

Output o1

# kubectl get pods  
↳ NO resources.

# terraform  
↳ registry  
↳ k8s  
↳ document

# mkdir/cd kube  
# kubectl

provides "kubernetes" g.  
config-path = "k/.kube/config"  
config-context = "my account" → minikube  
} → namespace

# terraform init

# mkdir sg  
# sg.tf  
resource "aws\_security\_group" "sg1" {  
name = "mysg"  
}

we can create this block dynamic  
↳ ingress {  
description = "test"  
from\_port = 80  
to\_port = 80  
protocol = "tcp"  
cidr\_blocks = ["0.0.0.0/0"]  
}

variable "sgports" {

-type = list

default = [80, 81, 8080, 8081]

}

resource "aws\_security\_group" "s1" {

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"]

}

{ }

}

-> "optional sections" are added

-> "ingress default port" is added

by adding "port" in "port" : 80

"port" = 80

"port" = 80

"port" = 80

Session: 176

Terraform

## Remote State Management

# mkdir s

# S.TF

provider "aws" {

region = "ap-south-1"

profile = "default"

{}

resource "aws\_instance" {

=

{}

# AWS

less

Create bucket

Lab name:

# mkdir s3

# s3.tf

provider "aws" {

=

{}

resource "aws\_s3\_bucket" "b" {

bucket = "my\_tf\_laravel\_bucket"

acl = "public" (Encryption for pub)

tags = {

Name = "tf"

{}

versioning {

enabled = true

{}

remote state  
project

list
------

lifecycle {

  prevent\_destroy : true

}

↳ won't be able to destroy the resource  
using terraform destroy cmd.

}

\* terraform init / apply

# Cd ..

# mkdir / cd ss

\* s.tf

provider "aws" {

=

}

terrafrom {

  backend "s3" {

    bucket = "my-tf-bucket" <sup>just</sup>

    key = "my.tfstate"

    region = "ap-south-1" ↓

}

}

creates a  
file my.tfstate

\* terraform init

\* e.tf

resource "aws\_instance" {

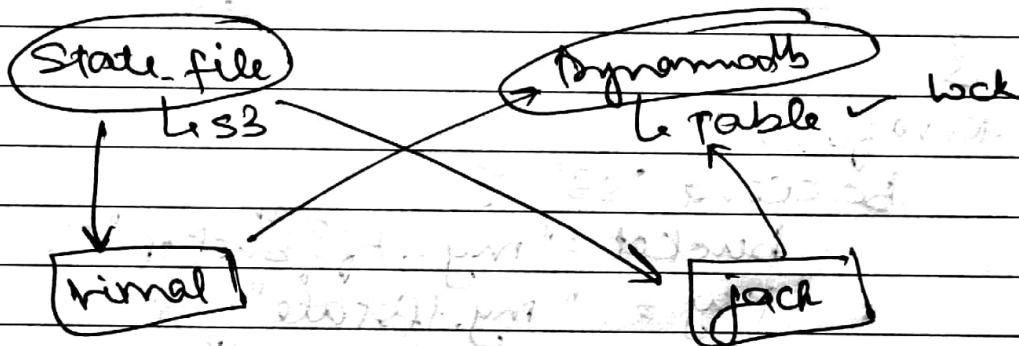
=

}

↳ stores the state file in s3 but  
still not solves the remote state  
problem.

## mkdir dy

```
# dy.tf provider { db
  resource "aws_dynamodb_table" "lock" {
    name = "tf-lock"
    read_capacity = 5
    write_capacity = 5
    hash_key = "LockID" ← fixed
    range_key =
    attribute {
      name = "LockID"
      type = "S"
    }
  }
}
```



lock file will store lock details of  
state changes and version lock file

Session: 17

Terraform

## ↳ workspace

# mkdir/cd w

# w.tf → provider "aws" of "resource" "aws instance" of

{ } instance-type = lookup var.type

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,  
terrafrom.workspace)

\* Terraform workspace select dev  
↳ Selected the dev ws.

\* <use> terraform regex, func

↳ `l regex(pattern, string)` \*Syntax

↳ `l replace(string, substring, replacement)`

↳ `l join(list, separator)`

↳ `l split(string, separator)`

↳ `l sort(list) - last`

↳ `l sort(list) - first`

↳ `l join(list, separator)`

↳ `l sort(list)`

↳ `l join(list, separator)`

↳ `l sort(list)`

↳ `l join(list, separator)`

↳ `l join(list, separator)`

↳ `l join(list, separator)`

↳ `l join(list, separator)`

↳ `l join(list, separator)`