**TERRAFORM**

DAY-01: 30-10-2023 (INTRO, SETUP, FILES, VARIABLES, TARGET)

TERRAFORM:

INFRA: RESOURCES USED TO RUN OUR APPLICATION ON CLOUD.

EX: EC2, VPC, ALB

If we create infra manually

1. Time consume

2. Mistakes

3. Tracking

Infra creation -- > Automate -- > Terraform

Terraform:

its a free and opensource too1.

its also called as Infra as a code(IAAC) too1.

it is used to automate the infra creation.

its platfrom independet.

year: 2014

language: GO lang

who: Mitchel hasimoto

Owned: Hashicorp

HOW TO WORK:

code (hcl) -- > execute -- > infra

in terraform we use Hashicorp Configuration Language syntax to write the code.

once we write the code we can reuse it for infra creation.

we can resue the configuration files multiple times.

ADVATNAGES:

1. Time saving

2. Automate

3. Resource Tracking

4. Reusable

5. easy mainatinace

6. can create multiple resources

CFT : AWS

ARM : AZURE

GDE : GOOGLE

TERRAFORM: AWS, AZURE, GCP, -----

SETUP:

apt update -y

apt install awscli -y

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

terraform -v

mkdir terraform

cd terraform

vim main.tf

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "one" {

ami = "ami-03eb6185d756497f8"

instance\_type = "t2.micro"

}

Commands:

terraform init : to download provider plugins for resource craetion

terraform plan : to create execution plan

terraform apply : to create resource by terraform

terraform destroy: to delete resource

+ : Creating

- : Deleting

~ : Update

state file:

terrform state file is used to store resource current state information.

it will contain end to end info of our resource.

its very important file in terraform so we need to keep it safe and secure.

if we lost that file we cant track the infra.

terraform state list : to show total resources inside state file

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "one" {

count = 5

ami = "ami-03eb6185d756497f8"

instance\_type = "t2.micro"

}

terraform apply --auto-approve

terraform destroy --auto-approve

target: used to delete a specific resource

Single target: terraform destroy -target=aws\_instance.one[0]

Multi target: terraform destroy -target=aws\_instance.one[1] -target=aws\_instance.one[2]

variable:

its a block in yterraform used to define the variables

why -- > when the values change frequenlty we use vars

vim main.tf

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "one" {

count = var.instance\_count

ami = var.ami\_id

instance\_type = var.instance\_type

}

variable "instance\_type" {

description = "\*"

type = string

default = "t2.large"

}

variable "instance\_count" {

description = "\*"

type = number

default = 5

}

variable "ami\_id" {

description = "\*"

type = string

default = "ami-0dbc3d7bc646e8516"

}

terraform apply --auto-approve

terraform destroy --auto-approve

Variable.tf files:

In terraform all tyhe varibales we can store on seperatefiles.

it makes opur work easy to avoid confusion.

vim main.tf

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "one" {

count = var.instance\_count

ami = var.ami\_id

instance\_type = var.instance\_type

}

vim variable.tf

variable "instance\_type" {

description = "\*"

type = string

default = "t2.large"

}

variable "instance\_count" {

description = "\*"

type = number

default = 5

}

variable "ami\_id" {

description = "\*"

type = string

default = "ami-0dbc3d7bc646e8516"

}

terraform apply --auto-approve

terraform destroy --auto-approve

HISTORY:

1 apt update -y

2 wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/has hicorp-archive-keyring.gpg

3 echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.ha shicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

4 sudo apt update && sudo apt install terraform

5 terraform -v

6 aws configure

7 curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

8 unzip awscliv2.zip

9 sudo ./aws/install

10 apt install unzip -y

11 unzip awscliv2.zip

12 sudo ./aws/install

13 /usr/local/bin/aws --version

14 aws configure

15 ls -al

16 cd .aws/

17 ll

18 vim config

19 cd

20 ls -al

21 mkdir terraform

22 cd terraform/

23 vim main.tf

24 terraform init

25 ll

26 cat .terraform.lock.hcl

27 ll

28 terraform plan

29 terraform apply

30 ll

31 cat terraform.tfstate

32 terraform state list

33 cat main.tf

34 terraform destroy

35 vim main.tf

36 terraform plan

37 terraform apply --auto-approve

38 terraform state list

39 cat main.tf

40 terraform destroy --auto-approve

41 terraform state list

42 terraform apply --auto-approve

43 terraform state list

44 terraform destroy --auto-approve -target=aws\_instance.one[0]

45 terraform state list

46 terraform destroy --auto-approve -target=aws\_instance.one[1] -target=aws\_instance.one[4]

47 terraform destroy --auto-approve

48 vim main.tf

49 terraform apply

50 terraform destroy

51 vim main.tf

52 vim variable.tf

53 cat main.tf

54 cat variable.tf

55 terraform apply --auto-approve

56 terraform state list

57 terraform destroy --auto-approve

====================================================

DAY-02: 31-10-2023

TFVARS, CLI, OUTPUTS, IMPORT, RESOURCES, TAINT

TERRAFORM TFVARS:

we use tfvar files when we have multiple conifurations.

each configuration we can write on variable file and attach it while running.

cat main.tf

provider "aws" {

}

resource "aws\_instance" "one" {

ami = var.ami\_id

instance\_type = var.instance\_type

tags = {

Name = var.instance\_name

}

}

cat variable.tf

variable "ami\_id" {

}

variable "instance\_type" {

}

variable "instance\_name" {

}

cat dev.tfvars

ami\_id = "ami-03caf91bb3d81b843"

instance\_type = "t2.micro"

instance\_name = "dev-server"

cat test.tfvars

ami\_id = "ami-0a1aabd978d1ea0a5"

instance\_type = "t2.medium"

instance\_name = "test-server"

terraform apply --auto-approve -var-file="dev.tfvars"

terraform destroy --auto-approve -var-file="dev.tfvars"

terraform apply --auto-approve -var-file="test.tfvars"

terraform destroy --auto-approve -var-file="test.tfvars"

TERRAFORM CLI: used to pass the variables values from cli during runtime.

provider "aws" {

}

resource "aws\_instance" "one" {

ami = "ami-0a1aabd978d1ea0a5"

instance\_type = var.instance\_type

tags = {

Name = "raham"

}

}

variable "instance\_type" {

}

terraform apply --auto-approve -var="instance\_type=t2.medium"

terraform destroy --auto-approve -var="instance\_type=t2.medium"

TERRAFORM OUTPUT:

this block is used to print the resource outputs.

ex: public ip, dns, ebs -------

provider "aws" {

}

resource "aws\_instance" "one" {

ami = "ami-0a1aabd978d1ea0a5"

instance\_type = "t2.micro"

tags = {

Name = "raham"

}

}

output "abc" {

value = [aws\_instance.one.public\_ip, aws\_instance.one.private\_ip, aws\_instance.one.public\_dns]

}

TERRAFORM IMPORT: used to track the resource which is created outside of terraform.

Ex:if we create a server using terraform we can track information from terraform beacuase here server is created by using terraform whereas if the server is created manually and we cant track the information because not created and managed by terraform but here we can use **“TERRAFORM IMPORT”** by using this command we can track the and store the information even though if we can create a server manually.

cat main.tf

provider "aws" {

}

resource "aws\_instance" "one" {

}

Command : #terraform import aws\_instance.one i-02366c0cb62440bbb[instance-id]

ex:2full resources

EC2

provider "aws" {

}

resource "aws\_instance" "one" {

ami = "ami-0a1aabd978d1ea0a5"

instance\_type = "t2.micro"

tags = {

Name = "raham"

}

}

S3 BUCKET CREATING

resource "aws\_s3\_bucket" "two" {

bucket = "sudheer887766prodbucket"

}

resource "aws\_ebs\_volume" "three" {

size = 20

availability\_zone = "ap-southeast-1a"

tags = {

Name = "raham-ebs"

}

}

IAM CREATION

resource "aws\_iam\_user" "four" {

name = "rahams"

}

resource "aws\_iam\_group" "five" {

name = "devops"

}

TAINT: it is used to recreate specific objects.

when: in real time some resource might have chances to crash

son in that insted of creating that resource manually we can use taint command

terraform state list

terraform taint aws\_instance.one

terraform apply --auto-approve

HISTORY:

1 wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyring s/hashicorp-archive-keyring.gpg

2 echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releas es.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

3 sudo apt update && sudo apt install terraform

4 apt install awscli -y

5 aws configure

6 mkdir terrafor

7 cd terrafor/

8 vim main.tf

9 vim variable.tf

10 cat main.tf

11 vim dev.tfvars

12 vim test.tfvars

13 cat main.tf

14 cat variable.tf

15 cat dev.tfvars

16 cat test.tfvars

17 terraform init

18 terraform plan

19 terraform plan -var-file="dev.tfvars"

20 terraform apply --auto-approve -var-file="dev.tfvars"

21 terraform state list

22 terraform destroy --auto-approve -var-file="dev.tfvars"

23 cd

24 vim .aws/credentials

25 cd terrafor/

26 terraform apply --auto-approve -var-file="dev.tfvars"

27 terraform destroy --auto-approve -var-file="dev.tfvars"

28 terraform apply --auto-approve -var-file="test.tfvars"

29 terraform destroy --auto-approve -var-file="test.tfvars"

30 rm -rf dev.tfvars test.tfvars variable.tf

31 vim main.tf

32 cat main.tf

33 terraform apply --auto-approve -var="instance\_type=t2.medium"

34 terraform destroy --auto-approve -var="instance\_type=t2.medium"

35 terraform apply --auto-approve

36 terraform destroy --auto-approve

37 vim main.tf

38 terraform apply --auto-approve

39 vim main.tf

40 terraform apply --auto-approve

41 vim main.tf

42 terraform apply --auto-approve

43 terraform destroy --auto-approve

44 vim main.tf

45 cat terraform.tfstate

46 terraform state list

47 cat main.tf

48 terraform import aws\_instance.one i-02366c0cb62440bbb

49 terraform state list

50 cat terraform.tfstate

51 cat -n terraform.tfstate

52 terraform destroy --auto-approve

53 vim main.tf

54 terraform apply --auto-approve

55 terraform state list

56 cat main.tf

57 terraform state list

58 terraform apply --auto-approve

59 terraform state list

60 terraform taint aws\_instance.one

61 terraform apply --auto-approve

62 terraform state list

63 terraform taint aws\_s3\_bucket.two

64 terraform taint aws\_iam\_user.four

65 terraform taint aws\_iam\_group.five

66 terraform untaint aws\_iam\_group.five

67 terraform apply --auto-approve

68 history

==========================================================

DAY-03: 1-11-2023

LOCALS, WORKSPACES, GRAPH AND BACKEND SETUP

TERRAFORM LOCALS:

its a blcok used to define values

once we define a value we can use it multipple times.

if you change value on local block all the values will be replaced to all block.

provider "aws" {

}

locals {

env = "prod"

}

resource "aws\_vpc" "one" {

cidr\_block = "10.0.0.0/16"

tags = {

Name = "${local.env}-vpc"

}

}

resource "aws\_subnet" "two" {

vpc\_id = aws\_vpc.one.id

cidr\_block = "10.0.0.0/24"

availability\_zone = "ap-southeast-2a"

tags = {

Name = "${local.env}-subnet"

}

}

resource "aws\_instance" "three" {

subnet\_id = aws\_subnet.two.id

ami = "ami-0cb441cf7bb9cba22"

instance\_type = "t2.micro"

tags = {

Name = "${local.env}-server"

}

}

TERRAFORM WORKSPACE:

Workspace: its a place where we write our code.

It is used to isloate the resources.

all the commands will work on workspace only.

without workspace we cant run commands and we cant create resources.

in real time we create diff workspaces for diff envs.

each env is isloated with another env with help of workspaces.

no need to write multiple config files, just chane the values

NOTE:

WE CANT DELETE CURRENT WORKSPACE.

BEFOR DELETING WORKSPACE WE NEED TO DELETE RESOURCES ON IT.

WE CANT DELETE DAFAULT WORKSPACE

COMMANDS:

terraform workspace list : to show list of workspace

terraform workspace new dev : to Create and switch to workspace "dev"

terraform workspace show : to show current workspace

terraform workspace select test : to switch blw workspaces

terraform workspace delete test : to delete the workspaces

provider "aws" {

}

locals {

env = "${terraform.workspace}"

}

resource "aws\_vpc" "one" {

cidr\_block = "10.0.0.0/16"

tags = {

Name = "${local.env}-vpc"

}

}

resource "aws\_subnet" "two" {

vpc\_id = aws\_vpc.one.id

cidr\_block = "10.0.0.0/24"

availability\_zone = "ap-southeast-2c"

tags = {

Name = "${local.env}-subnet"

}

}

resource "aws\_instance" "three" {

subnet\_id = aws\_subnet.two.id

ami = "ami-0cb441cf7bb9cba22"

instance\_type = "t2.large"

tags = {

Name = "${local.env}-server"

}

}

Terraform fmt: used to set the indentation for terraform files.

Terraform Graph: to show the blue frint of infra.

terraform graph : copy the code and past it on

graphwiz online

TERRAFORM BACKEND SETUP:

in real time state file we need to keep backup and secure on remote laction.

for that purpose we need to use s3.

the state file will be store on s3 for saftey.

if state file we lost on local we can retrive from s3

Create a bucket manually.

provider "aws" {

}

terraform {

backend "s3" {

bucket = "rahamterraoprodbucket9988"

key = "prod/terraform.tfstate"

region = "ap-southeast-2"

}

}

locals {

env = terraform.workspace

}

resource "aws\_vpc" "one" {

cidr\_block = "10.0.0.0/16"

tags = {

Name = "${local.env}-vpc"

}

}

resource "aws\_subnet" "two" {

vpc\_id = aws\_vpc.one.id

cidr\_block = "10.0.0.0/24"

availability\_zone = "ap-southeast-2c"

tags = {

Name = "${local.env}-subnet"

}

}

resource "aws\_instance" "three" {

subnet\_id = aws\_subnet.two.id

ami = "ami-0cb441cf7bb9cba22"

instance\_type = "t2.large"

tags = {

Name = "${local.env}-server"

}

}

resource "aws\_ebs\_volume" "three" {

size = 20

availability\_zone = "ap-southeast-2a"

tags = {

Name = "raham-ebs"

}

}

resource "aws\_iam\_user" "four" {

name = "rahams"

}

resource "aws\_iam\_group" "five" {

name = "devops"

}

HISTORY:

69 terraform destroy --auto-approve

70 vim main.tf

71 cd

72 vim .aws/config

73 cd terrafor/

74 terraform apply --auto-approve

75 terraform destroy --auto-approve

76 cd terrafor/

77 ll

78 vim main.tf

79 terraform init

80 terraform plan

81 terraform plan -out

82 terraform plan -help

83 terraform apply --auto-approve

84 vim main.tf

85 terraform apply --auto-approve

86 terraform state list

87 vim main.tf

88 terraform apply --auto-approve

89 vim main.tf

90 terraform apply --auto-approve

91 terraform destroy --auto-approve

92 vim main.tf

93 terraform workspace list

94 terraform workspace new dev

95 vim main.tf

96 terraform apply --auto-approve

97 terraform state list

98 terraform workspace new test

99 terraform workspace list

100 terraform workspace show

101 vim main.tf

102 terraform apply --auto-approve

103 terraform workspace new prod

104 vim main.tf

105 terraform apply --auto-approve

106 terraform workspace delete prod

107 terraform workspace select test

108 terraform workspace show

109 terraform workspace delete prod

110 terraform workspace select prod

111 terraform state list

112 terraform destroy --auto-approve

113 terraform workspace select test

114 terraform workspace delete prod

115 terraform workspace list

116 terraform destroy --auto-approve

117 terraform workspace select dev

118 terraform workspace delete test

119 terraform graph

120 terraform destroy --auto-approve

121 terraform workspace select default

122 terraform workspace delete dev

123 terraform workspace list

124 terraform workspace delete default

125 cat main.tf

126 terraform fmt

127 cat main.tf

128 terraform workspace list

129 terraform state list

130 vim main.tf

131 terraform init

132 terraform apply --auto-approve

133 ll

134 cat -n terraform.tfstate

135 cat -n terraform.tfstate.

136 cat -n terraform.tfstate.backup

137 ll

138 vim main.tf

139 terraform fmt

140 terraform apply --auto-approve

141 vim main.tf

142 terraform apply --auto-approve

143 cat -n terraform.tfstate.backup

144 terraform destroy --auto-approve

145 cat -n terraform.tfstate.backup

146 cat terraform.tfstate

147 cat main.tf

148 history

==========================

LAST 2 DAYS

provider "github" {

token = "\*"

}

resource "github\_repository" "example" {

name = "example"

description = "My awesome codebase"

visibility = "public"

}

resource "aws\_iam\_user" "two" {

name = "rahamshaik0099"

}

resource "aws\_iam\_group" "three" {

name = "devopsteam"

}

resource "aws\_instance" "four" {

ami = "ami-01eccbf80522b562b"

instance\_type = "t2.large"

tags = {

Name = "terra-server"

}

}

terraform apply --auto-approve

terraform state list

terraform taint aws\_instance.four

terraform apply --auto-approve

resource "local\_file" "one" {

filename = "abc.txt"

content = "hai all my name is raham shaik"

}

provider "github" {

token = "\*"

}

resource "github\_repository" "example" {

name = "rahamshaikterrarepo"

description = "My awesome codebase"

visibility = "public"

}

VERSION CONSTARTINGS:

provider "local" {

}

terraform {

required\_providers {

local = {

source = "hashicorp/local"

version = "<2.2.0"

}

}

}

============================================================

LIFECYCYLE:

1. PREVENT\_DESTROY: it wont destroy the resource even if we run destroy command

provider "local" {

}

resource "aws\_instance" "three" {

ami = "ami-01eccbf80522b562b"

instance\_type = "t2.large"

tags = {

Name = "abc-server"

}

lifecycle {

prevent\_destroy = false

}

}

2. IGNORE CHANGES: when we cahnge some componets of resources terraform will ignore it.

3. DEPENDS ON: here resouce creation will depends on another resource.

provider "local" {

}

resource "aws\_instance" "three" {

ami = "ami-01eccbf80522b562b"

instance\_type = "t2.large"

availability\_zone = "us-east-1a"

tags = {

Name = "abc-server"

}

}

resource "aws\_ebs\_volume" "two" {

size = 10

availability\_zone = "us-east-1a"

tags = {

Name = "abc-volume"

}

depends\_on = [aws\_instance.three]

}

for\_each loop:

provider "local" {

}

resource "aws\_instance" "three" {

for\_each = var.instance\_type

ami = "ami-01eccbf80522b562b"

instance\_type = each.key

tags = {

Name = "abc-server"

}

}

variable "instance\_type" {

description = "\*"

type = set(string)

default = ["t2.micro", "t2.medium", "t2.large"]

}

provider "aws" {

}

resource "aws\_instance" "one" {

count = length(var.instance\_type)

ami = "ami-06006e8b065b5bd46"

instance\_type = var.instance\_type[count.index]

tags = {

Name = var.instance\_name[count.index]

}

}

variable "instance\_type" {

default = ["t2.medium", "t2.micro", "t2.nano"]

}

variable "instance\_name" {

default = ["webserver", "appserver", "dbserver"]

}

DYNAMIC BLOCK: it is used to reduce the length of code and used for reusabilty of code in loop.

provider "aws" {

}

locals {

ingress\_rules = [{

port = 443

description = "Ingress rules for port 443"

},

{

port = 80

description = "Ingree rules for port 80"

},

{

port = 8080

description = "Ingree rules for port 8080"

}]

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-0c02fb55956c7d316"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.main.id]

tags = {

Name = "Terraform EC2"

}

}

resource "aws\_security\_group" "main" {

egress = [

{

cidr\_blocks = ["0.0.0.0/0"]

description = "\*"

from\_port = 0

ipv6\_cidr\_blocks = []

prefix\_list\_ids = []

protocol = "-1"

security\_groups = []

self = false

to\_port = 0

}]

dynamic "ingress" {

for\_each = local.ingress\_rules

content {

description = "\*"

from\_port = ingress.value.port

to\_port = ingress.value.port

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

tags = {

Name = "terra sg"

}

}

LIST:

provider "aws" {

}

resource "aws\_iam\_user" "one" {

count = length(var.user\_name)

name = var.user\_name[count.index]

}

variable "user\_name" {

description = ""

type = list(string)

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

}

MAP:

provider "aws" {

}

resource "aws\_instance" "three" {

ami = "ami-01eccbf80522b562b"

instance\_type = "t2.large"

availability\_zone = "us-east-1a"

tags = var.ec2\_tags

}

variable "ec2\_tags" {

description = ""

type = map(string)

default = {

env = "prod"

client = "swiggy"

Name = "rahamserver"

}

}

Note: Practise is optinal

MODULES: Used to divide the terrafrom code to multiple folders.

it makes our work easy and flexiblue.

it can be reusable

cat main.tf

module "my\_instance\_module" {

source = "./modules/instances"

ami = "ami-04823729c75214919"

instance\_type = "t2.micro"

instance\_name = " rahaminstance"

}

module "s3\_module" {

source = "./modules/buckets"

bucket\_name = "devopsherahamshaik009988"

}

cat provider.tf

provider "aws" {

}

cat modules/instances/main.tf

resource "aws\_instance" "my\_instance" {

ami = var.ami

instance\_type = var.instance\_type

tags = {

Name = var.instance\_name

}

}

cat modules/instances/variable.tf

variable "ami" {

type = string

}

variable "instance\_type" {

type = string

}

variable "instance\_name" {

description = "Value of the Name tag for the EC2 instance"

type = string

}

cat modules/buckets/main.tf

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

bucket = var.bucket\_name

}

cat modules/buckets/variable.tf

variable "bucket\_name" {

type = string

}

HISTORY:

last 2 days

1 sudo yum install -y yum-utils shadow-utils

2 sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

3 sudo yum -y install terraform

4 aws configure

5 ls -al

6 aws configure

7 mkdir terraform

8 cd terraform/

9 vi main.tf

10 terraform init

11 terraform validate

12 vim main.tf

13 terraform validate

14 vim main.tf

15 terraform fmt

16 terraform plan

17 terraform apply --auto-approve

18 terraform destroy --auto-approve

19 vim main.tf

20 terraform destroy --auto-approve

21 vim main.tf

22 terraform apply --auto-approve

23 terraform destroy --auto-approve

24 vim /root/.aws/credentials

25 terraform apply --auto-approve

26 terraform destroy --auto-approve

27 vim main.tf

28 terraform apply --auto-approve

29 vim main.tf

30 terraform apply --auto-approve

31 terraform destroy --auto-approve

32 vim main.tf

33 terraform apply --auto-approve

34 vim main.tf

35 terraform apply --auto-approve

36 vim main.tf

37 terraform apply --auto-approve

38 vim main.tf

39 terraform apply --auto-approve

40 terraform destroy --auto-approve

41 vim main.tf

42 terraform apply --auto-approve

43 │ terraform init -upgrade

44 terraform init -upgrade

45 terraform apply --auto-approve

46 ll

47 cat abc.txt

48 terraform destroy --auto-approve

49 terraform init

50 vim main.tf

51 terraform init -upgrade

52 vim main.tf

53 terraform init -upgrade

54 vim main.tf

55 terraform init -upgrade

56 vim main.tf

57 terraform init -upgrade

58 vim main.tf

59 terraform init -upgrade

60 vim main.tf

61 terraform init -upgrade

62 vim main.tf

63 terraform init -upgrade

64 terraform apply --auto-approve

65 terraform destroy --auto-approve

66 cat main.tf

67 cd terraform/

68 ll

69 rm -rf raham.tf

70 terraform apply --auto-approve

71 cat /root/.aws/config

72 vim main.tf

73 cd terraform/

74 vim main.tf

75 terraform apply --auto-approve

76 terraform destroy --auto-approve

77 vim main.tf

78 terraform apply --auto-approve

79 terraform destroy --auto-approve

80 vim main.tf

81 terraform apply --auto-approve

82 terraform destroy --auto-approve

83 vim main.tf

84 terraform apply --auto-approve

85 vim main.tf

86 terraform apply --auto-approve

87 vim main.tf

88 terraform apply --auto-approve

89 terraform destroy --auto-approve

90 vim jenkins.sh

91 sh jenkins.sh

92 cat /var/lib/jenkins/secrets/initialAdminPassword

93 cat /root/.aws/credentials

94 cd /var/lib/jenkins/workspace/

95 cd terraformpipeline

96 ll

97 terraform state list

98 vim main.tf

99 cd -

100 cd ..

101 cd /root/

102 cd terraform/

103 vim main.tf

104 terraform apply --auto-approve

105 terraform destroy --auto-approve

106 vim main.tf

107 terraform apply --auto-approve

108 vim main.tf

109 terraform apply --auto-approve

110 vim main.tf

111 terraform apply --auto-approve

112 history

====================

ALIAS AND PROVIDER:

10-11-2023:

provider "aws" {

region = "ap-southeast-1"

}

resource "aws\_instance" "one" {

ami = "ami-05caa5aa0186b660f"

instance\_type = "t2.micro"

}

provider "aws" {

region = "ap-south-1"

alias = "south"

}

resource "aws\_instance" "two" {

provider = aws.south

ami = "ami-02e94b011299ef128"

instance\_type = "t2.micro"

}