Terraform da kuracağımız yapı moduler bir yapı olmalıdır. Bu sayede şirket içindeki çalışanlar ihtiyacına göre değişiklikler yaparak terraformda oluşturduğumuz infrastructure ı kullanırlar.

Yine uygulama geliştirirken development environment için oluşturduğumuz infrastructure ı daha sonra ufak değişikliklerle test environment ında da kullanılır.

Bu nedenle herkesin ortak kullanacapı bir ana module oluşturuyoruz. Diğer birimler için ayrı birer modul oluyor ve o modulden bunu çalıştırıyorlar.

Modules 
aws 
resource "aws_instance" "L=" 
ami = ami-123456789 
instance_type = var . instance _type 
security groups = ["tf-sec-grp"] 
Source 
instance_type 
Development 
Testing 
Production 
t3a.medium 

Variables 
ami 
instance—type 
key_name 
tags 
= var.ec2_ami 
= var. instance-type 
= var.key_name 
= var. tags 
Modules 
Amazon EQ 

Handsona geçiyoruz

Önce şu yapıyı oluşturmamız lazım:

txt 
terraform-modules 
dev 
I e_ dev-vpc. tf 
modules 
I outputs .tf 
variables . tf 
L— prod 
L_ prod-vpc.tf 

Her bir branch için ayrı ayrı vpc oluşturmaya kalksak onlarca tf dosyası takip etmemiz gerekecek. Biz içinde sadece vpc olan bir module oluşturacaz. Daha sonra DEV branch ı da PROD branch ı da gelip bu module u kullanıp kendileri için vpc ayağa kaldırabilecekler.

Main.tf e bu kod bloğunu yapıştırıyoruz:

Bu kod bloğu bir vpc oluşturmak için bir blok. yine dokümantasyona gidip bakıyoruz vpc nasıl oluşturuluyor diye.

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc>

provider "aws" {

region = "us-east-1"

}

resource "aws\_vpc" "module\_vpc" {

cidr\_block = var.vpc\_cidr\_block

tags = {

Name = "terraform-vpc-${var.environment}"

}

}

vpc oluşturma dökümandan:

Basic usage: 
"aws_vpc" "main" { 
resource 
cidr block 
"la.e.a.e/16" 

vpc nin cidr bloğunu variable olarak veriyoruz.

resource "aws\_subnet" "public\_subnet" {

cidr\_block = var.public\_subnet\_cidr

vpc\_id = aws\_vpc.module\_vpc.id

tags = {

Name = "terraform-public-subnet-${var.environment}"

}

}

resource "aws\_subnet" "private\_subnet" {

cidr\_block = var.private\_subnet\_cidr

vpc\_id = aws\_vpc.module\_vpc.id

tags = {

Name = "terraform-private-subnet-${var.environment}"

}

}

şirketin neye ihtiyacı olduğunu manuel olarak bilecez ve buna göre tf dosyasını oluşturacaz.

variable lar tanımlanmış. bunları da variable.tf içine yapıştırdık.

variable "environment" {

default = "clarusway"

}

variable "vpc\_cidr\_block" {

default = "10.0.0.0/16"

description = "this is our vpc cidr block"

}

variable "public\_subnet\_cidr" {

default = "10.0.1.0/24"

description = "this is our public subnet cidr block"

}

variable "private\_subnet\_cidr" {

default = "10.0.2.0/24"

description = "this is our private subnet cidr block"

}

Output larımız da şunlar:

output "vpc\_id" {

value = aws\_vpc.module\_vpc.id

}

output "vpc\_cidr" {

value = aws\_vpc.module\_vpc.cidr\_block

}

output "public\_subnet\_cidr" {

value = aws\_subnet.public\_subnet.cidr\_block

}

output "private\_subnet\_cidr" {

value = aws\_subnet.private\_subnet.cidr\_block

}

şimdi biz module klasörü içinde terraform init / terraform apply dersek bu vpc oluşur. ancak biz böyle değil module ile çalıştırmak istiyoruz. Bu nedenle develeoperlar ın vpc ye ihtiyacı olduğu senaryoyu uygulayalım. Ya developerlar sıfırdan bir vpc oluşturacaklar ya da bizim oluşturduğumuz module u kullanarak kendileri için gerekn vpc yi ayağa kaldıracaklar.

burada oluşturacağımız bloğun adı module dür. module bloğunda source yani kaynak belirtiyoruz. bunu lokalden path ile gösterebileceğiimiz gibi remote tan da çekebiliriz.

şimdi dev klasörünün altında dev.tf dosyası oluşturuyoruz ve module bloğunu bu dosyaya yapıştırıyoruz:

module "tf-vpc" {

source = "../modules"

}

source olarak modules klasörümüzün path ini girdik. dev klasöründeyken terraform init / terraform plan komutlarını giriyoruz:

id 
instance tenancy 
ipv6 ass6ciation id 
+ ipv6¯cidr bloc 
ipv6:cidrZbloc _ 
+ main route table id 
plan : 3 
= (kncwn at' 
= •default" 
= (known af' 
— (knowl af 
(known af 
= (known af 
= (known af 
m•meF id 
tags 
tags _ atl 
to add. 
"Name" 
"Name" = •terraform-vpc-clarusway" 
to 
change, to destroy. 

vpc, public subnet ve private subnet olmak üzere üç tane kaynak oluşturacağını görüyoruz. vpc ismi clarusway olarak geldi. Neden? Çünkü variable larda environment variable ında biz bunu beöyle default olarak belirledik:

terraform-modules > modules > variables.tf > 
1 
2 
variable "environment" { 
default 
"clarusway" 

biz dev environment i için dev isminin yazmasını istiyoruz. bu nedenle module bloğunda environment vairable ını "DEV" olarak belirtiyoruz:

module "tf-vpc" {

source = "../modules"

environment = "DEV"

}

takrar terraform plan yapalım:

vpc isminin değiştiiğini gördük. Çünkü environment ın default u clarusway ken biz DEV olarak belirttik.

bu kez terraform apply diyoruz.

plan: 3 to add, O to change, O to destroy. 
Do you want to perform these actions? 
Terraform will perform the actions described above. 
Only 'yes' witl be accepted to approve. 
Enter a value: yes 
module. tf -VPC. : Creating... 
module. tf-vpc . nodule_vpc: Creation cmptete aftel 
module. Creating... 
module. tf-vpc. public_subnet: Creating... 
nodule. tf-vpc. private_subnet: Creation compte* 
module. tf-vpc .ws_subnet. pubtic_subnet: Creation co.tet. 
Apply complete! Resources: 3 added, O changed, e 
1 

output çıkmadı, demek ki module bloğu module klasöründeki output u çekmiyor. o nedenle module bloğunun altında output belirlememiz gerekiyor:

module "tf-vpc" {

source = "../modules"

environment = "DEV"

}

output "vpc-cidr-block" {

value = module.tf-vpc.vpc\_cidr

}

output u module.tf-vpc ile çekiyoruz. sonunda ise vpc\_cidr yazdığımızda vipc\_cidr ını çekecek output olarak.

şimdi terraform apply dediğimizde outputun da geldiğini göreceğiz:

Outputs : 
vpc-cidr-block = 
"10.0.0.0/16' 

şimdi production için bir vpc ihtiyacı var. PROD klasörüne gidip orada da module bloğunu yazacaz:

module "tf-vpc" {

source = "../modules"

environment = "PROD"

}

burada environmentımızı PROD olarak belirledik. Örneğin public private subnet leri de değiştrebiliriz:

wrnodules prod > prodA'pctf > module •tf%'øc 
nodule •tt-vpc" { 
source = "../mcdules" 
environment = "PROD" 
vpc_cidr block = 
29.e.e.e,'16" 
public subnet_cidr "2e.e.1.0/24" 
private subnet cidr 
"20.0.2.0/24" 

PROD yeni bir klasör olduğu için terraform init yapmamız gerkiyor. Sonrasında terraform plan diyoruz:

nodule. tf -VPC. 
resource •aws_vpc" "nodule_vpc" { 
cidr block 
• defailt network act id 
be 
= (knm•m after apply) 
= (known after apply) 

cidr bloğunun module deki gibi değil, bizim yaptığımız overwrite gibi olduğunu gördük.

module ayarlamadaki trick ; değiştirilebilecek her şeyi variable olarak yazmaktır.

klasörlerimizde terraform destroy diyebiliriz.

ana module de output ları neden yazıyoruz? örneğin child-küçük module ile oluşturduğumuz vpc de sec grp oluşturmak istiyoruz. vpc.id ye ihtiyacımız var, id yi nasıl çekecek?

resource "aus security_group• •testsg• { 
vpc_id module.tf-vpc.vpc_id 

output 
value 
"vpc_id" { 
aws_vpc . module_vpc . 

output tan çekecek.

Hazır module ler de çekip kullanabiliyoruz:

Modules 
O policy Libraries @ Run Tasks 
Modules 
Modules are self-contained packages of Terraform configurations that are managed as a group. 
terraform-aws-modules / vpc 
Terraform module Whi reates VPC reso urces on AWS 
O a month ago 
d 
cloudposse / label 
Terraform Module to define a consistent naming convention by (namespace, stage, name, (attributes)) 
0 2years ago 
35.4M 
provider 
null provider 
terraform-aws-modules / iam 

ClaruswayLLC... O Instances EC... 
modules}vpc-endpoints 
. editorconfig 
. gitignore 
. pre-comm it-config .yaml 
.releaserc.json 
CHANGELOG.md 
D LICENSE 
README , md 
UPGRADE-3.O.md 
variables.tf 
versions.tf 
vpc-flmv-bgs.tf 
O GitHub docker Docker Hub 
fix; Allow' security_group_ids to take null 
Ici skip) Create editorconfig". 
Online Course... 
values (#825) 
chore: update dcwumentation based on latest terraform-aocs Which 
fix: use a version for to avoid GitHub API rate Omiting On CI 
chore: Update release configuration tiles to correctly use convention . 
chore(release): version 3.19.0 [skip cil 
chore: update CI/CD to use stable terraform-docs release artifact 
feat: Add public and private tags per az (#860) 
refactor remove existng vpc endpoint configurations from base modul... 
feat: Add public and private tags per az (#860) 
feat: Add 'PAM IPva support (#716) 
feat: Add public and private tags per az (#860) 
feat: Add IPAM IPv4 support (#716) 
feat Added ability to specify cloudwatch Log group name for VPc Flo. 
SDLC 
Excalidraw 
6 months ago 
3 years ago 
2 years ago 
2 months ago 
last year 
2 months ago 
2 years ago 
2 months ago 
2 years ago 
2 months ago 
6 months ago 
2 months ago 
6 months ago 
5 months ago 

1229 
1229 
1231 
1232 
1233 
1234 
1235 
1236 
1237 
1238 
1239 
1240 
1241 
1242 
1243 
1244 
1245 
1246 
1247 
1248 
1249 
1251 
1252 
1253 
elemen t ( 
concat( 
this( • . id, 
. this( • I . 
count index, 
Defaults 
resource "this" 
count = var 7 1 : e 
enab dns_support 
enable_classic Link 
tags : 
var. 
: Var 
= null https://g1thub.ccWhashicorp/terraform/issues/31730 
•Name" coalesce(var. 
var . tags, 
var . 
"default") ) , 

AWS burada her ihtiyacı olan kullansın diye VPC nin bütün özelliklerini koymuş 1250 satır. bize ne lazım olacak diye ayıklamaya çalışırken kendi ihtiyacımız olan argumanlarla kendimiz main.tf i oluşturuyoruz zaten.

Şimdi Terraform import konusuna geliyoruz:

Hali hazırda çalışan resource ları, örneğin bir instance ı state dosyamızadahil etmek istiyorsak terraform import ları kullanıyoruz. Çok tavsiye edilen bir kullanım değildir ancak bazen gerekebilir.

TERRAF-ORM iM?CRT 
linux ec2 
ubuntu 
red hat 
sg 
linux ec2 
ubuntu 
sg 

Şimdi hali hazırda çalışan bir environment imiz olsun diye iki instance ve bu iki instance ın kullanacağı bir sec grp oluşturacaz:

tf-import-sg isminde 22 ve 80 portları açık olan bir sec grp oluşturacaz.

bir linux-2 bir de ubuntu instance ayağa kaldırıyoruz.

Oluşturduğumuz bu instance ları ve sec grp laı terraform state dosyamıza ekleyerek terraform ile yöneteceğiz.

şimdi vs code terraform-import klasörü oluşturup bu klasörün içinde main.tf oluşturacaz.

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "4.58.0"

}

}

}

provider "aws" {

region = "us-east-1"

}

variable "tf-ami" {

type = list(string)

default = ["ami-005f9685cb30f234b", "ami-0557a15b87f6559cf", "ami-0c9978668f8d55984"]

}

ami ları variable olarak tanımlamış olduk, bunu yapmak için de liste içine aldık.

variable "tf-tags" {

type = list(string)

default = ["aws-linux-2", "ubuntu-22.04", "red-hat-linux-8"]

}

tag verirken de yine listeyle yazdık nedeni aşağıda:

resource "aws\_instance" "tf-instances" {

ami = element(var.tf-ami, count.index )

instance\_type = "t2.micro"

count = 3

# key\_name = "clarusway" // change here

security\_groups = ["tf-import-sg"]

tags = {

Name = element(var.tf-tags, count.index )

}

}

resource "aws\_security\_group" "tf-sg" {

name = "tf-import-sg"

description = "terraform import security group"

tags = {

Name = "tf-import-sg"

}

ingress {

from\_port = 80

protocol = "tcp"

to\_port = 80

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 22

protocol = "tcp"

to\_port = 22

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

protocol = -1

to\_port = 0

cidr\_blocks = ["0.0.0.0/0"]

}

}

bu main.tf için terraform plan dediğimizde 4 resource oluşturacak:

vpc_id 
= (known after apply) 
plan: 4 to add, to change, to destroy. 

ancak biz oluşturmak istemiyoruz. biz import etmek istiyoruz. import ettiğimiz de main.tf değişmez sadece state dosyası değişir.

önce instance kısmını comment leyelim ve sec grp özelinde çalışalım.

sec grp u nasıl import edecez? sec grp dokümantasyona gittik ve en altta import kısmına geldik.

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group#import>

$ terraform import aws\_security\_group.elb\_sg sg-903004f8

bu kodla yapılacağı yazıyor.

sadeleştirelim: terraform import aws\_secuirty\_group.tf-sg sg-0b4a1cfc22a545c9e

komutu girelim.

.tf-sg: Importing ID 
.tf-sg: Import prepared! 
Prepared aws security_group for import 
.tf-sg: Refreshing state 
Import successful ! 
"sg-Øb4a1cfc22a545c9e" ... 
[id=sg-Øb4a1cfc22a545c9e] 
The resources that were imported are shown above. These resources are now in 
your Terraform state and will henceforth be managed by Terraform. 
ec2-user@ip-172-31-59-57 : N/terraform- import 

şimdi sec grp state dosyamıza geldi. Instance lar için de aynı şeyi yapacağız fakat instance ları import ederken sec grp bire aws deki haliyle bir aynı olmak zorunda, eğer aynı olmazsa state dosyamızı değiştirecek ve aws deki sec grubumuzu destroy edecek çünkü bu sec grp artık terraform kontrolünde. bu bütün resource lar için de geçerlidir.

Şimdi instance ları import edelim.

Önce Linux instance ı import edelim:

terraform import "aws\_instance.tf-instances[0]" i-090291cc33c16504c

id yi değiştirmemiz gerekiyor. import ettik:

" resources" : 
mode" • "managed" , 
"type" : 
"aws instance" 
": "tf-instances", 
name 
"provider": "provider[\"registry. terraform. io/hashicorp/aws\"]", 
"instances" : 
"index_key": 0, 
"schema version" 
"attributes " : 
"ami": "ami-ØØ5f9685cb30f234b", 
"arn"• "arn:aws : ec2 :us-east-l : 995714465210: instance/i-Ø45ee32962e3b5cc6" , 

tfstate in değiştiğini gördük.

burada [0] ile neden index verdik?

resource "aws\_instance" "tf-instances" {

ami = element(var.tf-ami, count.index )

instance\_type = "t2.micro"

count = 3

key\_name = "first-key" // change here

security\_groups = ["tf-import-sg"]

tags = {

Name = element(var.tf-tags, count.index )

}

}

çünkü ami leri ve tagleri liste içinde verdik, count olarak da 3 verdik. Dolayısıyla linux-2 index sıralamasına göre 0. instance

element nedir ona bakalım: <https://developer.hashicorp.com/terraform/language/functions/element>

element retrieves a single element from a list. 
element(list, index) 
copy [tv 

bir listeden tek bir eleman çekmek için kullanılıyormuş. terraform console da deneyelim:

terraform console 
> •ubuntu-22.04•, •red-hat-linux-8") 
-red-hat-linux-8" 
> •ubuntu-22.04% 

ubuntu ise 1. sırada, ubuntu yu import edelim:

terraform import "aws\_instance.tf-instances[1]" i-0c0fd85d4edb70bcc

" index_key" • 
"schema version" 
"attributes " : 
"ami": "ami-Ø557a15b87f6559cf", 
arn"• "arn:aws : ec2 : us-east-I : 995714465210: instance/i-øcØfd85d4edb7øbcc" , 
"associate_public_ip_address" . 
true, 
"availability_zone": "us-east -1b" 
"capacity _ reservation specification" . 
"capacity _ reservation_preference" . 
"capacity _ reservation_target" • 
"open" 

tfstate te gördük.

şimdi terraform plan dersek ne olacak?

volume 
id 
volume 
slze 
volume_type 
Plan: 1 to add, 
2 to change, 
(known a 
(known a 
(known a 
to destroy. 

değişiklikler tagler. ancak yeni bir resource ekleyecek neden çünkü resource kısmında 3 instance vardı, 3. instance ı kaldıracak.

eğer key name gibi önemli bir argüman farklı olursa apply dediğimizde instance ları terminate eder. o nedenle dikkatli olmamız gerekir.

volume 
volume 
slze 
volume_type 
Known 
(known 
(known 
2 to destroy. 
after 
after 
after 
apply ) 
apply) 
apply) 
Plan: 3 to add, 
to change, 

key pair ismini farklı girdik ve 2 destroy dedi. bu nedenle dikkatli olmamız gerekir.

key pair i düzeltip, terraform destroy diyebiliriz.