Terraform da önceden hazırlanmış module leri kullanabileceğimiz gibi biz de module leri publish edebiliriz ( yayınlayabiliriz).

Module yayınlarka github üzerinden yayınlanıyor, terraform oradan çekiyor module u.

<https://registry.terraform.io/browse/modules>

Herkes module publish edebilir terraformda:

<https://developer.hashicorp.com/terraform/registry/modules/publish>

Module dosyaları GitHub da public repoda olmalı.

Repo isminde bu kısım olmak zorunda : terraform-<PROVIDER>-<NAME>

Repositorynin bir açıklaması olmalı.

Standart module yapısında olmalı (main.tf, variables.tf, outputs.tf)

x.y.z tags for releases

Yamalar en sonra, küçük değişiklikler ortada ve büyük değişiklikler en başta olacak şekilde semantik versiyonlama ile versiyon gösteriliyor.

Şimdi home ya da lesson klasöründe clarusway-modules klasörü oluşturup, içerisinde main.tf variables.tf outputs.tf versions.tf userdata.sh README.md .gitignore

Dosyalarını oluşturuyoruz.

Şimdi version.tf dosyamızda provider ve versiyon belirten bloğumuzu yapıştırıyoruz:

9 
aws 
aws 
Overview 
Documentation 
O USE PROVIDER • 
aws 
Official by: 
Public Cloud 
Hash iCoræ 
Lifecycle management of AWS resources, including EC2, Lambda, EKS, ECS, VPC, S3, ROS, 
DynamoDB, and more. This provider is maintained internally by the HashiCorp AWS Provider team. 
VERSION 
4.58.o 
O PUBLISHED 
5 days ago 
< > SOURCE CODE 
O hashicorp/terrafarm- provider-aws 
How to use this provider 
To install this provider, copy and paste this 
code into your Terraform configuration. Then, 
run terraform init. 
Terraform 0.13+ 
terraforn { 
required_providers { 
aws = { 
- "hashicorp/aws" 
source - 
version = " 
4.58.a" 
provider "aus 

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

Buradan alıyoruz.

Terraform 0.13+ 
terraform { 
required_providers { 
aws = { 
hashicorp/ 
source = " 
version = " 
4.58.0" 
provider "aws" { 
* Configuration options 

Version için de ayrı bir version.tf oluşturmak best practice.

Module leri kullanmasak da yayınlanmış modulleri inceleyip nasıl hazırlanmış diye bakabiliriz. Projelerimiz için de bize örnek olur.

Örneğimizde docker içeren bir instance ayağa kaldıran terraform dosyası oluşturacaz.

Şimdi variables.tf i oluşturalım:

variable "instance\_type" {

type = string

default = "t2.micro"

}

Module paylaşacağımız için instance type ı da variable yazmak daha mantıklı. Default yazma nedeni ise değişiklik yapmadan kullanmak isteyenler olabilir. Anlaşılır bir variable olduğu için description yazmadık.

variable "key\_name" {

type = string

}

Key-name in default girilmesinin bir mantığı yok.

variable "num\_of\_instance" {

type = number

default = 1

}

Instance sayısını belirten bir değişken.

variable "tag" {

type = string

default = "Docker-Instance"

}

Deault tag

variable "server-name" {

type = string

default = "docker-instance"

}

Makineye verilen bir host name.

variable "docker-instance-ports" {

type = list(number)

description = "docker-instance-sec-gr-inbound-rules"

default = [22, 80, 8080]

}

Docker bir konteynır yapısı, konteynır lar bir çok port kullanabiliyor.

Burada da portlar variable olarak beliriliyor. Anlaşılması için de description yazılmış.

Şimdi main.tf e geçelim.

İlk önce bir data bloğumuz var:

data "aws\_ami" "amazon-linux-2" {

owners = ["amazon"]

most\_recent = true

filter {

name = "root-device-type"

values = ["ebs"]

}

filter {

name = "virtualization-type"

values = ["hvm"]

}

filter {

name = "architecture"

values = ["x86\_64"]

}

filter {

name = "owner-alias"

values = ["amazon"]

}

filter {

name = "name"

values = ["amzn2-ami-kernel-5.10-hvm\*"]

}

}

Her zaman kullandığımız en güncel amazon linux 2 ami sine ulaşmak için bu filtrelemeler giriliyor. Ancak aslında son filtre olan name de filtre olarak yeterli olurdu.

Userdata ile ilgili bir data template file ımız var:

data "template\_file" "userdata" {

template = file("${abspath(path.module)}/userdata.sh")

vars = {

server-name = var.server-name

}

}

Data template file data source gibi bir veri çekmek değil bir template oluşturmak için kullanılır. Userdata yı daha rahat kullanmaya yardımcı olur. Userdata yı dosya olarak belirtip onu data template file ile çekiyoruz.

<https://registry.terraform.io/providers/hashicorp/template/latest/docs>

Server-name makineye bir isim vermek için eklenmiş, variable olarak belirlenmiş ve burada bu şekilde gösteriliyor.

File, dosya içeriğini okuyan ve bir string olarak kullanılmasını sağlayan gömülü bir fonksiyondu. Abspath ise absolutepath in kısaltılmışıdır. Abspath bu userdata yı göstermek ve absoulte path haline getirebilmek için kullanılan bir gömülü fonksiyondur.

Examples 
> abspath(path. root) 
/ home/ user / some/ terraform/ 

Userdata:

#!/bin/bash

hostnamectl set-hostname ${server-name}

yum update -y

amazon-linux-extras install docker -y

systemctl start docker

systemctl enable docker

usermod -a -G docker ec2-user

# install docker-compose

curl -L "<https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname> -s)-$(uname -m)" \

-o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

Resource kısmına geliyoruz:

resource "aws\_instance" "tfmyec2" {

ami = data.aws\_ami.amazon-linux-2.id

instance\_type = var.instance\_type

count = var.num\_of\_instance

key\_name = var.key\_name

vpc\_security\_group\_ids = [aws\_security\_group.tf-sec-gr.id]

user\_data = data.template\_file.userdata.rendered

tags = {

Name = var.tag

}

}

Userdata da sonunda rendered fonk ile çekiyoruz. Dokümantasyonda bu şekilde gösterilmiş:

<https://registry.terraform.io/providers/hashicorp/template/latest/docs>

Example Usage 
# Template for initial configuration bash script 
data "template_file" "init" { 
template 
vars 
consul 
# Create a 
"init . tp 
"${aws_instance. consul. private_ip} " 
address 
web server 
"aws instance" 
resource 
"web" 
user data 
"${data. template_file . init. rendered} " 

İKinci resource umuz ise security groups;

resource "aws\_security\_group" "tf-sec-gr" {

name = "${var.tag}-terraform-sec-grp"

tags = {

Name = var.tag

}

dynamic "ingress" {

for\_each = var.docker-instance-ports

iterator = port

content {

from\_port = port.value

to\_port = port.value

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

Burada dynamic bloğu kullanılmış. Tekrar eden blokalr için dynamic blok kullanılır, resource, data, provider, and provisioner blocks altında kullanılır.

<https://developer.hashicorp.com/terraform/language/expressions/dynamic-blocks>

<https://learning-ocean.com/tutorials/terraform/terraform-security-group>

<https://learning-ocean.com/tutorials/terraform/terraform-security-group>

Variable olarak değişen kısımları belirledik, örneğimizde değişen veri portlar. Dynamic blokta for\_each kullanıyoruz.

Iterator kullanarak da variable isminin yerine iteraotr karşılığını kullanabiliyoruz port.value şeklinde.

Iterator kullanmasaydık docker-instance-ports.value kullanmamız gerekirdi.

<https://stackoverflow.com/questions/56442607/what-is-the-iterator-feature-for-in-terraforms-for-each>

output "instance\_public\_ip" {

value = aws\_instance.tfmyec2.\*.public\_ip

}

output "sec\_gr\_id" {

value = aws\_security\_group.tf-sec-gr.id

}

output "instance\_id" {

value = aws\_instance.tfmyec2.\*.id

}

output ta \* kullanmamızın sebebi, count kullanılan yerde \* kullanmazsak outputu verirken hata verir.

Output un şöyle bir önemi var: biz o module den bir bilgi çekeceksek o output içinde tanımlanmış olması gerekir.

resource "name" { 
vpc_id = module.na—. 

Örneğin instance ın public ip sini çekeceksek böyle çekemeyiz. Output ta tanımlı olması gerekir oradan çekilir.

Gitignore dosyamız örnek olması için hazırlanmış bir dosya. Bazı toollarda ignore dosyaları oluyor. Belirttiğimiz dosyaları ignore eder. Burada belirttiğimiz dosyaları push etmemesi için kullanılır:

# Local .terraform directories

\*\*/.terraform/\*

# Terraform lockfile

.terraform.lock.hcl

# .tfstate files

\*.tfstate

\*.tfstate.\*

# Crash log files

crash.log

# Exclude all .tfvars files, which are likely to contain sentitive data, such as

# password, private keys, and other secrets. These should not be part of version

# control as they are data points which are potentially sensitive and subject

# to change depending on the environment.

\*.tfvars

```hcl

provider "aws" {

region = "us-east-1"

}

module "docker\_instance" {

source = "<github-username>/docker-instance/aws"

key\_name = "clarusway"

}

```

Bir module kullanacakken bu kod bloğu ile kullanıyoruz. Başta ve sondaki üçer tane back tick bu aradaki ifadelerin renkli gösterilmesi için kullanılıyor.

Docker yüklü bir instace için module oluşturduk ve kaydettik. Şimdi GitHub da bir repository oluşturacaz:

**terraform-aws-docker-instance**

Ismi bu şekilde olacak. Eğer bir module yayınlayacaksak repo isminin başında terraform olmalı. Sonrasında ise provider ismi. Yani terraform-provider olmak zorunda başta.

Daha sonra çalıştığımız instance a sudo yum install git diyerek git yüklüyoruz.

Daha sonra git clone ile repository i locale klonluyoruz.

Sonrasında da cp ya da mv komutuyla module dosyalarını lokal repoya kopyalıyoruz.

Sırasıyla

git add .

git commit -m "should define your key file"

git push

git tag v0.0.1

git push --tags

Komutlarını giriyoruz. Git tag komutu repoyu tagliyor. Bu tag de terraformda version olarak görülecek.

Sonrasında terraform registry ye gidiyoruz. Sign in dedikten sonra Sign in with GitHub butonuna tıklayıp Authorize HAshcrop u seçiyoruz.

Sonrasında,

Publish v 
Provider 
Module 
Y. 1.1 
policy Library 

Publish / Module diyoruz.

Select Repository on GitHub 
yemreaydeniz/terraform-aws-dacker-instance 
Terms of Use 
PUBLISH MODULE 
How it works 

Burada repomuzun imi görünecek zaten. Seçtikten sonra bu repodaki module u terraform yayınlayacak.

Daha sonra GitHub repoya giidyoruz ve

main 
1 branch 
2 tags 
Go to file 
Add file 
EC2 Default User docker-instance-terraform-module files a... 
3ecceba 3 hours ago 
README.md 
main.tf 
docker-instance-terraform-module files are added. 
docker-instance-terraform-module files are added. 
< > Code • 
2 commits 
3 hours ago 
3 hours ago 
About 
No description, website, or topics provided. 
Readme 
O stars 
1 watching 

Sağdaki çarka tıklıyoruz.

Edit repository details 
Description 
Terraform module which creates a docker instance resource on AWS. 
Website 
https://registry'.terraform.io/modules/<account>/docker-instance/aws/latest 
Topics (separate with spaces) 
Include in the home page 
Cancel 
Save changes 

Description ve website ayarlarını yapıyoruz.

Bunları readme den aldık:

Description: Terraform module which creates a docker instance resource on AWS. 
Website: bttps: //registry. terraform. io/modules/<account>/docker-instance/aws/lates 

Şimdi published bir module ü kullanmayı görelim:

Cw-modules diye bir klasör oluşturup içinde main.tf oluşturuyoruz.

provider "aws" {

region = "us-east-1"

}

module "docker-instance" {

source = "clarusway/docker-instance/aws"

key\_name = "clarusway"

}

Main.tf e bunu yapıştırıyoruz. Key-name i değiştirmemiz gerekiyor. Buraya istersek variable ekleyebiliriz:

"aws { 
"us-east-I" 
region = 
module { 
source = "ofidan/docker-instance/aus•• 
key_name = "oliver" 
docker-instance-ports 122, 80, 81, 82, 
Of instance = 2 
gage , 
8ø9e] 

output "puh_ip" 
value = 

Output da ekleyebiliriz.

Terraform init dedik ve dosyalarımız oluştu:

v cw-modules 
v .terraform 
> modules 
> providers 
.terraform.lock.hcl 
.terraform.tfstate.lock.info 
main.tf 
{l terraform.tfstate 

Terraform apply diyoruz.

Instances (2) Info 
Q Find instance by attribute or tog (case-se 
Name 
terraform-server 
Docker Instance 

Dockes instance oluştu.

oliveremydesktop: S pen 
$ ssh —i "oliver.pem" ec 
oliver@mydesktop: 
The authenticity host 'ec2-3U-291-2U-127.com, 
ED2SS19 key fingerprint is SHA2S6:erUIYp9hbX2A11 
This key is not known by any Other names 
Are you sure you want to continue connecting (y 
Warning: Permanently added 
/ Amazon Linux 2 AMI 
https : 
docker —version 
Docker version 23.1ö.17, build laec7e1 

Docke da yüklenmiş instance a

Terraform destroy diyoruz.

Workspace egeçiyoruz:

Terraform da aynı dizin içinde farklı ortamlarda çalışıyormuş gibi çalışmaya imkan tanır. Bu genelde deneme amaçlı kullanılır. Developer lar production u denemek için konfigurasyon dosyasını başka dizine taşımadan workspace oluşturup aydı dizinde çalışıyorlar.

Ayrı bir dizin oluşturalım workspace isminde. Içinde workspace.tf dosyası oluşturalım. Amacımız bir insatnce oluşturmak ve dev ve prod olarak iki workspace te farklı ami ler kullanmak:

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "tfmyec2" {

ami = lookup(var.myami, terraform.workspace)

instance\_type = "${terraform.workspace == "dev" ? "t3a.medium" : "t2.micro"}"

count = "${terraform.workspace == "prod" ? 3 : 1}"

key\_name = "<your-pem-file>"

tags = {

Name = "${terraform.workspace}-server"

}

}

Burada lookup ve aşağıda map terraform fonk ile daha güzel bir terraform dosyası oluşturuyoruz.

variable "myami" {

type = map(string)

default = {

default = "ami-0cff7528ff583bf9a"

dev = "ami-06640050dc3f556bb"

prod = "ami-08d4ac5b634553e16"

}

description = "in order of aAmazon Linux 2 ami, Red Hat Enterprise Linux 8 ami and Ubuntu Server 20.04 LTS amis"

}

Default ortam, dev ortam ve prod ortamları için üç farklı ami belirlenmiş. Lookup fonk ile bu ami leri çağırıyoruz. Map içinden bir key seçip onun değerini seçmek istiyorsam lookup fonk kullanıyorum.

instance\_type = "${terraform.workspace == "dev" ? "t3a.medium" : "t2.micro"}"

Instance type ta ise condition kullanılmış. Terraform workspace dev se t3a.medium değilse t2.micro kullan.

count = "${terraform.workspace == "prod" ? 3 : 1}"

Burada da terraform workspace prod ise 3 , değilse 1 tane instance kaldır.

Key\_name değiştirmeliyiz.

Terraform workspace –help ile komutları inceleyebiliriz

terraform workspace - -help 
• ec2-use 
Usage: terraform [global options] workspace 
new, list, show, select and delete Terraform workspaces . 
Subcommands : 
delete 
list 
select 
show 
o ec2-use 
Delete a workspace 
List Workspaces 
Create a new workspace 
Select a workspace 
Show the name of the current workspace 

Terraform workspace show hangi workspace te olduğumuzu gösterir.

Delete siler, list listeler, new yeni workspace oluşturur, select work space i seçer.

New dediğimizde hem workspace i oluşturur hem o workspace e geçer.

r@ip-172-31-6e-183" wrkspaces:$ 
• ec2-use 
Created and 
*itched to workspace 
You' 
e their state, 
o if 
for this configuration _ 
• terrafor. list 
pmd 

Şimdi dev ve prod workspace lerini oluşturuyoruz.

v workspaces 
> .terraform 
v terraform.tfstate.d 
> dev 
> prod 

Sol tarafı incelediğimizde tfstate dizinimiz oluştu ve içinde dev ve prod dizinleri var. Workdpacelerin tfstate dosyaları bu klasörlerde oluşacak. Default tfstate ise normalde olduğu gibi workspace altında olacak.

v OPEN EDITORS 
G) READMEmd 
workspace.tf —/worksp... 
X environment —/worksp... 
v EC2-USER [SSH: LINUX-PLUS] 
.dotnet 
> .ssh 
.terraform.d 
.vscode-server 
clarusway-modules 
cw-modules 
terraform-aws-docker-insta... 
terraform-aws-docker-insta... 
workspaces 
v .terraform 
> providers 
environment 
> terraform.tfstate.d 
.terraform.lock.hcl 
workspaces > .terraform > 
prod 
1 
environment 
PROBLEMS 
o ec2-use 
OUTPUT 
DEBUG CONSOLE 
TERMINAL 

Environment hangi workspace te isek onu gösterir.

Şimdi terraform workpace select dev / terraform init / terraform plan diyoruz.

Init dedikten sonra provider workspace dizini altına iner, bütün workspace ler için ortaktır çünkü.

• ec2-use 
r@ip-172-31 
Terraform used the 
with the following 
+ create 
-58-149* workspaces:$ terraform plan 
selected providers to generate the following execution 
symbols : 
Terraform will perform the following actions: 
# will be created 
+ resource "aws_instance" "tfmyec2" { 
"ami-Ø664W5Ødc3f556bb" 
(known after apply) 

Burada ami dev için lookup ile seçildiğini görüyoruz dev = "ami-06640050dc3f556bb"

Tag ve instance type lar da aynı şekilde dev workspace için seçilmiş.

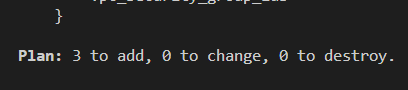
Dev workspace in tfstate i de dev dizini içinde:

v terraform.tfstate.d 
v dev 
.terraform.tfstate.lock.i 
f} terraform.tfstate 

Burada terraform apply diyelim.

Prod workspace ine geçelim ve orada da terraform plan diyelim.

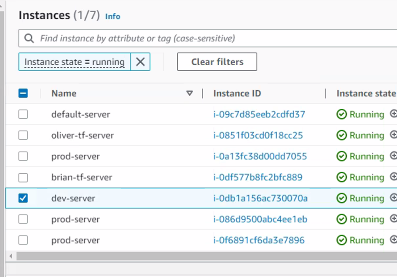
3 makine ayağa kaldıracağını gördük:



Ami ve type lar da prod workspace için veirlen değerler seçili.

Default workspace e geçip orada da apply diyoruz.

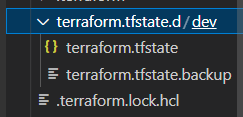
Konsolda da instance ları görüyoruz:



Bütün workspace ler için terraform destroy dememiz gerekiyor.

Şimdi workspace leri siliyoruz. Ancak içinde oldğumuz workspace i silemeyiz, default a geçip silelim.

Önce prod workspace ini sildim ve tfstate klasörü altından prod da silindi :



Şimdi de dev workspace ini silelim.



Default workspace kaldı.

Default ta oluşan resource için de destroy komutunu girmemiz gerekiyor.