Terraform Data Sources

Hiçbir şeyi ezberlemiyoruz. Terraform Data Sources neymiş gidip internette aratıyoruz.

<https://developer.hashicorp.com/terraform/language/data-sources>

Terraform da bir takım verileri çekmek için kullanılan bloktur. Örneğin AWS Linux instance için gidip konsoldam AMI yi bulup terraform a koyduk.

Ancak DevOps mühendisinin iki görevi var, süreklilik ve otomasyon. Bu nedenle terraform özelliklerini kullanarak bu bilgileri otomatik olarak çekmemiz gerekir.

Şimdi anlamak adına Terraform serverden bir AMI alıyoruz.

(1 / 2) Into 
Instances 
Q Find instance by attribute or tog (case •sensitive' 
Name 
brian-tt-se Ner 
Her-ser 
Instance state 
Actions 
V 
Instance ID 
i-0dfS77bgfc2btcgg9 
i-osbfd7dcfb5b12ac3 
Instance state A 
@ Running 
@ Running 
Instance type 
t2.smaII 
Q. micro 
V 
Status check 
@ 2/2 checks 
@ 2/2 checks 

Terraform server ı seçip Actions / Image and templates / Create Image diyoruz.

İsme my-ami veriyoruz, description a da aynısını yazabiliriz. Tag de istenirse eklenebilir.

Create image Info 
An image (also referred to as an AMI) defines the programs and 
Instance ID 
0 i-02dbcd12f73d19f09 (Terraform-server) 
Image name 
my-ami 
Maximum 127 characters. Can't be modified after creation. 
Image description - optional 
my-am 
Maximum 255 characters 

Şimdi de bu AMI den bir instance ayağa kaldıracaz. Snapshot aldığımız için instance stop edilir snapshot alınır ve tekrar start edilir, bu nedenle VS Code da bir çökme olur ancak geri gelir.

Şimdi çalıştığımız klasörde yoksa bir main.tf dosyası oluşturuyoruz ve readme deki

provider "aws" {

region = "us-east-1"

}

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "4.8.0"

}

}

}

locals {

mytag = "clarusway-local-name"

}

data "aws\_ami" "tf\_ami" {

most\_recent = true

owners = ["self"]

filter {

name = "virtualization-type"

values = ["hvm"]

}

}

variable "ec2\_type" {

default = "t2.micro"

}

resource "aws\_instance" "tf-ec2" {

ami = data.aws\_ami.tf\_ami.id

instance\_type = var.ec2\_type

key\_name = "aduncan"

tags = {

Name = "${local.mytag}-this is from my-ami"

}

}

Kodları main.tf e yapıştırıyoruz. Eğer klasör ilk kez oluşturuyorsak terraform init diyoruz. (daha önce terraform çalıştığımız klasörde çalışıyorsak gerek yok) Kod bloğundaki key name i kendi key name mizle değiştiriyoruz.

Amacımız AWS deki AMI yi id sini kullanmak. Data bloğunu inceleyelim.

Örneğin, ASG sürekli aynı AMI dan instance ayağa kaldırıyor ya da develeoperlar instance ta çlaışıyor ve her akşam image alıp kapatıyorlar ertesi gün o image dan instance kaldırıp çalışmaya başlıyorlar. Bu durumlarda bize ami gerekiyor.

Peki bu ami yi kullanan 100 tane terraform dosyası olduğunu düşünün. Bu durumda hepsini tek tek mi değiştireceğiz? Hayır, data bloğu ile bu ami nin id sini değiştirecez.

Terraform registry den geldik ve data source altında aws\_ami yi gördük:

AWS DOCUMENTATION 
158 matching results 
q uest 
aws_spot_instance_request 
v Data Sources 
ami 
ami ids 
aws_availability_zone 
• aWS 
aws 
av.'S 
aws 
aws 
host 
ec2 
ec2_instance_type 
ec2_instance_type_offering 

Burada açılan sayfada örnekler, argümanları görüyoruz.

Data Source: aws ami 
Use this data source to get the ID of a registered AMI for use in other resources. 
Example Usage 
data "aws ami" "example" { 
executable users 
self"] 
most recent 
name regex 
owners 
true 
self"] 

Argument Reference 
owners - (Optional) List of AMI owners to limit search. Valid values: an 
AWS account ID, self (the current account), or an AWS owner alias (e.g., 
microsoft 
aus-marketplace , 
amazon 
most_recent - (Optional) If more than one result is returned, use the most 
recent AMI. 
executable_users - (Optional) Limit search to users with explicit launch 
permission on the image. Valid items are the numeric account ID or self . 
- (Optional) If true, all deprecated AMIs are included in 
in clude_deprecated 
the response. If false, no deprecated AMIs are included in the response. If 
no value is soecified, the default value is false. 

data ile başlıyoruz;

most recent en güncelini al demektir.

- (Optional) If more than one result is returned, use the most recent AMI. 

owners ise ami yi hangi id den alacak, self demek terraform u çalıştıran kişinin aws id si, başka birinin ad si ya da amazon a ait bir ami istiyorsak amazon da yazılabilir.

filter bloğu ise ami leri filtrelemek içindir. biz tek bir ami id lazım bu nedenle filtreleme yapıyoruz.

filter { 
name 
values 
"virtualization-type" 
[ "hvm"] 

bu da bir filtreleme seçeneğidir. sanallaştırma tipi hvm olarak seçiyoruz.

Burada most recent le en günceli seçtik, eğer most recent demeseydik sorgunuz birden fazla değer verdi diye uyarı veriyor.

terraform plan 
data. aws_ni. tf_ni: Reading... 
Error: qtrry one Please try a mre specific search criteria, 
with data.aws ami.tf ami, 
on main.tf lihe 18. in data "aws ami" "tf ami": 
18: data "aws_ami• -tf_ami• 
or set 
mst recent' 
attribute to true. 

biz tek bir ami id ye indirgemek zorundayız.

terraform apply dediğimizde aynı ami ile instance ın ayağa kalktığını göreceğiz.

Gördükten sonra terraform destroy ediyoruz. snapshot u da silelim.

Şimdi Remote backend ya da remote state e bakalım:

Biz terraform apply dediğimizde terraform.tfstate dosyası oluştutruyor, bir öncekini ise tf.state.backup dosyasına koyuyuor.

Remote backend de state dosyalarını remote ta versiyonlayarak depolar.

<https://developer.hashicorp.com/terraform/language/settings/backends/configuration>

AWS S3 yi kullanabiliyoruz remote state göndermek için. Önce bucket oluşturmamız gerekiyor ve bucket ı backend olarak gösterecez.

Example Configuration 
terraform 
backend 
bucket = •mybucket• 
•path/to/my/key• 
key 
region 
= •us-east-I" 

şimdi backend diye bir klasör oluşturalım ve backend.tf dosyasını bu klasörde oluşturalım.

bucket nasıl oluşturulur; dokümantasyona gidip bakıyoruz.

Private Bucket w/ Tags 
resource 
bucket 
"aws s3 bucket" 
"my-tf-test-bucket" 
tags 
Name 
Envi ronment 
"my bucket" 
"Dev" 
resource 
bucket 
acl 
"aws s3 bucket acl" "example" 
aws s3 bucket.b.id 
"private" 

örneği var.

readme deki kod bloğunu backend.tf in içine yapıştıralım:

provider "aws" {

region = "us-east-1"

}

resource "aws\_s3\_bucket" "tf-remote-state" {

bucket = "tf-remote-s3-bucket-clarusways-changehere"

force\_destroy = true # Normally it must be false. Because if we delete s3 mistakenly, we lost all of the states.

}

resource "aws\_s3\_bucket\_server\_side\_encryption\_configuration" "mybackend" {

bucket = aws\_s3\_bucket.tf-remote-state.bucket

rule {

apply\_server\_side\_encryption\_by\_default {

sse\_algorithm = "AES256"

}

}

}

resource "aws\_s3\_bucket\_versioning" "versioning\_backend\_s3" {

bucket = aws\_s3\_bucket.tf-remote-state.id

versioning\_configuration {

status = "Enabled"

}

}

resource "aws\_dynamodb\_table" "tf-remote-state-lock" {

hash\_key = "LockID"

name = "tf-s3-app-lock"

attribute {

name = "LockID"

type = "S"

}

billing\_mode = "PAY\_PER\_REQUEST"

}

bucket 
"tf-remote -s 3- bucket-clarusways -changeher 
force_destroy = true # Normally it must 
be false. 
Because if we delete s3 
mistakenly, we lost all of the states. 

burada force\_destroy bucket in içi dolu olsa da silmek içindir. ders için kullandık normalde bu kullanılmaz. konsolda da s3 bucket boş olmadan silinmez.

encryption ve versionlama da terraform da ayrı resource olarak yapılıyor:

resou rce 
"aws_s3_bucket_server_side_encryption_confi 
guration" 
bucket 
bucket 
rule 
"mybackend" { 
aws s3 bucket .tf-remote-state. 
apply_server_side_encryption by_default 
sse_algorithm 
"AES256" 

üstte bucket ımızın encryption ayarlarını yapıyoruz.

resource "aws_s3_bucket_versioning" 
"versioning_backend s3" { 
bucket 
aws s3 bucket .tf-remote-state. id 
versioning_configuration 
— "Enabled" 
status 

üstte bucketımızın versiyonlama ayarlarını yapıyoruz. burada bucket ın id si yazılması gerekiyor. Neden? Dokümantasyon a bakıyoruz. oraya göre hareket ediyoruz.

Peki aynı anda başkası da terraform u aynı anda apply ederse ne olur? sistem hata verir çakışmadan dolayı. Bu nedenle dosyayı kilitlememiz gerekiyor. Dosyayı kilitlerken dynamodb kullanıyoruz.

DynamoDB tablosu kullanıyoruz ve onun içindeki partition key ile state dosyası kilitleniyor.

s3 lock için dynamodb kullanılmasını istiyor. dokümantasyondan alınan bilgiler.

şirketlerde bunu kullanmak zorundayız, profesyonellik bunu gerektirir.

resource "aws\_dynamodb\_table" "tf-remote-state-lock" {

hash\_key = "LockID"

name = "tf-s3-app-lock"

attribute {

name = "LockID"

type = "S"

}

billing\_mode = "PAY\_PER\_REQUEST"

}

kod bloğunu backend.tf e yapıştırıyoruz. kod bloğunun çalışması için dynamodb ve s3 full access yetkilerinin IAM role e tanınması gerekiyor.

resource 
name 
"aws_dynamodb table" "basic-dynamodb-table" 
billing_mode 
read _ capaci ty 
write_capacity 
hash_key 
range_key 
"GameScores" 
"PROVISIONED" 
"Userld" 
"GameTit1e" 

dokümantasyondan alınıyor bu kod bloğu da.

terraform init / terraform apply dedik. bucket ımız ve dynamodb table ımız oluşmuş olmalı.

backend.tf içinde provider aws dedikt ancak version vermedik. vermediğimiz için en güncel versionu çekti.

v backend 
.terraform/providers/registry.terraform.io/ hashicorp/aws/4.58.O/linux amd64 

peki versionu belirlememiz gerekiyor? neden? sonraki kod karışıklılıklarını engellemek istiyoruz. Her version da farklılıklar olabiliyor.

Role leri eklerken ihtiyaç olan policy leri ekliyoruz. fazladan bir yetki vermiyoruz.

şimdi main.tf ddosyamızın olduğu klasöre gidip backend configurationlaru yapıyoruz.

provider "aws" {

region = "us-east-1"

}

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "4.8.0"

}

}

backend "s3" {

bucket = "tf-remote-s3-bucket-clarusways-emrre"

key = "env/dev/tf-remote-backend.tfstate"

region = "us-east-1"

dynamodb\_table = "tf-s3-app-lock"

encrypt = true

}

}

burada hangi bucket ı kullanacağını belirtiyoruz ( backend.tf te oluşturduğumuz bucket ın ismi olmalı) key, prefiz oluşturuyor. dyanmo\_table ise nerede kitleyeceğini gösteriyor.

configuration dan sonra terraform init / terraform apply diyelim. neden tekrar terraform init diyoruz çünkü configurasyonları değiştirdik. configurasyon değiştirdiğimizde init tekrar yapmamız gerekiyor.

şimdi state dosyaları artık bucket ta version olarak depolanacak. bunu görebilmek için şimdi bir bucket oluşturalım. bu bucket sadece state dosyasında değişiklik görebilmek için yapılıyor.

resource "aws\_s3\_bucket" "tf-test-1" {

bucket = "clarusway-test-1-versioning"

}

terraform apply dedik.

s3 bucket a bakıyoruz versiyonları görüyoruz:

Objects 
Properties 
Objects 
(2) 
Objects are the fundamental entities stored in Amazon SS. You can use Amazon S3 inventory 
you'll need to explicitly grant them permissions. Learn more 
to get a list of all objects in your bucket. For others to access your objects, 
c 
O copy S3 URI 
Create folder 
Q Find objects by prefix 
Name 
tf-remote- 
backend.tfstate 
tf-remote- 
backend.tfstate 
Copy URL 
Type 
Download 
Open 
Show versions 
Version ID 
OD 5..x4aZzKeOsmsn 1 PQZqX5XPi3n pb6 
y Q nyTgLVesGn6rX4YoZzXLQlpwyLBp8R 
Delete 
Actions 
Last modified 
March 11, 2023, 
(UTC*OI :OO) 
March 11, 2023, 
1 ll50I44 
Size 
2.1 KB 
180.0 
> 
Storage 
class 
Standard 
Standard 

şimdi de aynı anda apply etmeyi simüle edelim:

resource "aws\_s3\_bucket" "tf-test-2" {

bucket = "clarusway-test-2121-locking-2"

}

yine bir resource oluşturalım ve terminali duplicate ederek ikisinde de terraform appy komutunu girelim:

r apply) 
Plan: 1 to add, to change, to destroy. 
Do you want to perform these actions? 
Terraform will perform the actions describe 
d above. 
Only 
'yes' will be accepted to approve. 
Enter 
a value: 

ilk termianl apply kodunu çalıştırırken;

ec2-user@ip-172-31-22-176 : lesson 
/terraform-ami-backend$ terraform apply 
Error : 
Error acquiring the state lock 
Error message: 
ConditionalCheckFai1edException: The 
conditional request failed 
Lock Info: 
ID: 
f8823c3 
Path: 
4c098a74-577c-4581-c6b6-b41bc 
tf-remote-s3-bucket-c1arusway 

ikinci terminal state lock tan dolayı kodu uygulamıyor. buradaki ID dynamodb nin ID si.

şimdi main.tf i destroy edebiliriz. daha sonra backend.tf dosyasının olduğu klasöre gidip orada da terraform destroy diyoruz.

bu konuyu interview de de sorabilirler. Aynı anda iki kişi apply etmeye çalışırsa ne olur, ne yapmak gerekir diye sorabilirler.

Terraform da provisioners nedir?

Biz instance ayağa kaldırırken configurasyon ayarları da yapıyoruz. Bunun için uzaktan instance a bağlanabilmenin yolu provisioner dır.

bunu AWS de user data ile yapıyorduk. terraform dokümantasyonunda da instance sayfasında user data argümanı var:

user_data - (Optional) User data to provide when launching the instance. 
Do not pass gzip-compressed data via this argument; see 
instead. Updates to this field will trigger a stop/start of the EC2 instance by 
default. If the is set then updates to this field 
will trigger a destroy and recreate. 
- (Optional) Can be used instead of user_data to pass 
base64-encoded binary data directly. Use this instead of user_data 
whenever the value is not a valid UTF-8 string. For example, gzip-encoded 
user data must be base64-encoded and passed via this argument to avoid 
corruption. Updates to this field will trigger a stop/start of the EC2 instance 
by default. If the is set then updates to this 

Eğer user datayı kullanamayorsak ve instance ayağa kalktıktan sonra conf ihtiyacımız da varsa provisioner kullanırız.

mantığı uzaktan instance a bağlanmak ve onun içinde işlemler yapmaktır.

local-exec , file- remote-exec olmak üzere üç tane provisioners var.

şimdi terminalde ana klasörde ya da lessons altında provisioner isimli bir directory oluşturup onun içine gidip main.tf oluşturuyoruz.

readme deki kod bloğunu bu main.tf e atıyoruz:

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~>4.0"

}

}

}

# burada version ~>4.0 4.0 dan büyük olsun da ne olursa olsun demektir.

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "instance" {

ami = "ami-0c02fb55956c7d316"

instance\_type = "t2.micro"

key\_name = "oliver"

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

tags = {

Name = "terraform-instance-with-provisioner"

}

#bir instance kaldırılacak.

provisioner "local-exec" {

command = "echo [http://${self.public\_ip](about:blank)} > public\_ip.txt"

}

#bu terraformun çalıştığı instance ta çalıştıracağı komutlar için kullanılır. Oluşturacağın instance ın public ipsini lokalde public\_ip.txt ye yazdır komutudur bu. bu instance henüz oluşmadığı için self kullanılıyor.

connection {

host = self.public\_ip

type = "ssh"

user = "ec2-user"

private\_key = file("~/oliver.pem")

}

#bu remote-exec çalıştırmadan önce remote instance ile bağlantı kurmak için gerekli kod bloğudur.

provisioner "remote-exec" {

inline = [

"sudo yum -y install httpd",

"sudo systemctl enable httpd",

"sudo systemctl start httpd"

]

}

#remote-exec ile uzak instance a bağlanarak bu kodları çalıştırma için kod bloğudur.

provisioner "file" {

content = self.public\_ip

destination = "/home/ec2-user/my\_public\_ip.txt"

}

}

#lokalden uzağa dosya kopyalamak için kullanılan file kod bloğudur.

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

name = "tf-provisioner-sg"

tags = {

Name = "tf-provisioner-sg"

}

# security grubu oluşturma

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" ]

}

#inboundlar

egress {

from\_port = 0

protocol = -1

to\_port = 0

cidr\_blocks = [ "0.0.0.0/0" ]

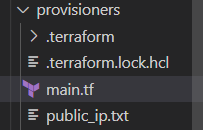
}

}

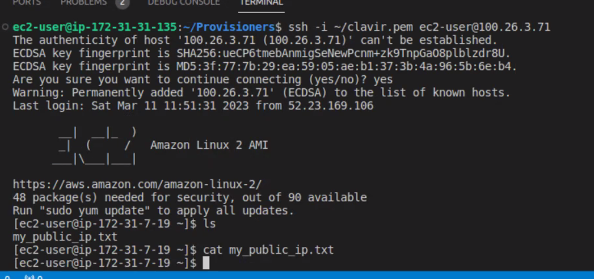
#outbound everywhere

Öncelikle pem file ımızı bulunduğumuz klasöre atmamız lazım. bunu sürükleyip vs code a da atabiliriz. yada lokalden scp komutu ile yapacaz.

terraform init / terraform apply diyoruz.

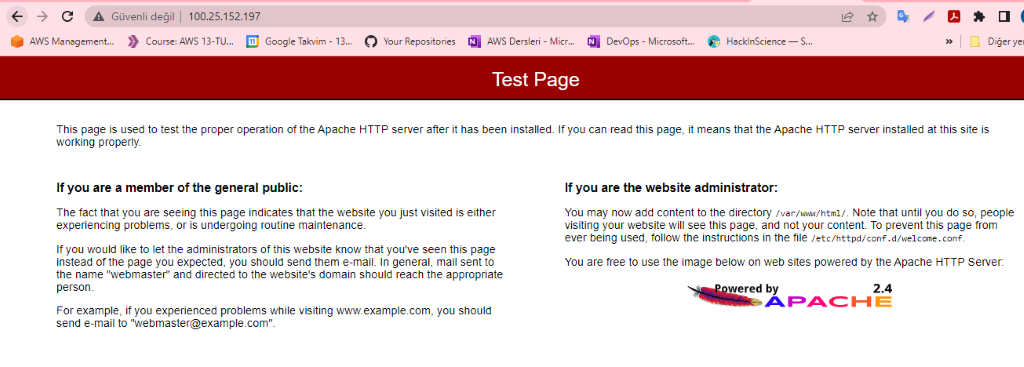


public\_ip.txt nin oluştuğunu gördük.



instance a bağlanırsak orada da file işlemiyle public\_ip.txt nin kopyalandığını görüyoruz.

appachi kurulması komutlarını girdiğimiz instance ın public ip sini browser a yapıştıralım:



site gelmiş.

terraform destroy diyebiliriz.