**Terraform’a Giriş — #1**



Terraform, HashiCorp tarafından piyasaya sürülen bir açık kaynak “infrastructure as a code” (IaC) tool’udur. Bu tool ile bulut sistemlerde birçok servisi aynı anda kurmak/yönetmek ve birçok defa kullanmak mümkündür.

Terraform; Açık kaynaktır, Stateful’dur, Declarative’dir,

AWS, Azure, GCP, GitHub, Docker gibi birçok provider ile uyumludur,

Ücretsizdir.

**Terraform’un Avantajları:**

**Platform Agnostic**

Birçok bulut sistem ve platform Terraform ile aynı anda yönetilebilir.

**State management**

Tüm resource’lar state file ile elimizin altında olur.

**Operator Confidence**

Kod yazıldıktan sonra operatöre yapıyı incelemesi istenir ve gözden kaçan hatalar ayıklanabilir.

**Terraform Nasıl Yüklenir**

<https://www.terraform.io/downloads.htm>

**Ubuntu/Debian**

$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -  
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"  
$ sudo apt-get update && sudo apt-get install terraform

**Centos/RHEL**

$ sudo yum install -y yum-utils  
$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo  
$ sudo yum -y install terraform

**MacOS**

$ wget https://releases.hashicorp.com/terraform/1.1.0/terraform\_1.1.0\_linux\_amd64.zip  
$ unzip terraform\_1.1.0\_linux\_amd64.zip

Unzip’lediğiniz dosyayı yeni oluşturduğunuz bir klasörün içine de atabilirsiniz.

$ mkdir terraform\_sample   
$ mv terraform terraform\_sample/

Son olarak da herhangi bir editör ile (vim veya nano gibi) .profile dosyasını açarak bash profilini düzenleyin. Aşağıdaki komutu dosyanın sonuna ekleyin ve kaydedip çıkın.

export PATH="$PATH:~/terraform\_sample"

Önemli not:

.profile dosyası gizlidir ve ls -la ile aranmalıdır.

PATH değişkenini güncellemek için source ~/.profile komutunun çalıştırılması gerekebilir.

İşlemler yapıldıktan sonra \*\*\*$ terraform version\*\*\* komutuyla yüklemeyi test edin.

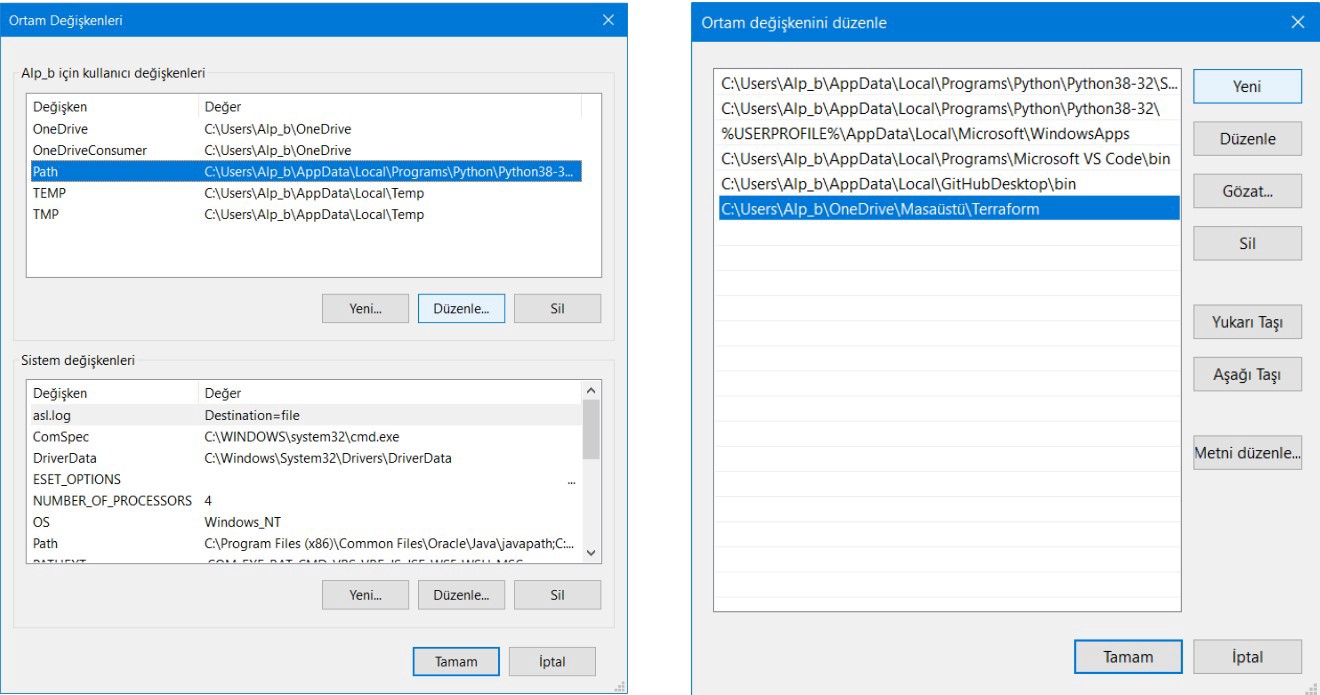
**Windows**

Dosyayı yükledikten sonra **.zip** klasörünün içindeki terraform dosyasını dışarı çıkarın ve bu dosyanın yolunu **PATH**’inize ekleyin.

Bunun için

1. Bilgisayarım / Bu Bilgisayar’ı açıp boşlukta sağ tıklayın,
2. Sistem penceresinde sol sütunda “Gelişmiş sistem ayarları”nı seçin,
3. Sistem Özellikleri penceresinde “Gelişmiş” / “Ortam Değişkenleri” sekmesini seçin,

4. Path’inizi düzenleyerek Terraform.exe dosyanızın bulunduğu klasörün yolunu ekleyin.



Cmd veya PowerShell’de**terraform version** yazarak kontrolü sağlayın.

**Amazon Linux**

$ sudo yum update -y  
$ sudo yum install -y yum-utils  
$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo  
$ sudo yum -y install terraform  
$ terraform version

$ terraform $ terraform -help komutları ile başlıca terraform komutlarını listeleyebilirsiniz.

$ terraform -help apply ya da $ terraform apply -help

$ terraform -help plan ya da $ terraform plan -help gibi komutlar ile alt komutlar hakkında detaylara ulaşabilirsiniz.

VS Code kullananlar için **HashiCorp Terraform**, **IntelliJ IDEA Keybindings**, **Terraform doc snippets** eklentileri çok kullanışlı olacaktır.

Linux tabanlı işletim sistemlerinde terraform -install-autocomplete komutuyla otomatik kod tamamlamayı yükleyebilirsiniz. Böylece ter yazıp **tab'a** bastıktan sonra terraform tamamlanacaktır.

**EC2 Üzerinden Terraform Çalışmak**

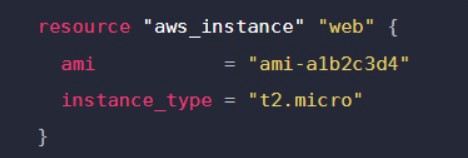
Çalışmaya başlamak için öncelikle uygulamamız gereken bazı hususlar var. Bunlardan biri aws configure komutu ile access\_key ve secret\_key girerek erişim sağlamak, ya da kurduğumuz EC2'ya IAM role oluşturmak.

Secret\_key açık kaynak olarak yazıldığında bir güvenlik açığı oluşturacağından IAM role oluşturmak daha güvenli olacaktır.

Bunun için kurduğumuz EC2'yu seçerek “Actions >> Security >> Modify IAM role” menüsü altından oluşturduğumuz “AmazonEC2FullAccess” tanımlı role’ü seçiyoruz. (Daha sonra kullanılacak servisler için de erişim izinleri verilmelidir)

**Resources**

Resource’lar Terraform dilinin en önemli elementidir. Infrastructure’ı oluşturan konfigure ettiğimiz objeleri (VM, DNS records vs.) tanımlar.



**Resource type:** “aws\_instance”

**Resource/local name:** “web”

**Resource name** aynı Terraform modülünde başka bir yerden bu kaynağa atıfta bulunmak için kullanılır, ancak bu modülün kapsamı dışında hiçbir önemi yoktur. **Harf veya alt çizgi ile başlamalıdır ve yalnızca harf, rakam, alt çizgi ve tire içerebilir.**

**Configuration Arguments** iki süslü parantezin ( { } ) arasına yazılanlardır.

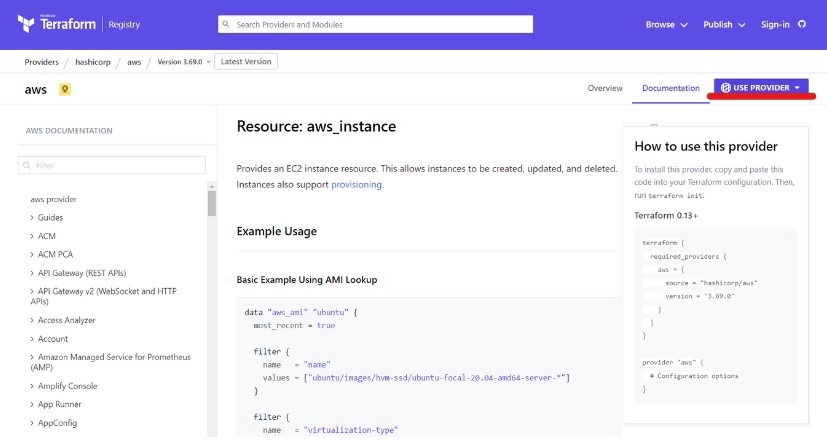
**Providers**

Providers, Terraform’un bulut sağlayıcılar, API’lar ver SaaS sağlayıcılar ile iletişime geçmesini sağlayan “plug-in”lerdir

Terraform konfigürasyonu kurulurken provider’lar mutlaka yazılmalıdır. Çünkü bu şekilde Terraform ilgili yüklemeyi yapar ve kullanır.



Provider bloğunu Terraform’un sitesinden alabiliriz. Ya da daha kolayı için google’a terraform aws instance yazdığınızda ilk link sizi ilgili sayfaya yönlendirecektir.



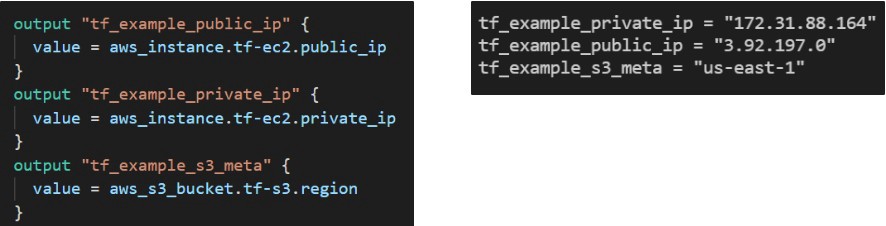
**State File**

terraform apply komutu ile **terraform.tfstate** dosyası oluşur ve içine infrastructure'ın metadata'sı kaydedilir. Bundan sonra infrastructure'da yapılacak her değişiklik state file ile kıyaslanır ve üzerinde yapılan değişiklikler planı görmek isteyene ya da apply edene gösterilir.

terraform refresh real-world infrastructure'da yapılan değişiklikleri **apply** komutunu uygulamadan state dosyasında günceller.

**Outputs**

Output değerleri, infrastructure hakkındaki bilgileri **command line’da** kullanılabilir hale getirir ve diğer Terraform konfigürasyonlarının kullanması için bilgileri açığa çıkarır.



Yazdığımız kod blokları ile infrastructure oluşturulduktan sonra çıktı olarak kurduğumuz EC2'nun public/private IP’leri ve S3 region yazacaktır.

**Bir Infrastructure’ın Ömrü**

**Init →** Yazdığımız .tf dosyalarını öncelikle initialize etmek gerekir. Bunun için terraform init komutu kullanılır ve böylece kullanılacak providerlar yüklenir ve kullanıma hazır hale gelir.

* Provider eklentileri yüklenir.
* .terraform.lock.hcl dosyası yüklenir. Versiyon kilitlenir, böylece başkası başka versiyon ile dosya üzerinde çalışamaz.

**Plan →** 0.11 versiyonundan sonra zorunluluğu kalkan bir aşamadır. terraform plan komutu ile **current state** ve **desired state** arasında fark olup olmadığı gözlemlenir. Özellikle çoklu çalışmalarda kontrolü sağlamak için hayati önem taşır.

“+” → oluşturulacak

“-” → silinecek

“~” → değiştirilecek

* **Real-world infrastructure** içinden bazı resource bloklarını alarak yeni bir plan oluşturmak için terraform plan out <plan\_ismi> kodu girilir ve yeni bir plan oluşturulur. Çok kullanılan bir uygulama değildir.

**Apply →** Infrastructure’ı aktif hale getirmek için terraform apply komutu kullanılır.

* Execution plan tekrar gösterilir ve onay istenir.
* “Yes” dendikten sonra **creating** başlar.
* **terraform.tfstate** dosyası oluşturulur. İçerisinde **resource** bilgileri yer alır. Ana dosyada değişiklik yapıldığında bu dosyayı referans alarak değişiklikleri gösterir.
* Bir kez daha apply yapıldığında bu sefer **terraform.tfstate.backup** dosyası oluşur. Bunun içinde de bir önceki **state file** yedeklenir.
* terraform apply -auto-approve yazarak otomatik olarak **yes** cevabını almış olur.
* terraform apply -refresh=false çok fazla instance açık olduğunu varsayarsak, ufak bir girdi için zaman kaybına sebep olmamak adına refresh yapmadan uygular.

**Destroy →** Her fani gibi çalışan sistemleri terminate etmek için terraform destroy komutu kullanılır.

* \*\*\*Konfigürasyonda servisin ana özelliklerinden birinde değişiklik yapılıp tekrar **apply**edilirse eski kaynak **terminate**edilir ve infrastructure yeniden oluşturulur. Ancak **Name-tag** gibi basit değişiklikler için terminate etmeden ekleme/çıkarma yapılır.

**En Çok Kullanılan Terraform Kodları**

terraform state list

Oluşturulan tüm resource’lar listelenir. (linux **ls** komutu gibi)

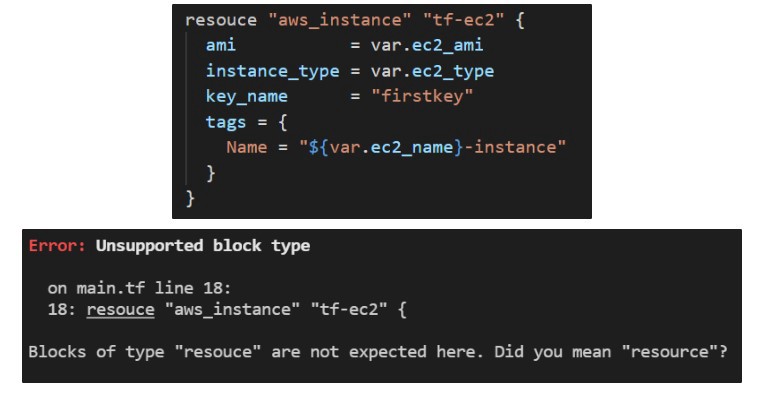
terraform fmt

Syntax’ı düzenlemek / derleyip toplamak için kullanılır. Özellikle çok kişinin üzerinde çalıştığı kodların okunabilirliğini artırmaya yarar.



terraform validate

Syntax hatası ve duplicate olup olmadığını gösterir. Hatalı satır ve kodu belirtir.

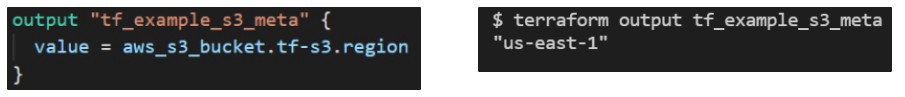


terraform import

Real-world infrastructure’dan state dosyasına bilgi çekmeyi sağlar.

terraform output

Output bloklarındaki bilgilerin çıktısını almayı sağlar.



terraform graph

Konfigürasyonun ya da execution planın görsel sunumunu sağlar. Bunun için yapılması gereken **terraform graph** komutunu girdikten sonra çıkan **digraph** ile başlayan kod bloğunu kopyalamak ve <https://dreampuf.github.io/GraphvizOnline> sitesine girip soldaki konsola yapıştırmak.

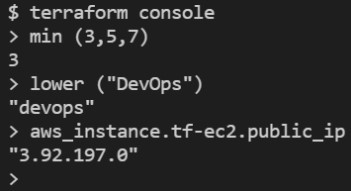
terraform show

Yazdığımız kodu **human-readable** formatta gösterir.

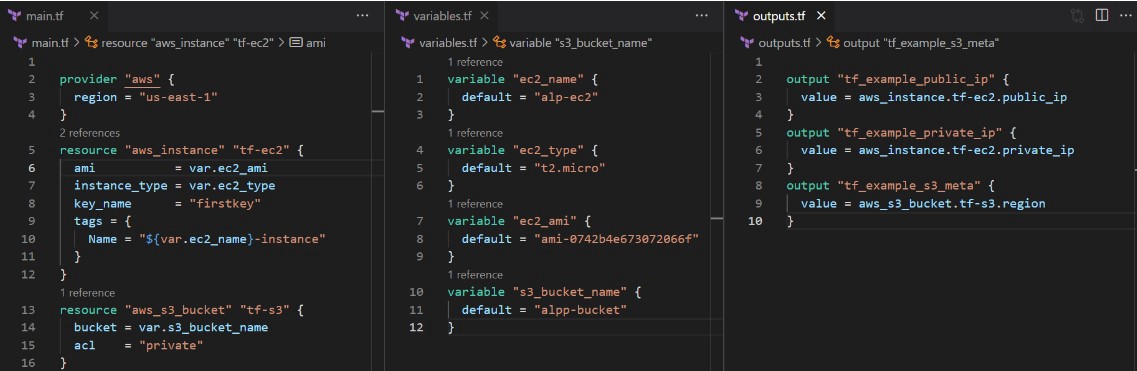
Machine-readable çıktı almak için sonuna -json eklenmelidir.

terraform console

Konsol modunu açarak işlem yapmayı sağlar.



**Blokların aynı dosya içinde olma şartı yoktur.** Aynı klasör içindeki tüm .tf uzantılı dosyalar beraber okunur. Böylece her blok çeşidini ayrı dosyalarda gösterip karışıklığın önüne geçilebilir.



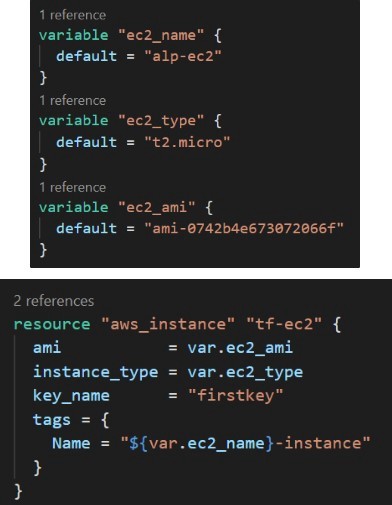
**Terraform’a Giriş — #2**



**Variables**

Variable’lar neden kullanılır? Başlıca sebepleri şunlardır:

* Gizli olması gereken bilgileri saklamak (AWS credentials gibi).
* Değişme ihtimali olan bilgileri rahat kullanmak (AMI’lar region’a göre değişiklik gösterir).
* Hazır variable’ları tekrar ve kolayca kullanmak.

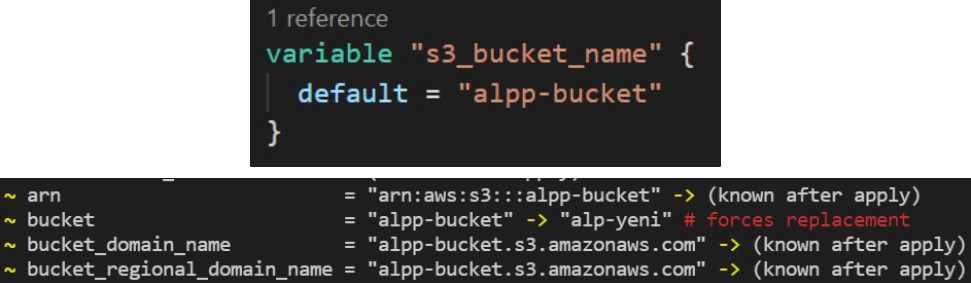


Variables ile kurulan EC2-instance

Terraform, değişken atamalarını aşağıdaki sıraya göre önceliklendirir:

1. -var ve -var-file komutları.
2. \***.auto.tfvars** ya da \***.auto.tfvars.json** uzantılı dosyalar (alfabetik sıraya göre).
3. terraform.tfvars.json
4. terraform.tfvars
5. Environment variables
6. Default variables

**-var**

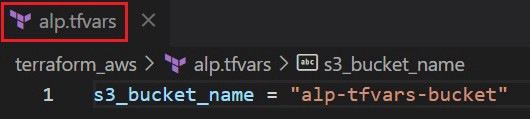


terraform plan -var="s3\_bucket\_name=alp-yeni” kodunu çalıştırdığımızda planda **bucket** isminin resimdeki **“alpp-bucket”** yerine **“alp-yeni”** olarak değiştirileceği görülecektir. Bunun sebebi de değişkenlerin yukarda belirtilen öncelikleridir. Uygulamak için de

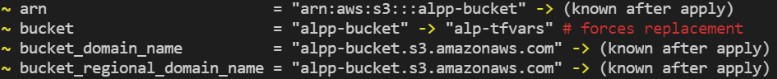
terraform apply -var="s3\_bucket\_name=alp-yeni” komutunu çalıştırmamız gerekmektedir.

**-var-file**

Herhangi bir isimde <name>.tfvars uzantılı dosyaya değişkenleri yazdığımızda, değişkenleri uygulamak için terraform plan -var-file="<name>.tfvars" yazılmalıdır.



$ terraform plan -var-file="alp.tfvars"



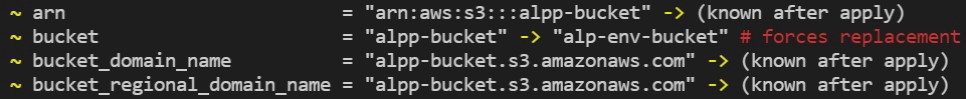
\*\*\*İçerisinde **terraform.tfvars** dosyası olan bir klasörde, farklı bir isimde **<name>.tfvars** dosyası oluşturulup -var-file eklentisi kullanılmazsa değişkenler default olarak **terraform.tfvars** dosyasından atanır.

**TF\_VAR\_**

Environment variables TF\_VAR\_ komutu ile tanımlanabilir.

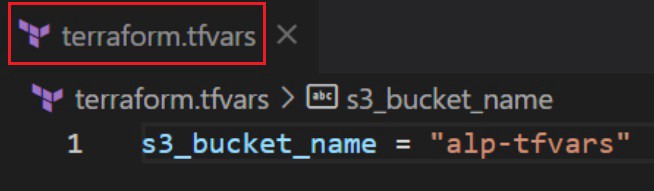
$ export TF\_VAR\_s3\_bucket\_name=alp-env-bucket

$ terraform plan

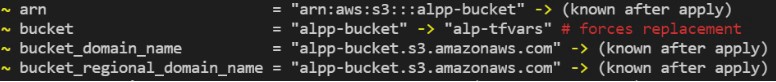


**terraform.tfvars**

terraform.tfvars dosyası oluşturarak değişken atamaya bir örnek:

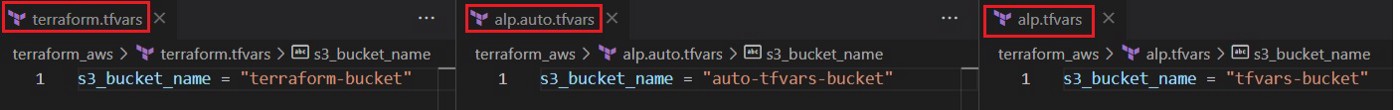


$ terraform plan



**.auto.tfvars**

Farklı isimde oluşturduğumuz dosyanın öncelik kazanması için uzantısının **auto.tfvars** olması gerekmektedir.





Kazanan **<name>.auto.tfvars**. Peki bu özellik nerede işimize yarayacak: Hazır bir şablonu birçok kişi kullanıyor ancak herkes aynı özellikleri kullanmıyor. Benim çalıştığım instance-type “t2.micro” iken başkasınınki “t2.large” olabilir. Bu sebeple variables ile referans edilen bu argümanlar için herkes bireysel **<name>.auto.tfvars** dosyasında kendi variable’larını belirlerse, atamalar herkesin kendine ait dosyası üzerinden gerçekleşecektir.

**Locals**

Sık geçen açıklamalar ve kolay değiştirilebilir değişkenler için birçok **variable** bloğu oluşturmak yerine tek bir **locals** bloğunda birçok değişken oluşturulabilir.



**Count & for\_each**

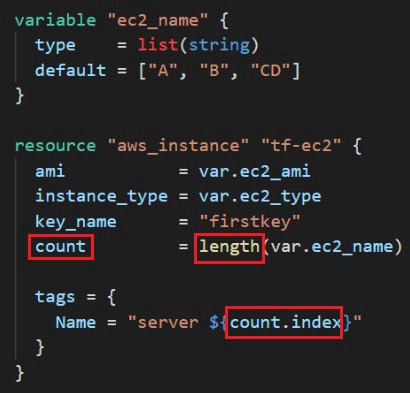
Bir resource bloğu default olarak bir tane infrastructure objesi oluşturabilir. Ancak bazen ayrı bloklar yazmadan aynı objeden birden çok oluşturulması gerekebilir. count ve for each çoklu obje oluşturmak için kullanılan **meta-data** argümanlarıdır.

\*Bir resource ya da module bloğunda count ve for\_each aynı anda kullanılamaz.

count: Her resource tipinde ve module’de kullanılabilir. Yalnızca doğal sayılar (0, 1, 2, ....) yazılabilir.

count.index: 0’dan başlayan dizinler için kullanılır.

length: String veya listenin eleman sayısını sayısal değere dönüştürmek için kullanılır.



Örnekte 3 elemanlı bir liste var. count = 3. Yani 3 adet EC2 instance kurulacak.

count.index yazdığımız kod (0, 1, 2) olarak atandığı için **Name-Tag’leri** server 0 / server 1 / server 2 olacaktır.

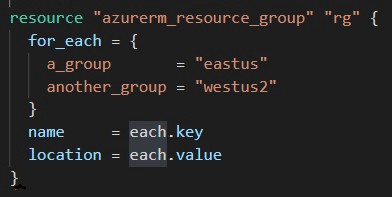
${count.index+1} deseydik bu sefer server 1 / server 2 / server 3 şeklinde çıktı alacaktık.

Oluşturulan instance’lar aynı ve sıralıysa count kullanmak kullanışlıdır. Ancak sıralı değil ve farklı adlandırmalara ihtiyaç duyuluyorsa for each daha kullanışlı olacaktır.

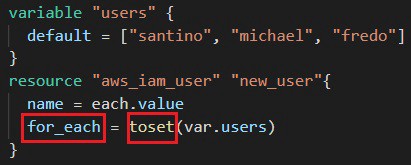
for\_each : Her resource tipinde ve module’de kullanılabilir. Yalnızca map veya set değerler yazılabilir.

toset : Terraform’un set için bir syntax’ı yoktur. Ancak string listeleri toset ile set elemanlarına çevrilebilir.

tomap : Argümanları map değerine çevirir. Pek kullanılmaz.



Map



Set

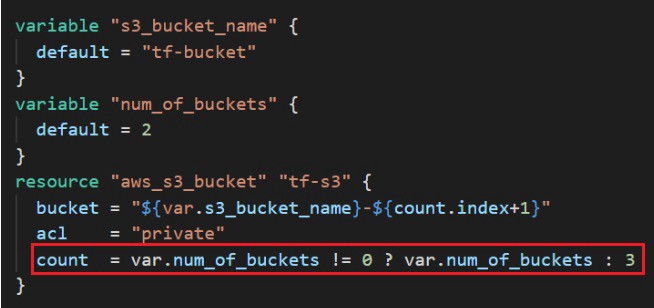
Aynı anda santino, michael ve fredo isimli 3 **iam user** oluşturduk.

\*toset - tomap gibi gömülü fonksiyonlar Terraform’u kullandıkça öğrenilecektir.

**Conditionals**

condition ? true\_value : false\_value yapısı kullanılır. Bu ne demektir? Aşağıdaki örnekte **count** için eğer ki **num\_of\_buckets** sıfıra eşit değilse (yani true ise) **num\_of\_buckets** değeri yazdırılır; eşitse (false ise) “3” yazdırılır.

Variable bloğunda **num\_of\_buckets**’a 2 değeri atandığı için sıfıra eşit değildir ve **condition** doğrudur. Bu sebeple **count** değeri 2 olur.

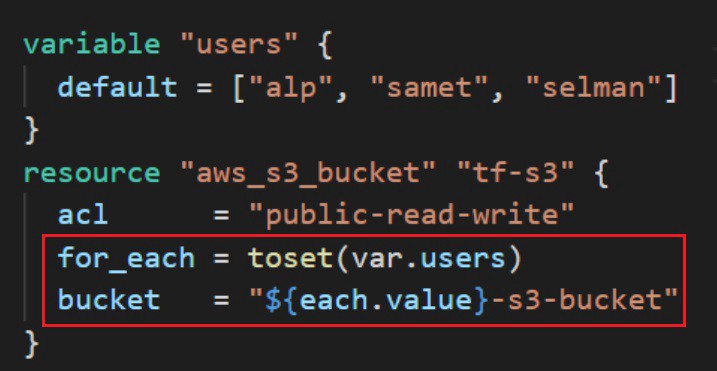


Böylece **tf-bucket-1**ve **tf-bucket-2** adlı iki bucket oluşturulur.

**Loops**

Birçok bucket, iam user vs. oluşturmak istiyorsak bunu **for expressions** ile kolayca yapabiliriz.

İlk kelimesi farklı isimlerden oluşan 3 adet **bucket** oluşturmak için 3 elemanlı bir liste yazıyoruz.



Çıktı olarak **alp-s3-bucket** , **samet-s3-bucket** , **selman-s3-bucket** isimli 3 bucket oluşturulur.

İkinci örnekte ise a **variable**’ı **users** listesinin içinde **iterate** edilir ve 3 karakterden uzun olma koşulunu sağlayan elemanların büyük harfle **output**’unun alınması sağlanır.

Kısaca listedeki 3 harften uzun elemanların büyük harflerle çıktısı alınır.



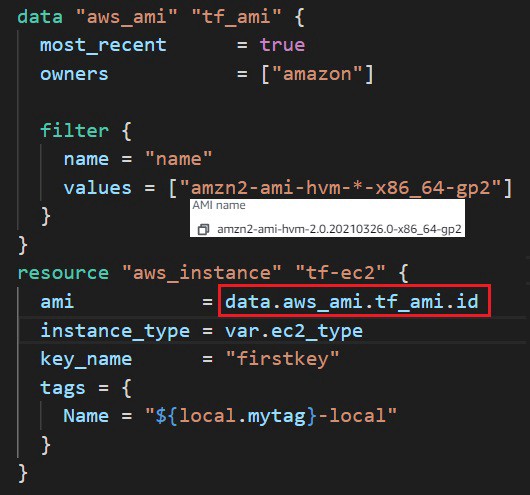
Çıktılar **SAMET**ve **SELMAN**olacaktır.

**Data Sources**

Data source’u olan bir bilgiyi çekmek için kullanırız. Bu Amazon’un kendi OS’ı, snapshot’ını aldığımız bir EC2 vs. olabilir. Örneğimizi yeni bir EC2 instance ayağa kaldırma üzerinden yapacağız. Bu kod bloğu için Terraform’un zengin dokümantasyonuna ulaşmak istiyorum ve google’a “data aws\_ami terraform” yazıyorum.

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami>

Ulaştığım sayfada tüm detaylar hizmetime sunulmuş durumda.

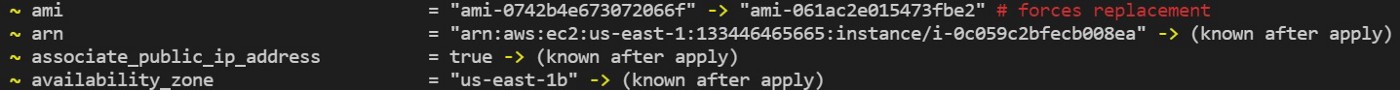


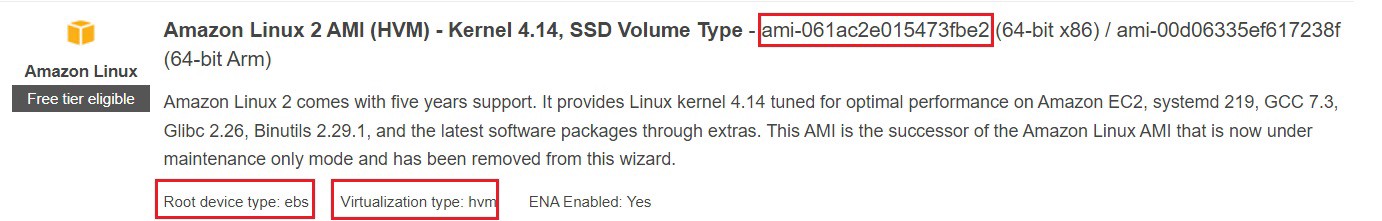
Yaptığımız örnekte;

most\_recent **true** diyerek en güncel sürümü seçtim.

owners **amazon**’un AMI’ını kullanmak istiyorum.

filter devreye sokarak **Root-device-type: ebs** ya da **Virtualization-type: hvm** filtrelemesi de yapabilirim. Ben AMI adını kullanarak daha spesifik bir filtrelemeyi tercih ettim.





**Terraform’a Giriş — #3**



**Provisioners**

Terraform kendi dokümanında provisioners kullanımını son çare olarak düşünmemizi ister.

Provisioners, sunucuları veya altyapı objelerini (Infrastructure Objects) servise hazırlamak için yerel veya uzak bir makinede belirli eylemleri modellemek için kullanılır.

**local-exec**

Terraform resource ile ayağa kaldırdığımız bir EC2’nun public ve private IP’lerini localimizde .txt dosyası içinde saklamak istiyoruz. Bunu local-exec komutu ile şu şekilde yaparız:



Öncelikle 2 adet EC2 ayağa kaldırdığımızı varsayalım. Artık uzak bağlantı gereksinimi duyduğumuz için **SSH**içeren security\_groups da tanımladık (Sec\_group kod bloğu aşağıda paylaşılacaktır.). provisioner bloğunu, süslü parantezlere dikkat edersek, resource bloğunun içinde yazıyoruz. Bu sebeple resource artık **parent resource** olarak tanımlanabilir.

**provisioner** bloğunun içinde dikkatimizi çeken 2 komut var: **command** ve **self**.

command - Tek satırlık kodları yazmak içindir. Biz burada instance’larımızın IP’lerinin .txt dosyalarının içine aktarılmasını istiyoruz.

self - Parent resource’un tüm niteliklerini taşıyan bir komuttur. Örnek olarak **self.public\_ip** yazarak resource’un public IP’sine yönlendirme yaparız.

Kodumuzu **apply** ettiğimizde infrastructure’ı kurduğumuz klasörde **public\_ip.txt** ve **private\_ip.txt** dosyaları oluşacak, içinde de oluşturduğumuz EC2’ların IP adresleri yazacaktır.

**remote-exec**

Az önce localimize yönelik bir işlem gerçekleştirdik. Şimdi remote-exec ile kurduğumuz EC2 içinde bir işlem yapacağız.

Örnek olarak E2’nun içinde Apache Server oluşturmak ve “Hello World” çıktısı almak istiyoruz.



Bu **provisioner** ve **connection** blokları da resource bloğunun içinde yer alacaktır.

host - Bağlanılacak kaynağın adresi.

type - Bağlantı tipi. Linux için ssh, Windows için winrm.

user - Amazon Linux 2 bağlantısı yaptığımız için “ec2-user”. Ubuntu olsaydı “ubuntu” olacaktı.

private\_key - key.pem dosyamız.

inline - command komutu ile tek satır yazabilirken, çoklu satır girmemizi sağlayan komut inline’dır.

**User data**’yı EC2 kurarken girmek zorundayız ancak remote-exec ile EC2 kurduktan sonra da uzaktan komut girebiliriz. Bu, ikisini birbirinden ayıran can alıcı noktadır.

Ancak sonradan eklediğimiz **provisioner** bloğunu parent resource içine eklersek Terraform herhangi bir değişiklik görmeyecektir. **Trigger** sağlamak için yeni bloğu **“null\_resource”** altında çalıştırmamız gerekecektir.

resource "null\_resource" "izmir" {  
 connection {  
 host = aws\_instance.instance.public\_ip  
 type = "ssh"  
 user = "ec2-user"  
 private\_key = file("firstkey.pem")  
 }provisioner "remote-exec" {  
 inline = [  
 "sudo yum -y install httpd",  
 "sudo systemctl enable httpd",  
 "sudo systemctl start httpd",  
 "echo 'Hello World' > index.html",  
 "sudo cp index.html /var/www/html/"  
 ]  
 }  
}

Security Group oluşturmak için gerekli kod bloğu:

resource "aws\_security\_group" "tf-sec-gr" {  
 name = "tf-provisioner-sg"  
 tags = {  
 Name = "tf-provisioner-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"]  
 }  
 }

Uzak bağlantı için port-22’ye, Apache Server kurmak için port-80’e ihtiyacımız vardır.

egress bloğunda **protocol’e “-1”** yazılmasının sebebi “**all”** demektir. Yani tüm protokol çıkışlarına izin verilir.

**Remote State**

State dosyamızı s3 bucket içine koyarak yapılan değişiklerin tüm departmanların ulaşabileceği bir uzak bilgisayarda muhafazası sağlanabilir.. Bunun için öncelikle ayrı bir klasörde yeni bir bucket resource bloğu oluşturmalıyız. Bu sefer detaylandırarak versiyonlamayı açtık, destroy için zorlama talimatı verdik ve **AES256** veri şifrelemesi ekledik.

Bunların haricinde birçok kişinin ulaşabildiği bir state dosyasında bizi zora düşürecek olan durum, birden fazla kişinin aynı anda **apply** etmesi olacaktır. Bunu engellemek için de **sınavlarda ve mülakatlarda sıkça karşımıza çıkabilecek bir konu** olan dynamoDB servisinin LockID özelliğini kullanıyoruz. hash\_key = “LockID” yazarak bir kişinin **apply** komutunu vermesiyle yeni bir **apply** için değişikliğin bitmesi beklenecek. Aynı anda apply komutu verebilmenin önüne geçilecek.



State dosyası için yeni bir S3\_bucket oluşturduk ancak local’de çalıştığımız main.tf dosyasına da bu bilgiyi eklememiz gerekmektedir.



[main.tf](http://main.tf/)’e eklediğimiz backend bloğu ile bundan sonra **state**dosyası local’e değil s3\_bucket içinde env/dev/tf-remote-backend.tfstate içine yüklenecektir.

**Modules**

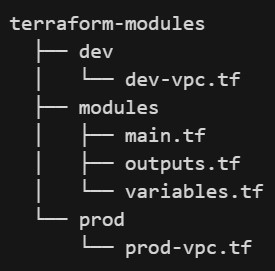
Şirketinizde 10 tane departman olduğunu düşünelim ve her departman kendi VPC’sini kurmak yerine DevOps ekibi bir tane VPC modülü oluşturur ve her departman o modülü kullanarak sistemini kurar.

Modül’ler 2'ye ayrılır:

Bir Terraform modülü (genellikle bir konfigürasyonun root modülü), kaynaklarını konfigürasyona dahil etmek için diğer modülleri çağırabilir.

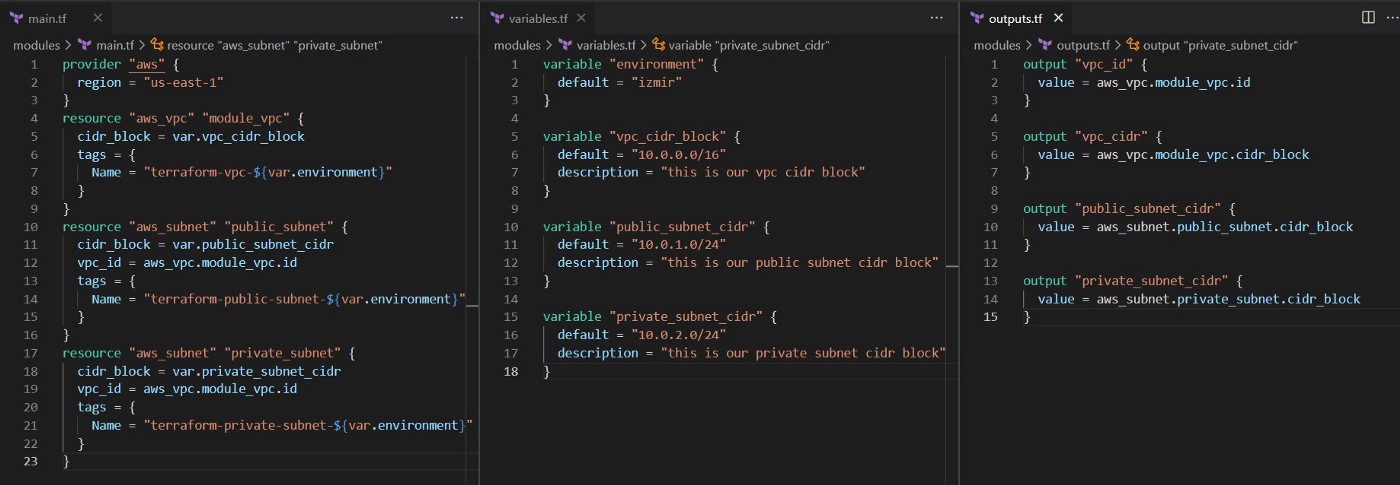
1. **Root Module:** Child modülleri çağırdığımız modüle denir.
2. **Child Modules:** Root modül tarafından çağrılan modüllere denir.

Örneğimizde “Development” ve “Production” departmanlarını ele alalım. Bu iki departmana VPC modülü oluşturmak için aşağıdaki gibi bir “ağaç” oluşturalım:



**dev** ve **prod** dizinleri ile root modüllerini,

**modules** dizini ile child modülünü oluşturmuş olduk.



modules child modülü

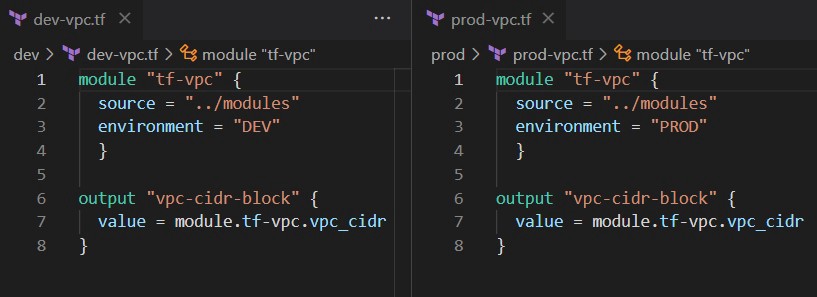
Bilgileri çekeceğimiz child modülümüzün içine baktığımızda:

10.0.0.0/16 cidr bloğu ile VPC (Name\_tag = terraform-vpc-izmir)

10.0.1.0/24 cidr bloğu ile public subnet (Name\_tag = terraform-public-subnet-izmir)

10.0.2.0/24 cidr bloğu ile private subnet (Name\_tag = terraform-private-subnet-izmir)

kurulacak.



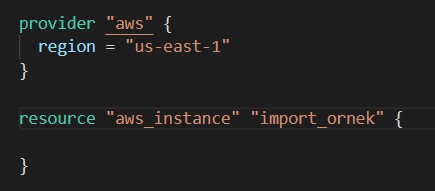
**dev-vpc**ve **prod-vpc** adlı root modüller, **modules** klasöründen child modülü çağırarak tüm kaynaklarına erişmiş olacak.

**Import**

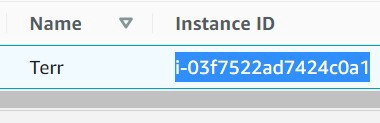
Elimizde hazır çalışan bir kaynak var ve biz bu kaynağın bilgilerini kendi kodumuza çekmek istiyoruz. **Import** komutu bu konuda bize yardımcı olacaktır.

Ancak şöyle bir durum mevcut. Hazır kaynağı bizim kodumuzun içine değil, state dosyasının içine atacaktır. Yani biz kaynak bilgilerini state dosyasından alıp kodumuzun içine işlemedikçe kaynakları kullanmamız mümkün olmayacaktır. Bu konuyla alakalı olarak doküman der ki: **“A future version of Terraform will also generate configuration.”**

Örnek olarak hazırda çalışan bir EC2-instance’ımız olsun ve biz bunu kodumuzun içine çekmek istiyoruz. Öncelikle Terraform’da bir **aws\_instance** resource’u oluşturalım:



Daha sonra Name\_Tag’i Terr olan EC2-İnstance’ımızın ID’sini kopyalayalım.



Instance ID’yi import etmek istediğimiz kaynağa ekleyelim

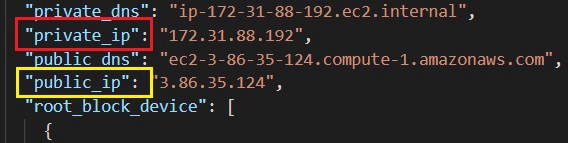
$ terraform import aws\_instance.import\_ornek i-03f7522ad7424c0a1

İşlem tamamlandığında import ettiğimiz kaynağın bilgileri state dosyasının içinde olacaktır.

Hazır instance’ın IP adresleri:



Import ettikten sonra state file içine eklenen IP adresler:



Bundan sonra benim yapmam gereken terraform show diyerek ihtiyacım olan argümanları kodumun içine eklemek.

