Terraform -1

Wednesday, 1 September 2021 17.14

Terraform is the **infrastructure as code** offering from **HashiCorp**. It is a tool for building, changing, and managing infrastructure in a safe, repeatable way. Operators and Infrastructure teams can use **Terraform** to manage environments with a configuration language called the **HashiCorp Configuration Language (HCL)** for human-readable, automated deployments.

From <https://lms.clarusway.com/mod/lesson/view.php?id=4344>

Terraform, HashiCorp'un sunduğu kod olarak altyapıdır. Altyapıyı güvenli ve tekrarlanabilir bir şekilde oluşturmak, değiştirmek ve yönetmek için bir araçtır. Operatörler ve Altyapı ekipleri, insan tarafından okunabilen, otomatikleştirilmiş dağıtımlar için HashiCorp Yapılandırma Dili (HCL) adı verilen bir yapılandırma diliyle ortamları yönetmek için Terraform'u kullanabilir.

Infrastructure as Code

TURKISH

Kod Olarak Altyapı

From <https://lms.clarusway.com/mod/lesson/view.php?id=4344&pageid=5359>

Infrastructure as Code is the process of managing infrastructure in a file or files rather than manually configuring resources in a user interface. A resource in this instance is any piece of infrastructure in a given environment, such as a virtual machine, security group, network interface, etc.

TURKISH

Kod Olarak Altyapı, kaynakları bir kullanıcı arabiriminde manuel olarak yapılandırmak yerine bir dosya veya dosyalardaki altyapıyı yönetme sürecidir. Bu örnekteki kaynak, belirli bir ortamdaki sanal makine, güvenlik grubu, ağ arabirimi vb. gibi herhangi bir altyapı parçasıdır.

At a high level, Terraform allows operators to use HCL (HashiCorp Configuration Language) to author files containing definitions of their desired resources on almost any provider (AWS, GCP, GitHub, Docker, etc) and automates the creation of those resources at the time of apply

TURKISH

Terraform, yüksek düzeyde, operatörlerin hemen hemen her sağlayıcıda (AWS, GCP, GitHub, Docker, vb.) istenen kaynakların tanımlarını içeren dosyaları yazmak için HCL (HashiCorp Yapılandırma Dili) kullanmasına izin verir ve o anda bu kaynakların oluşturulmasını otomatikleştirir. başvurmak

Workflows

İş akışları

A simple workflow for deployment will follow closely to the steps below:

TURKISH

Dağıtım için basit bir iş akışı aşağıdaki adımları yakından takip edecektir:

- **Scope** - Confirm what resources need to be created for a given project.
  Kapsam - Belirli bir proje için h**angi kaynakların oluşturulması gerektiğini** onaylayın.
- Author - Create the configuration file in HCL based on the scoped parameters
  Yazar - Kapsamlı parametrelere göre yapılandırma dosyasını HCL'de oluşturun

- **Initialize** - Run terraform init in the project directory with the configuration files. This will download the correct provider plug-ins for the project.

TURKISH

Başlat - terraform init'i proje dizininde yapılandırma dosyalarıyla çalıştırın. Bu, proje için doğru sağlayıcı eklentilerini indirecektir.

- Plan & Apply - Run terraform plan to verify creation process and then terraform apply to create real resources as well as state file that compares future changes in your configuration files to what actually exists in your deployment environment.

From <https://lms.clarusway.com/mod/lesson/view.php?id=4344&pageid=5361>

Planla ve Uygula - Oluşturma sürecini doğrulamak için terraform planını çalıştırın ve ardından yapılandırma dosyalarınızdaki gelecekteki değişiklikleri dağıtım ortamınızda gerçekte var olanlarla karşılaştıran durum dosyasının yanı sıra gerçek kaynaklar oluşturmak için terraform uygulayın.

Advantages of Terraform

TURKISH

Terraform'un Avantajları

While many of the current offerings for infrastructure as code may work in your environment, Terraform aims to have a few advantages for operators and organizations of any size.

TURKISH

Kod olarak altyapı için mevcut tekliflerin çoğu ortamınızda çalışabilirken, Terraform her büyüklükteki operatörler ve kuruluşlar için birkaç avantaj sağlamayı amaçlamaktadır.

From <https://lms.clarusway.com/mod/lesson/view.php?id=4344&pageid=5362>

⚜1. Platform Agnostic

In a modern datacenter, you may have several different clouds and platforms to support your various applications. With Terraform, you can manage a heterogeneous environment with the same workflow by creating a configuration file to fit the needs of your project or organization.

TURKISH
Modern bir veri merkezinde, çeşitli uygulamalarınızı desteklemek için birkaç farklı bulut ve platformunuz olabilir. Terraform ile projenizin veya kuruluşunuzun ihtiyaçlarına uygun bir konfigürasyon dosyası oluşturarak aynı iş akışına sahip heterojen bir ortamı yönetebilirsiniz.

2. State Management

Terraform creates a state file when a project is first initialized. Terraform uses this local state to create plans and make changes to your infrastructure. Prior to any operation, Terraform does a refresh to update the state with the real infrastructure. This means that Terraform state is the source of truth by which configuration changes are measured. If a change is made or a resource is appended to a configuration, Terraform compares those changes with the state file to determine what changes result in a new resource or resource modifications.

TURKISH
Terraform, bir proje ilk başlatıldığında bir durum dosyası oluşturur. Terraform, bu yerel durumu planlar oluşturmak ve altyapınızda değişiklikler yapmak için kullanır. Herhangi bir işlemden önce Terraform, durumu gerçek altyapı ile güncellemek için bir yenileme yapar. Bu, Terraform durumunun, konfigürasyon değişikliklerinin ölçüldüğü gerçeğin kaynağı olduğu anlamına gelir. Bir yapılandırmaya bir değişiklik yapılırsa veya bir kaynak eklenirse, Terraform hangi değişikliklerin yeni bir kaynak veya kaynak değişiklikleriyle sonuçlandığını belirlemek için bu değişiklikleri durum dosyasıyla karşılaştırır.

3. Operator Confidence (Operatör Guveni)

The workflow built into Terraform aims to instill confidence in users by promoting easily repeatable operations and a planning phase to allow users to ensure the actions taken by Terraform will not cause disruption in their environment. Upon terraform apply, the user will be prompted to review the proposed changes and must affirm the changes or else Terraform will not apply the proposed plan.

From <https://lms.clarusway.com/mod/lesson/view.php?id=4344&pageid=5362>

TURKISH
Terraform'da yerleşik iş akışı, kolayca tekrarlanabilir işlemleri teşvik ederek kullanıcılara güven aşılamayı ve kullanıcıların Terraform tarafından gerçekleştirilen eylemlerin çevrelerinde bozulmaya neden olmayacağından emin olmalarına olanak tanıyan bir planlama aşaması sağlamayı amaçlar. Terraform uygulandığında, kullanıcıdan önerilen değişiklikleri gözden geçirmesi istenir ve değişiklikleri onaylaması gerekir, aksi takdirde Terraform önerilen planı uygulamaz.

Alt yapimizi gelsitirme degistirme amacli yapilmis bir tooldur islerimizi kolaylastiran infrastruce (alt yapui=)as coded
Alt yapimizin basitce kod olarak ifadesidir

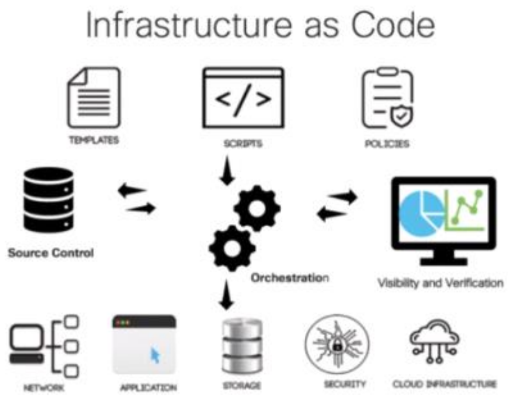
What is Terraform?

- **Terraform** is the infrastructure as code offering from HashiCorp.
- It is a tool for building, changing, and managing infrastructure in a safe, repeatable way.

DevOps
HashiCorp
Terraform

Gvo Programlari ile uretilmis bir dildir islerimizi kolaylöastirir.

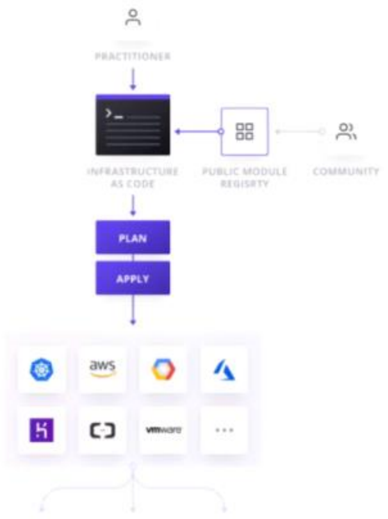
- **IaC** is the process of managing infrastructure in a file or files rather than manually configuring resources in a user interface.



How Terraform Works

- Terraform allows infrastructure to be expressed as code in a simple, human readable language called **HCL** (HashiCorp Configuration Language).

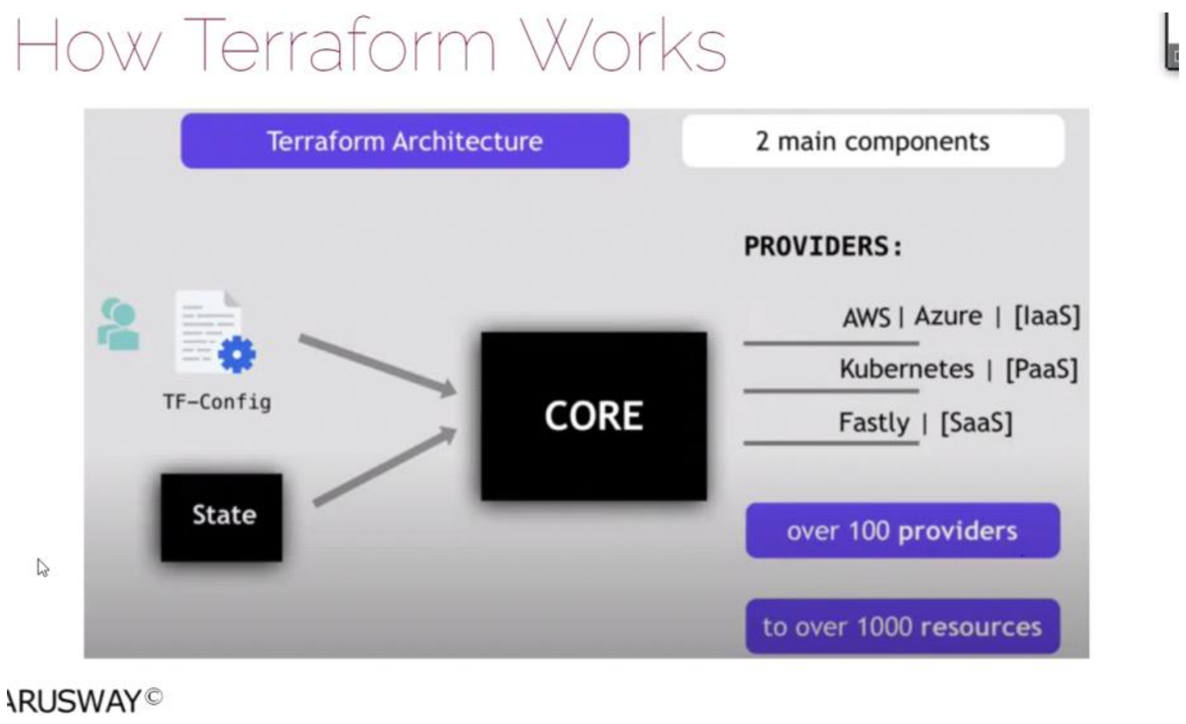
- Terraform reads configuration files and provides an execution plan of changes, which can be reviewed for safety and then applied and provisioned.



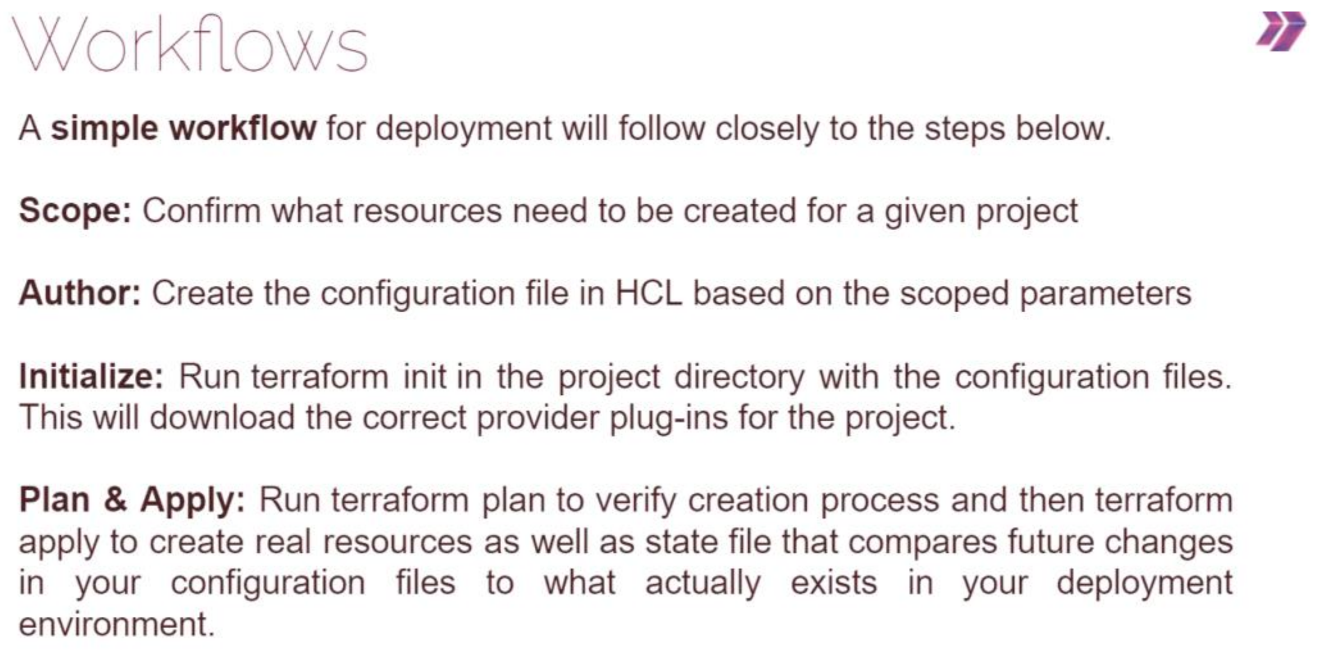
Iki ana yapi var biri provider digeri de core yani cekirdek.
- User bir config hazirlar Terraform a sunar. Terraform bunu okur gözden egecirir. Ve tekrardan bize bir excude plan cikarir. User bunu onaylar daha sonra kodu calistirir.
- Terraformda bunun neticesinde bir state file olusturur. Bu cok önemlidir

Tf uzantili bir file kuruyoruz

How Terraform Works



Is akisi



Workflows

A **simple workflow** for deployment will follow closely to the steps below.

Scope: Confirm what resources need to be created for a given project

Author: Create the configuration file in HCL based on the scoped parameters

Initialize: Run terraform init in the project directory with the configuration files. This will download the correct provider plug-ins for the project.

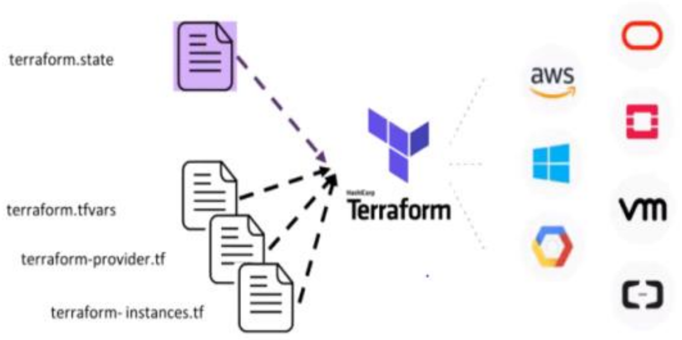
Plan & Apply: Run terraform plan to verify creation process and then terraform apply to create real resources as well as state file that compares future changes in your configuration files to what actually exists in your deployment environment.

Terraform icin bir zorunluluktur. Sizin hazirladiginiz plana bakarak Terraform mevcutta olan resourcelariniza bakar ve bunlardan planiniz icerisinde olan ve ihtiyaciniz olani size verir.

Terraform Elements

State

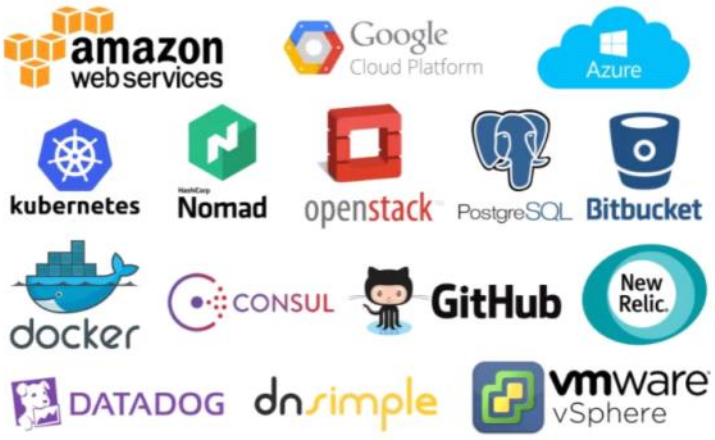
This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures. Terraform uses this local state to create plans and make changes to your infrastructure. State is a necessary requirement for Terraform to function.



Terraform Elements

Providers

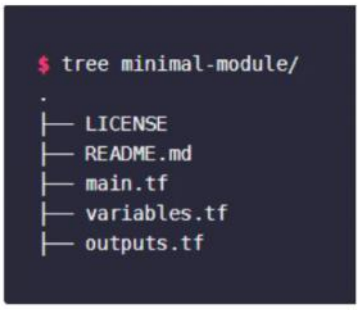
A provider is responsible for understanding API interactions and exposing resources. Every Terraform provider has its own documentation, describing its resource types and their arguments.



Terraform Elements

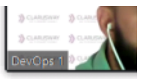
Modules

A Terraform module is a set of Terraform configuration files in a single directory. Even a simple configuration consisting of a single directory with one or more «.tf» files is a module. When you run Terraform commands directly from such a directory, it is considered the root module. So in this sense, every Terraform configuration is part of a module.



```
$ tree minimal-module/
.
├── LICENSE
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
```

Terraform Elements



Backends

A "backend" in Terraform determines how state is loaded and how an operation such as <apply> is executed. By default, Terraform uses the "local" backend, which is the normal behavior of Terraform you're used to. Backends are completely optional. You can successfully use Terraform without ever having to learn or use backends. Backends are used for keeping sensitive information off disk.

Advantages of Terraform

Platform Agnostic

- In a modern datacenter, you may have several different clouds and platforms to support your various applications.
- With Terraform, you can manage a **heterogeneous environment** with the same workflow by creating a configuration file to fit the needs of your project or organization.

Terraform bir cok Cloud Firmasinda calisabiliyor olmaz ise islem yapmiyor

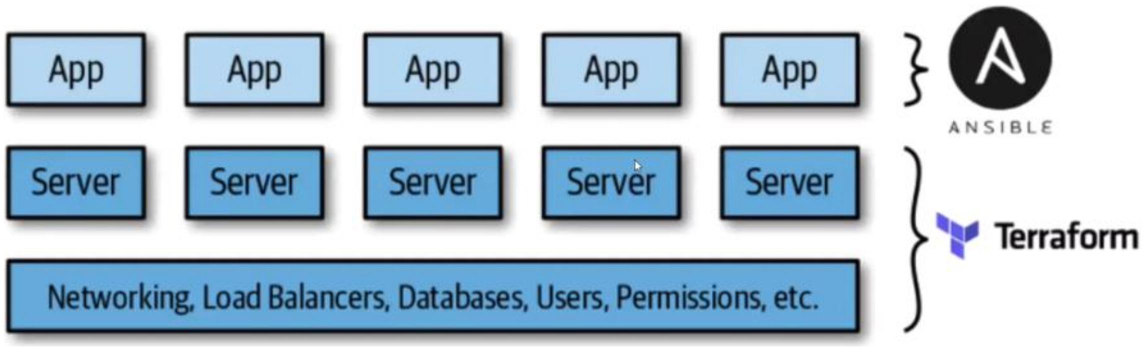
State Management

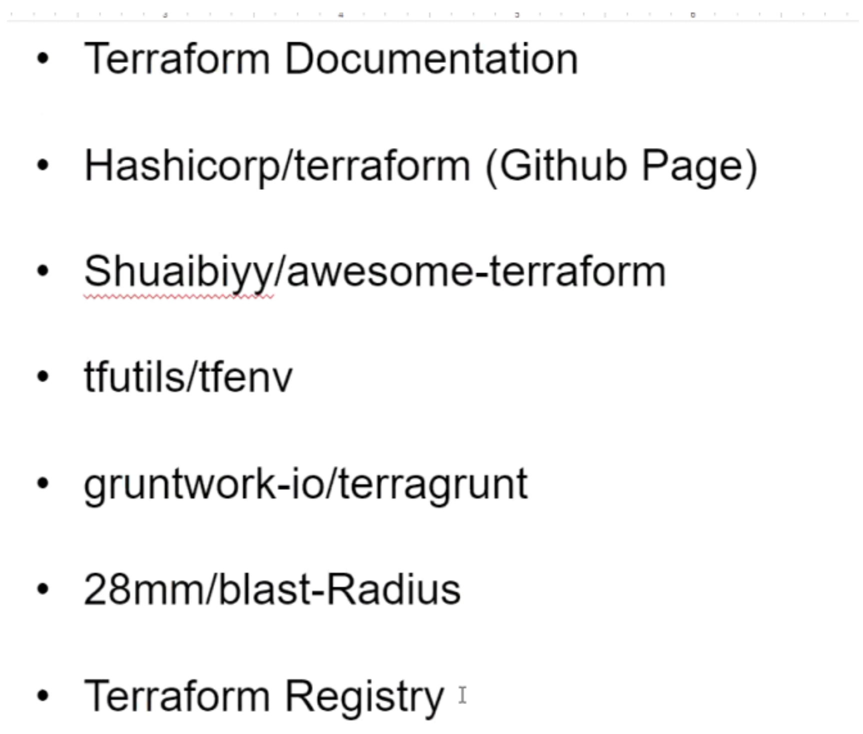
- Terraform creates **a state file** when a project is first initialized. Terraform uses this local state to create plans and make changes to your infrastructure.
- Prior to any operation, Terraform does a refresh to update the state with the real infrastructure. This means that Terraform state is the source of truth by which configuration changes are measured.
- If a change is made or a resource is appended to a configuration, Terraform compares those changes with the state file to determine what changes result in a new resource or resource modifications.

Operator Confidence

- The workflow built into Terraform aims to instill confidence in users by promoting easily repeatable operations and a planning phase to allow users to ensure the actions taken by Terraform will not cause disruption in their environment.
- Upon **terraform apply**, the user will be prompted to review the proposed changes and must affirm the changes or else Terraform will not apply the proposed plan.

Terraform & Ansible


App
App
App
App
App
ANSIBLE
Server
Server
Server
Server
Server
Terraform
Networking, Load Balancers, Databases, Users, Permissions, etc.



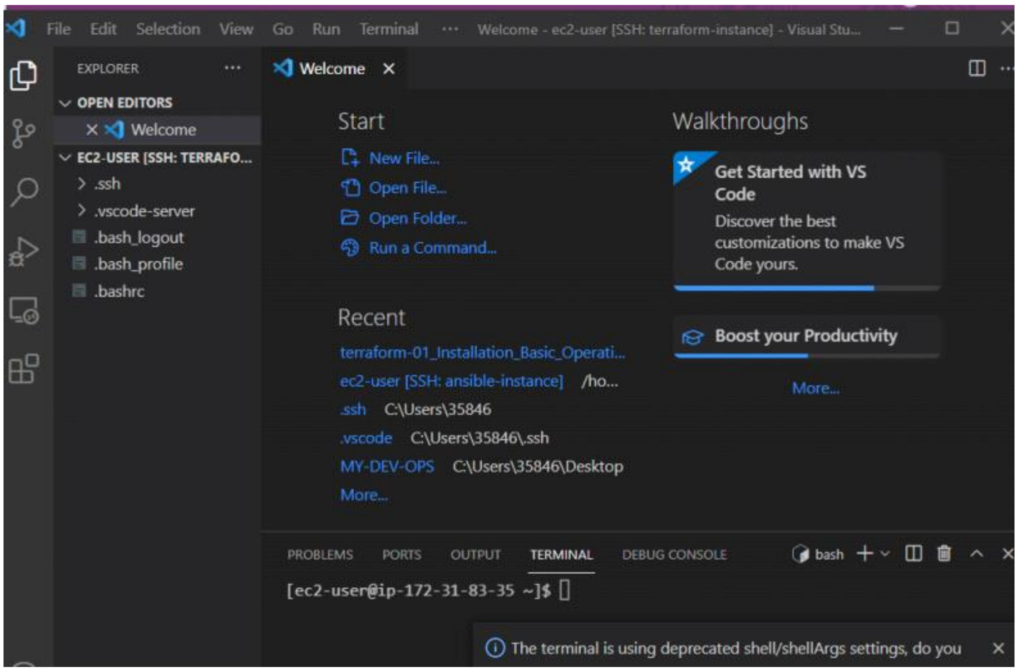
- Terraform Documentation
- Hashicorp/terraform (Github Page)
- Shuaibiyy/awesome-terraform
- tfutils/tfenv
- gruntwork-io/terragrunt
- 28mm/blast-Radius
- Terraform Registry

*******Terraform ile ilgili daha fazla bilgi edinilebilecek kaynaklar ********

Kisaca özetleyecek olursak Bizden bir Task isteniyor AWS den veyahut her hangi bir cloud dan bir server kurmamiz istendi ve ayrica Vpc ve Private ve Public Subnetler olusturmamiz istendi. Medya Filelarimiz s3 bucket ta olusmasi istendi .

Bizde tf uzantili bir Terraform kurduk file hazirladik ve init ettik daha sonra sisitemi olustrduk.

```
# Hands-on Terraform-01 : Terraform Installation and Basic Operations

Purpose of the this hands-
on training is to give students the knowledge of basic operations in Terraform.

## Learning Outcomes

Ssh ile baglandik
```


File Edit Selection View Go Run Terminal ··· Welcome - ec2-user [SSH: terraform-instance] - Visual Stu...
EXPLORER
OPEN EDITORS
Welcome
EC2-USER [SSH: TERRAFO...
.ssh
.vscode-server
.bash_logout
.bash_profile
.bashrc
Start
New File...
Open File...
Open Folder...
Run a Command...
Recent
terraform-01_Installation_Basic_Operati...
ec2-user [SSH: ansible-instance] /ho...
.ssh C:\Users\35846
.vscode C:\Users\35846\.ssh
MY-DEV-OPS C:\Users\35846\Desktop
More...
Walkthroughs
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Boost your Productivity
More...
PROBLEMS PORTS OUTPUT TERMINAL DEBUG CONSOLE
bash

```
[ec2-user@ip-172-31-83-35 ~]$
```

The terminal is using deprecated shell/shellArgs settings, do you

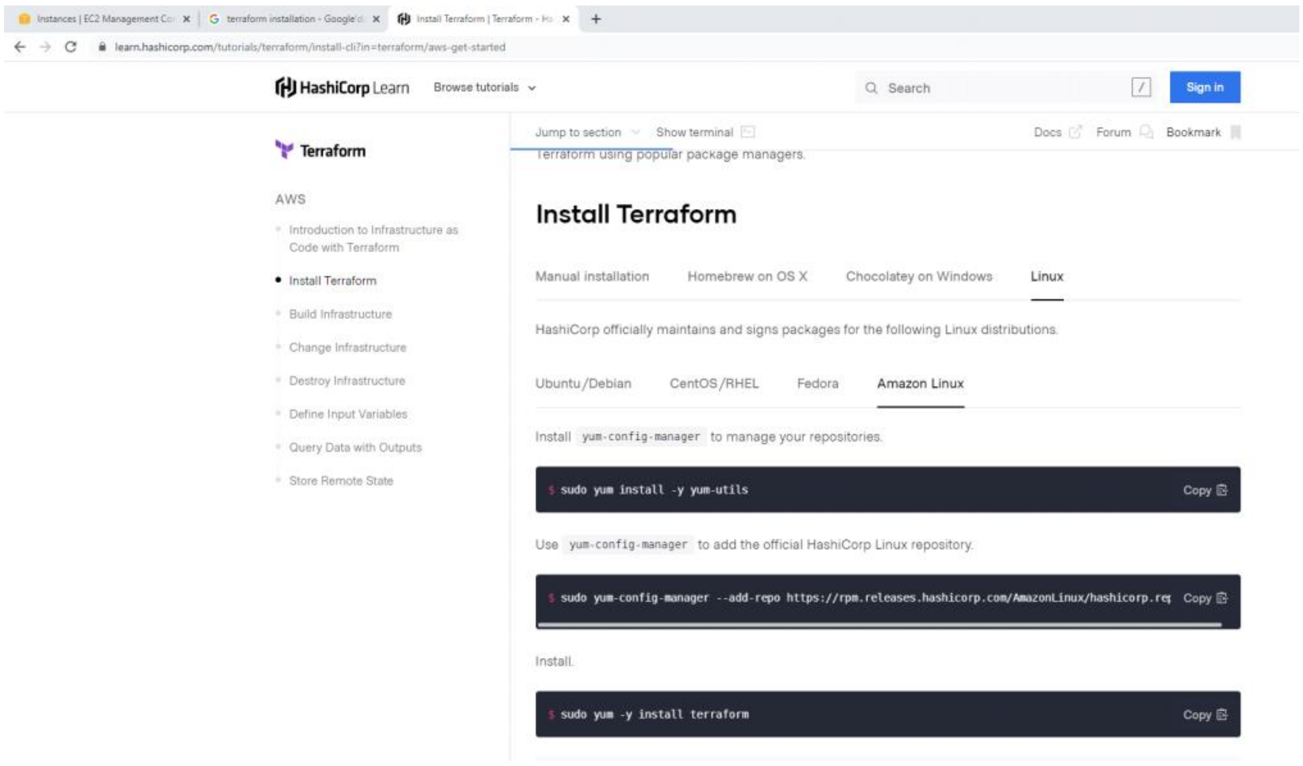
At the end of the this hands-on training, students will be able to;
- Install Terraform
- Build AWS Infrastructure with Terraform

## Outline
- Part 1 - Install Terraform
- Part 2 - Build AWS Infrastructure with Terraform

## Part 1 - Install Terraform
- Launch an EC2 instance using the Amazon Linux 2 AMI with security group allowing SSH connections.
- Connect to your instance with SSH.

```bash
ssh -i .ssh/call-training.pem ec2-user@ec2-3-133-106-98.us-east-2.compute.amazonaws.com
```

-
Nete instance terraform ve instal yazdigimizda karsimiza asagidaki link geliyor ve kodzxlari alip ekrana giriyoruz


learn.hashicorp.com/tutorials/terraform/install-cli?in=terraform/aws-get-started

Terraform

AWS

- Introduction to Infrastructure as Code with Terraform
- Install Terraform
- Build Infrastructure
- Change Infrastructure
- Destroy Infrastructure
- Define Input Variables
- Query Data with Outputs
- Store Remote State

Terraform using popular package managers.

Install Terraform

Manual installation | Homebrew on OS X | Chocolatey on Windows | Linux

HashiCorp officially maintains and signs packages for the following Linux distributions.

Ubuntu/Debian | CentOS/RHEL | Fedora | Amazon Linux

Install `yum-config-manager` to manage your repositories.

```
$ sudo yum install -y yum-utils
```

Use `yum-config-manager` to add the official HashiCorp Linux repository.

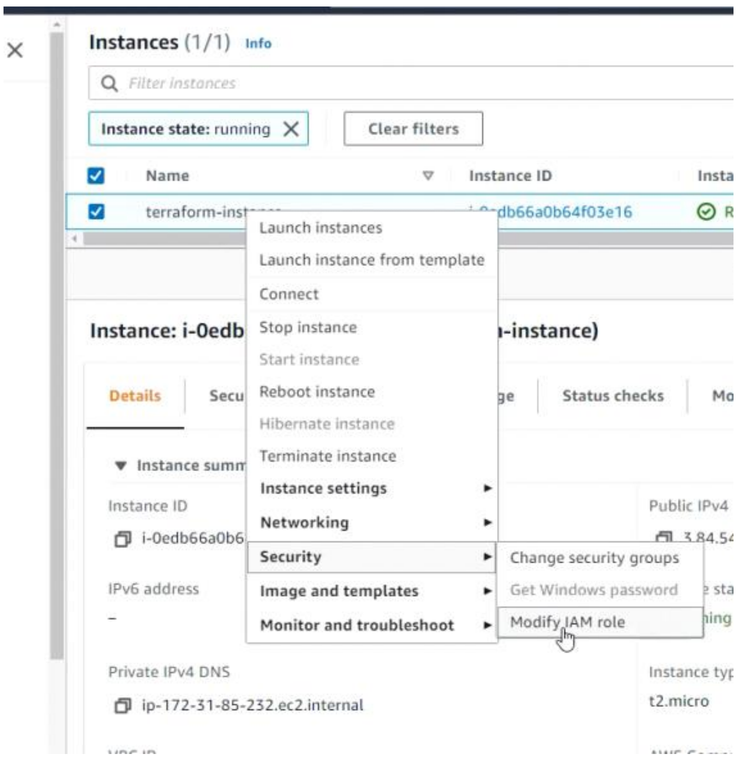
```
$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.rep
```

Install.

```
$ sudo yum -y install terraform
```

Update the installed packages and package cache on your instance.
```bash
$ sudo yum update -y
```
- Install yum-config-manager to manage your repositories.
```bash
$ sudo yum install -y yum-utils
```
- Use yum-config-manager to add the official HashiCorp Linux repository to the directory of /etc/yum.repos.d.
```bash
$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
```
- Install Terraform.
```bash
$ sudo yum -y install terraform
```
- Verify that the installation
```bash
$ terraform version
```
- list Terraform's available subcommands.
```bash
$ terraform -help
Usage: terraform [-version] [-help] <command> [args]
The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you are just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.
Common commands:
    apply              Builds or changes infrastructure
...
```
- Add any subcommand to terraform -help to learn more about what it does and available options.
```bash
```

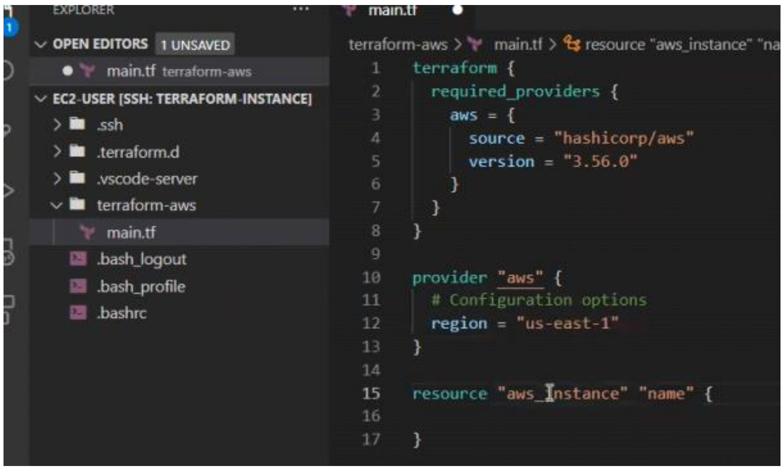
~~~
$ terraform -help apply ------genel olarak özelliklerini alabiliyruz
or
$ terraform apply -help --- bu komutun özelde anlatiyor
```


## Part 2 - Build AWS Infrastructure with Terraform
### Prerequisites
- An AWS account.
- The AWS CLI installed.
- Your AWS credentials configured locally.
```bash
$ aws configure
```
- Hard-coding credentials into any Terraform configuration is not recommended, and risks secret leakage should this file ever be committed to a public version control system. Using AWS credentials in EC2 instance is not recommended.
- We will use IAM role (temporary credentials) for accessing your AWS account.
### Create a role in IAM management console.

- Secure way to make API calls is to create a role and assume it. It gives temporary credentials for access your account and makes API calls.
- Go to the IAM service, click "roles" in the navigation panel on the left then click "create role".
- Under the use cases, Select `EC2`, click "Next Permission" button.
- In the search box write EC2 and select `AmazonEC2FullAccess` then click "Next: Tags" and "Next: Reviews".
- Name it `terraform`.
- Attach this role to your EC2 instance.

Aws configure de yazabiliriz veyahut terminalden bir rol de olusturabiliriz
~~~



~~~
### Write your first configuration
- The set of files used to describe infrastructure in Terraform is known as a Terraform configuration. You'll write your first configuration file to launch a single AWS EC2 instance.
- Each configuration should be in its own directory. Create a directory ("terraform-aws") for the new configuration and change into the directory.
```bash
~~~

```
$ mkdir terraform-aws && cd terraform-aws && touch main.tf
```
- Create a file named `main.tf` for the configuration code and copy and paste the following content.
```txt
provider "aws" {
  region  = "us-east-1"
}
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.38.0"
    }
  }
}
resource "aws_instance" "tf-ec2" {
  ami           = "ami-0742b4e673072066f"
  instance_type = "t2.micro"
  tags = {
    "Name" = "created-by-tf"
  }
}
```
- Install the HashiCorp Terraform extension in VSCode.
- Explain the each block via the following section.

Su an config by olusturmaya calisiyoruz< 3 ayri blok var
- Provider blogu
- Terraform blogu
- Provider blogu ---anlayan kisimdir isi yapan kisim resource kismidir.
- Resource blogu   --- ne olusturmak istiyorsak onu yazacagiz instance , s3bucket vs name de terraform daki lokal ismi



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.56.0"
    }
  }
}

provider "aws" {
  # Configuration options
  region = "us-east-1"
}

resource "aws_instance" "name" {

}
```

### Providers

The `provider` block configures the name of provider, in our case `aws`, which is responsible for creating and managing resources. A provider is a plugin that Terraform uses to translate the API interactions with the service. A provider is responsible for understanding API interactions and exposing resources. Because Terraform can interact with any API, you can represent almost any infrastructure type as a resource in Terraform.
The `profile` attribute in your provider block refers Terraform to the AWS credentials stored in your AWS Config File, which you created when you configured the AWS CLI. HashiCorp recommends that you never hard-code credentials into `*.tf configuration files`.
- Note: If you delete your AWS credentials from provider block, Terraform will automatically search for saved API credentials (for example, in ~/.aws/credentials) or IAM instance profile credentials.

### Resources

The `resource` block defines a piece of infrastructure. A resource might be a physical component
such as an EC2 instance.
The resource block must have two required data for EC2. : the resource type and the resource name
. In the example, the resource type is `aws_instance` and the local name is `tf-ec2
`. The prefix of the type maps to the provider. In our case "aws_instance" automatically tells Te
rraform that it is managed by the "aws" provider.
The arguments for the resource are within the resource block. The arguments could be things like
machine sizes, disk image names, or VPC IDs. For your EC2 instance, you specified an AMI for `Ama
zon Linux 2` and instance type will be `t2.micro`.
![terraform-workflow](terraform-workflow.png)

### Initialize the directory

When you create a new configuration you need to initialize the directory with `terraform init`.
- Initialize the directory.

```bash
$ terraform init ----- baslatama komutuidur
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.9.0"...
- Installing hashicorp/aws v3.9.0...
- Installed hashicorp/aws v3.9.0 (signed by HashiCorp)
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

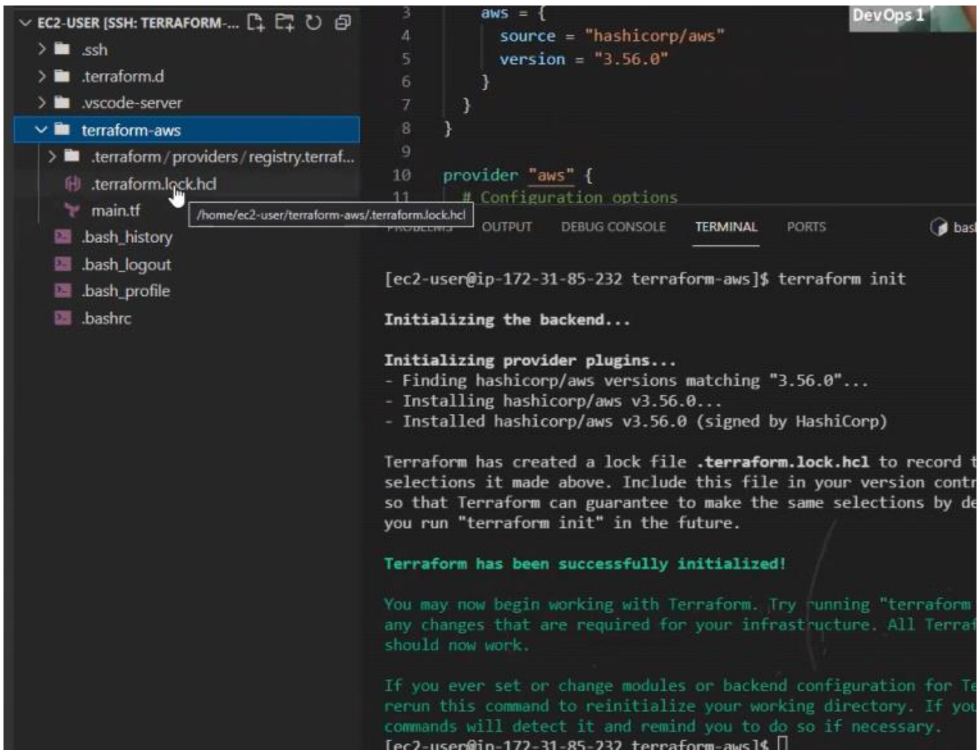
Terraform downloads the `aws` provider and installs it in a hidden subdirectory (.terraform) of t
he current working directory. The output shows which version of the plugin was installed.

Su ana kadar ne yaptik .tf uzantili bir config file olusturduk. Yapilandirma dosyasi

Resource; --kurmak istedigimiz kaynak ---trf icindeki ismi ne olsun

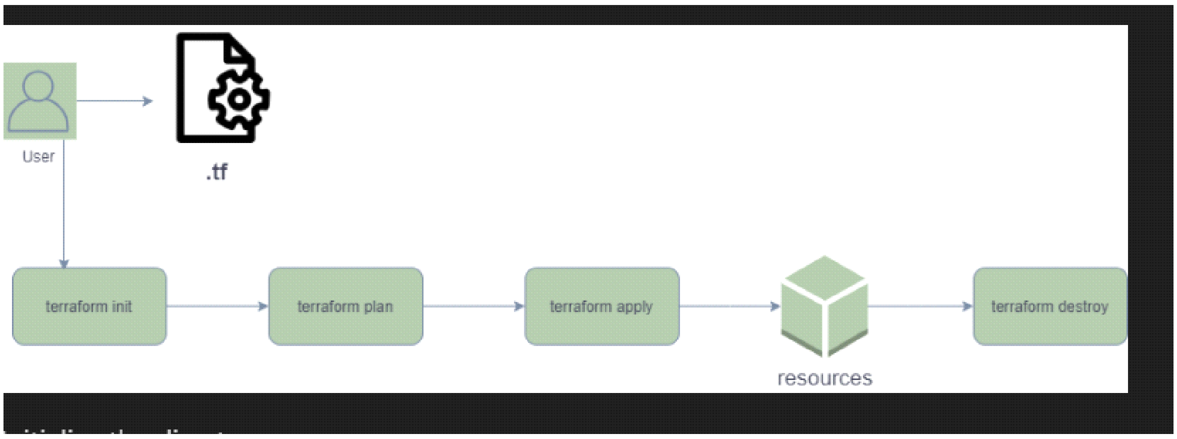
ami sini ve tag ini verdik daha sonra terraform init diyerek baslattik

lock.hcl isimli bir versiyonnk bir dosya olusturdu bir sene sonarada islem yapan dadzm
guncellemelerden dolayi bir problem yasamiyor bu sekilde

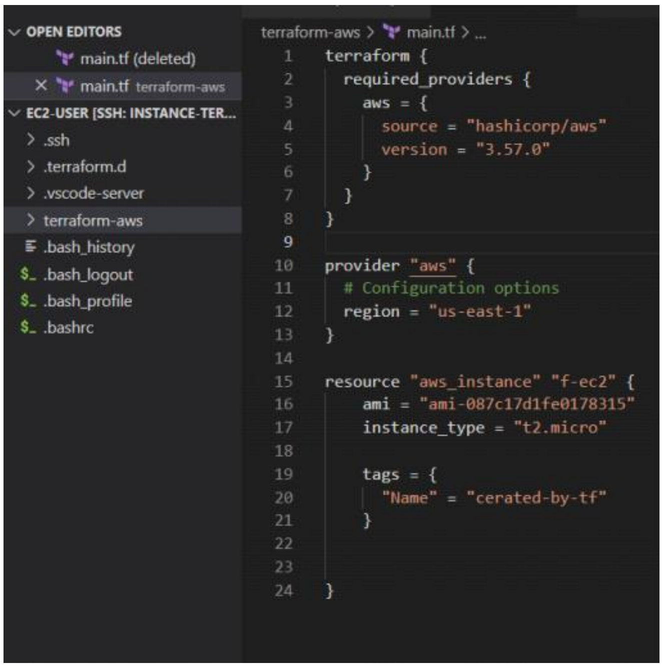


```
      aws = {
        source = "hashicorp/aws"
        version = "3.56.0"
      }
    }
  }

  provider "aws" {
    # Configuration options
```

```
[ec2-user@ip-172-31-85-232 terraform-aws]$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.56.0"...
- Installing hashicorp/aws v3.56.0...
- Installed hashicorp/aws v3.56.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record t
selections it made above. Include this file in your version contr
so that Terraform can guarantee to make the same selections by de
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform
any changes that are required for your infrastructure. All Terraf
should now work.

If you ever set or change modules or backend configuration for Te
rerun this command to reinitialize your working directory. If you
commands will detect it and remind you to do so if necessary.
[ec2-user@ip-172-31-85-232 terraform-aws]$ 
```



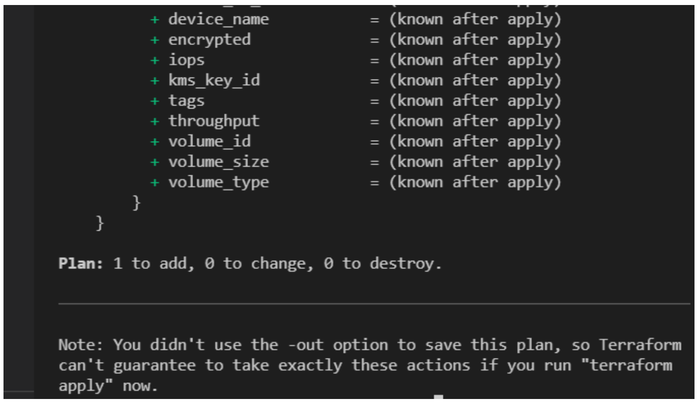
Tf uzantili bir config file yazdik, burada ne olusturmayi hedefliyorsak yaziyoruz load balancer mki veyahut terraform a init ettik simdi terraform plan diýecegiz



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.57.0"
    }
  }
}

provider "aws" {
  # Configuration options
  region = "us-east-1"
}

resource "aws_instance" "f-ec2" {
    ami = "ami-087c17d1fe0178315"
    instance_type = "t2.micro"

    tags = {
      "Name" = "cerated-by-tf"
    }


}
```

en son file mizin halli bu oldu ve

Terraform plan dedik eve dosyamizin son halini gördük



```
          + device_name              = (known after apply)
          + encrypted                = (known after apply)
          + iops                     = (known after apply)
          + kms_key_id               = (known after apply)
          + tags                     = (known after apply)
          + throughput               = (known after apply)
          + volume_id                = (known after apply)
          + volume_size              = (known after apply)
          + volume_type              = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

─────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform
can't guarantee to take exactly these actions if you run "terraform
apply" now.
```

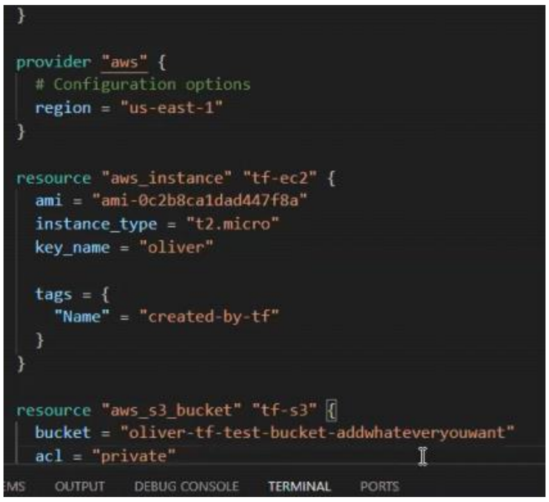
### Create infrastructure

- run `terraform plan`. You should see an output similar to the one shown below.

```bash
$ terraform plan
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create
Terraform will perform the following actions:
  # aws_instance.sample-resource will be created
  + resource "aws_instance" "tf-ec2" {
      + ami                          = "ami-04d29b6f966df1537"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      + network_interface_id         = (known after apply)
      + outpost_arn                  = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (known after apply)
      + primary_network_interface_id = (known after apply)
      + private_dns                  = (known after apply)
      + private_ip                   = (known after apply)
      + public_dns                   = (known after apply)
      + public_ip                    = (known after apply)
      + security_groups              = (known after apply)
      + source_dest_check            = true
      + subnet_id                    = (known after apply)
      + tenancy                      = (known after apply)
      + volume_tags                  = (known after apply)
      + vpc_security_group_ids       = (known after apply)
      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
      + ephemeral_block_device {
          + device_name  = (known after apply)
          + no_device    = (known after apply)
          + virtual_name = (known after apply)
        }
      + metadata_options {
          + http_endpoint               = (known after apply)
          + http_put_response_hop_limit = (known after apply)
          + http_tokens                 = (known after apply)
        }
      + network_interface {
          + delete_on_termination = (known after apply)
          + device_index          = (known after apply)
          + network_interface_id  = (known after apply)
        }
      + root_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }
Plan: 1 to add, 0 to change, 0 to destroy.
------------------------------------------------------------------------
Note: You didn't specify an "-out" parameter to save this plan, so Terraform can't guarantee that exactly these actions will be performed if "terraform apply" is subsequently run.
```
- This output shows the execution plan, describing which actions Terraform will take in order to change real infrastructure to match the configuration.
- Run `terraform apply`. You should see an output similar to the one shown above.
```bash
terraform apply
```
- Terraform will wait for your approval before proceeding. If anything in the plan seems incorrect it is safe to abort (ctrl+c) here with no changes made to your infrastructure.
- If the plan is acceptable, type "yes" at the confirmation prompt to proceed. Executing the plan will take a few minutes since Terraform waits for the EC2 instance to become available.
```txt
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
  Enter a value: yes
aws_instance.tf-example-ec2: Creating...
aws_instance.tf-example-ec2: Still creating... [10s elapsed]
aws_instance.tf-example-ec2: Still creating... [20s elapsed]
aws_instance.tf-example-ec2: Still creating... [30s elapsed]
aws_instance.tf-example-ec2: Still creating... [40s elapsed]
aws_instance.tf-example-ec2: Creation complete after 43s [id=i-080d16db643703468]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```
- Visit the EC2 console to see the created EC2 instance.

### Inspect state

- When you applied your configuration, Terraform fetched data from resources into a file called terraform.tfstate. It keeps track of resources' metadata.

### Manually Managing State
- Terraform has a command called `terraform state` for advanced state management. For example, if
 you have a long state file (detailed) and you just want to see the name of your resources, which
 you can get them by using the `list` subcommand.
~~~bash
$ terraform state list
aws_instance.tf-ec2
``

### Creating a AWS S3 bucket


}

provider "aws" {
  # Configuration options
  region = "us-east-1"
}

resource "aws_instance" "tf-ec2" {
  ami = "ami-0c2b8ca1dad447f8a"
  instance_type = "t2.micro"
  key_name = "oliver"

  tags = {
    "Name" = "created-by-tf"
  }
}

resource "aws_s3_bucket" "tf-s3" {
  bucket = "oliver-tf-test-bucket-addwhateveryouwant"
  acl = "private"

*****

S3 bucketimizi da yazdik, kaydedttik  ve  terraform init yapmayacagiz cunku dazha önce
baslatmistik  yeni bir provider eklemedik yeni bir back end eklemedik bundan dolayi tekrar
Terraform init dememize gerek yok

**** terraform plan
**** ardindan terraform apply diyoruz

- Create a S3 bucket. Go to the `main.tf` and add the followings.
```tf
provider "aws" {
  region  = "us-east-1"
}
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.38.0"
    }
  }
}
resource "aws_instance" "tf-ec2" {
  ami           = "ami-0742b4e673072066f"
  instance_type = "t2.micro"
  key_name      = "mk"    # write your pem file without .pem extension>
  tags = {
    "Name" = "tf-ec2"
  }
}
resource "aws_s3_bucket" "tf-s3" {
  bucket = "oliver-tf-test-bucket-addwhateveryouwant"
  acl    = "private"
}
~~~

~~~
- Write your pem file without .pem extension and change the "addwhateveryouwant" part of the bucket name. Because bucket name must be unique.
- Run the command `terraform plan` and `terraform apply`.
```bash
terraform plan
terraform apply
~~~
```txt
Error: Error creating S3 bucket: AccessDenied: Access Denied
        status code: 403, request id: 8C5E290CD1CD3F71, host id: NT6nPSh0nW9rripGZrOAo48qJpZ2yToKCiGxDl6gfKIXY97uVH67lcvBiQjX9bsJRX3cL1oNVNM=
```
- Attach S3FullAccess policy to the "terraform" role.
```bash
terraform apply -auto-approve
```
- `-auto-approve` means to skip the approval of plan before applying.
- Go to the AWS console, check the S3 bucket. Then check the `terraform.tfstate` and `terraform.tfstate.backup` file.
```bash
terraform plan -out=justs3
```
Filemizda bir cok resource oldugu taktirde, istedigimiz kaynaklarimizi baska bir file altina alabiliyoruz örnegin dosyamizda bulunan bazi resourcelari silmek istedik ancak bazilarinin silinmesini istemedik destroy komutundan etkilenmesini eistemedik bunu icin uyguluyoruz

- Now we have just an S3 bucket in justs3. Check that `terraform.tfstate` file has both ec2 and s3 bucket (real infrastructure). If we apply justs3 file it will delete the EC2 instance and modify the tfstate file. You can save your plans with -out flag.
```bash
terraform apply justs3
```
### Terraform Commands

- Validate command.
- Go to the terminal and run `terraform validate`. It validates the Terraform files syntactically correct and internally consistent.
```bash
terraform validate
```
Sizin filenizin incelerek ve gecerli bir file mi degil mi onu söyler terraform plan demeden yapabiliriz

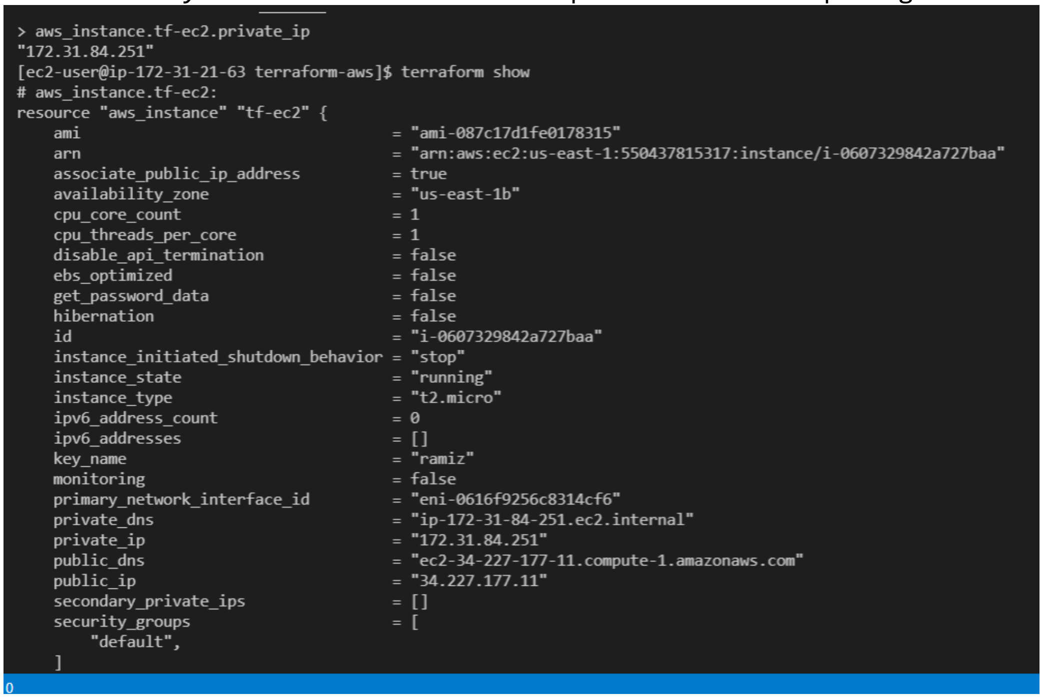
- Go to `main.tf` file and delete last curly bracket "}" and key_name's last letter (key_nam). And Go to terminal and run the command `terraform validate`. After taking the errors correct them. Then run the command again.
```bash
$ terraform validate
Error: Argument or block definition required
  on tf-example.tf line 20, in resource "aws_s3_bucket" "tf-s3":
An argument or block definition is required here.
$ terraform validate
Error: Unsupported argument
  on tf-example.tf line 9, in resource "aws_instance" "tf-example-ec2":
   9:     key_nam      = "mk"
An argument named "key_nam" is not expected here. Did you mean "key_name"?
$ terraform validate
Success! The configuration is valid.
```
- Go to `main.tf` file and copy the EC2 instance block and paste it. And Go to terminal and run the command `terraform validate`. After taking the errors correct them. Then run the command again.
$ terraform validate

```
│ Error: Duplicate resource "aws_instance" configuration
│
│   on main.tf line 24:
│   24: resource "aws_instance" "tf-ec2" {
│
│ A aws_instance resource named "tf-
ec2" was already declared at main.tf:15,1-33. Resource names must be unique per type in
│ each module.
```

- fmt command.
- Go to `main.tf` file and delete the second EC2 instance.
- Go to `main.tf` file and add random indentations. Then go to terminal and run the command `terraform fmt`. "terraform fmt" command reformat your configuration file in the standard style.

```bash
terraform fmt -----  file da bazi bosluk sorunlari var ise onlari duzenliyor
```

- Now, show `main.tf` file. It was formatted again.
- Go to the terminal and run `terraform console`.This command provides an interactive command-line console for evaluating and experimenting with expressions. This is useful for testing interpolations before using them in configurations, and for interacting with any values currently saved in state. You can see the attributes of resources in tfstate file and check built in functions before you write in your configuration file. Lets create a file under the terraform-aws directory and name it "cloud" and paste "hello devops engineers". Run the following commands.

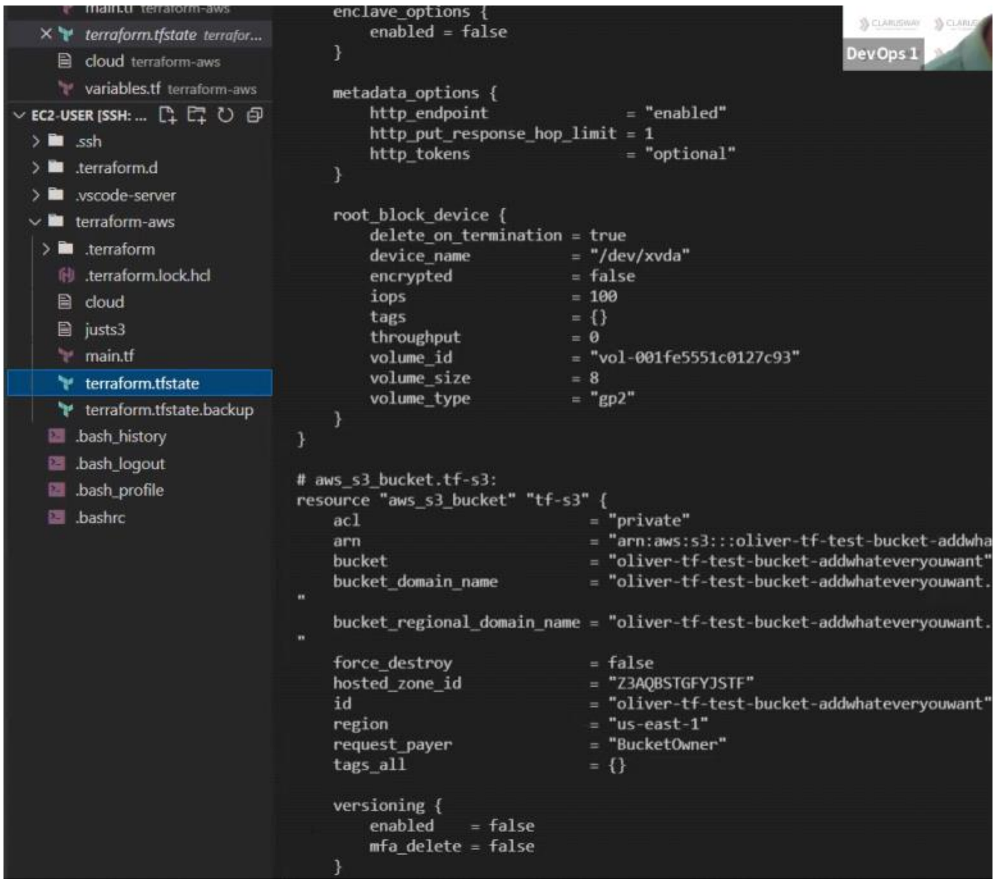


```
> aws_instance.tf-ec2.private_ip
"172.31.84.251"
[ec2-user@ip-172-31-21-63 terraform-aws]$ terraform show
# aws_instance.tf-ec2:
resource "aws_instance" "tf-ec2" {
    ami                                  = "ami-087c17d1fe0178315"
    arn                                  = "arn:aws:ec2:us-east-1:550437815317:instance/i-0607329842a727baa"
    associate_public_ip_address          = true
    availability_zone                    = "us-east-1b"
    cpu_core_count                       = 1
    cpu_threads_per_core                 = 1
    disable_api_termination              = false
    ebs_optimized                        = false
    get_password_data                    = false
    hibernation                          = false
    id                                   = "i-0607329842a727baa"
    instance_initiated_shutdown_behavior = "stop"
    instance_state                       = "running"
    instance_type                        = "t2.micro"
    ipv6_address_count                   = 0
    ipv6_addresses                       = []
    key_name                             = "ramiz"
    monitoring                           = false
    primary_network_interface_id         = "eni-0616f9256c8314cf6"
    private_dns                          = "ip-172-31-84-251.ec2.internal"
    private_ip                           = "172.31.84.251"
    public_dns                           = "ec2-34-227-177-11.compute-1.amazonaws.com"
    public_ip                            = "34.227.177.11"
    secondary_private_ips                = []
    security_groups                      = [
        "default",
    ]
0
```

Console komutu ile konsola gidip olusturdugumuz file larin icerisine gidebiliyor file icerigini gorebiliyoruz

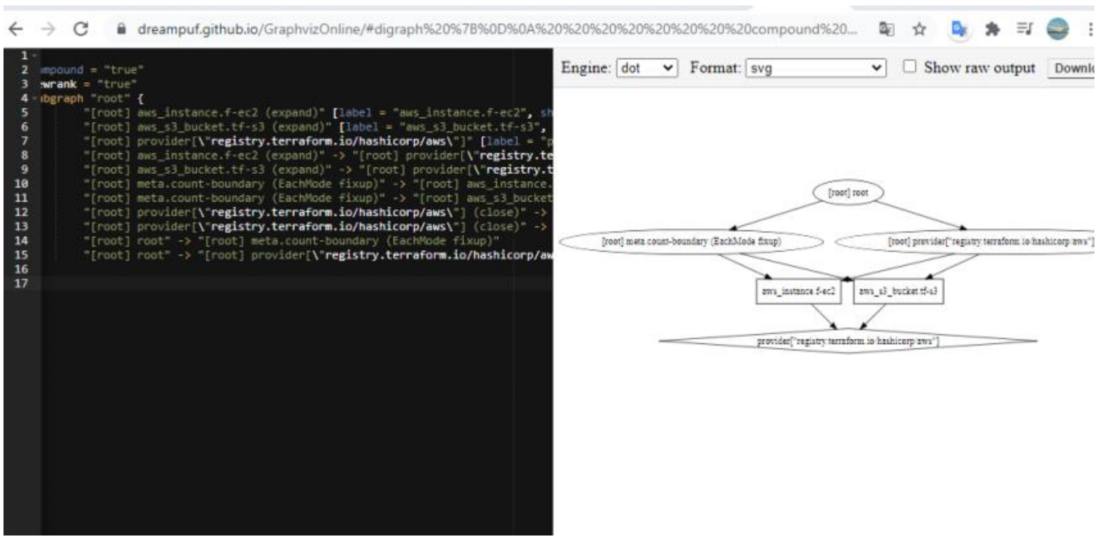
```bash
terraform console
> aws_instance.tf-ec2
> aws_instance.tf-ec2.private_ip
> min (1,2,3)
> lower("HELLO")
> file("${path.module}/cloud")
> aws_s3_bucket.tf-s3
> aws_s3_bucket.tf-s3.bucket
> exit or (ctrl+c)
```

- show command.
- Go to the terminal and run `terraform show`.
 You can see tfstate file or plan in the terminal. It is more readable than `terraform.tfstate`.
```bash
terraform show
```
terraform show form show --state file daha okunakli bir sekilde getirebiliyoruz



```
    enclave_options {
        enabled = false
    }

    metadata_options {
        http_endpoint               = "enabled"
        http_put_response_hop_limit = 1
        http_tokens                 = "optional"
    }

    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/xvda"
        encrypted             = false
        iops                  = 100
        tags                  = {}
        throughput            = 0
        volume_id             = "vol-001fe5551c0127c93"
        volume_size           = 8
        volume_type           = "gp2"
    }
}

# aws_s3_bucket.tf-s3:
resource "aws_s3_bucket" "tf-s3" {
    acl                         = "private"
    arn                         = "arn:aws:s3:::oliver-tf-test-bucket-addwha
    bucket                      = "oliver-tf-test-bucket-addwhateveryouwant"
    bucket_domain_name          = "oliver-tf-test-bucket-addwhateveryouwant.
"
    bucket_regional_domain_name = "oliver-tf-test-bucket-addwhateveryouwant.
"
    force_destroy               = false
    hosted_zone_id              = "Z3AQBSTGFYJSTF"
    id                          = "oliver-tf-test-bucket-addwhateveryouwant"
    region                      = "us-east-1"
    request_payer               = "BucketOwner"
    tags_all                    = {}

    versioning {
        enabled    = false
        mfa_delete = false
    }
```

- graph command.---state sekilsel olarak incelemeye yariyor cok buyuk yapilari uygulamada kullaniliyor internet adresine komutumuzu yapistirip nasil bir yapi oldugunu görebiliuyoruz
- Go to the terminal and run `terraform graph`. It creates a visual graph of Terraform resources. The output of "terraform graph" command is in the DOT format, which can easily be converted to an image by making use of dot provided by GraphViz.
- Copy the output and paste it to the ` https://dreampuf.github.io/GraphvizOnline `. Then display it. If you want to display this output in your local, you can download graphviz (`sudo yum install graphviz`) and take a `graph.svg` with the command `terraform graph | dot -Tsvg > graph.svg`.
```bash
terraform graph
```


dreampuf.github.io/GraphvizOnline/#digraph%20%7B%0D%0A%20%20%20%20%20%20%20%20compound%20...

```
mpound = "true"
ewrank = "true"
ubgraph "root" {
        "[root] aws_instance.f-ec2 (expand)" [label = "aws_instance.f-ec2", sh
        "[root] aws_s3_bucket.tf-s3 (expand)" [label = "aws_s3_bucket.tf-s3",
        "[root] provider[\"registry.terraform.io/hashicorp/aws\"]" [label = "p
        "[root] aws_instance.f-ec2 (expand)" -> "[root] provider[\"registry.te
        "[root] aws_s3_bucket.tf-s3 (expand)" -> "[root] provider[\"registry.t
        "[root] meta.count-boundary (EachMode fixup)" -> "[root] aws_instance.
        "[root] meta.count-boundary (EachMode fixup)" -> "[root] aws_s3_bucket
        "[root] provider[\"registry.terraform.io/hashicorp/aws\"] (close)" ->
        "[root] provider[\"registry.terraform.io/hashicorp/aws\"] (close)" ->
        "[root] root" -> "[root] meta.count-boundary (EachMode fixup)"
        "[root] root" -> "[root] provider[\"registry.terraform.io/hashicorp/aw
```

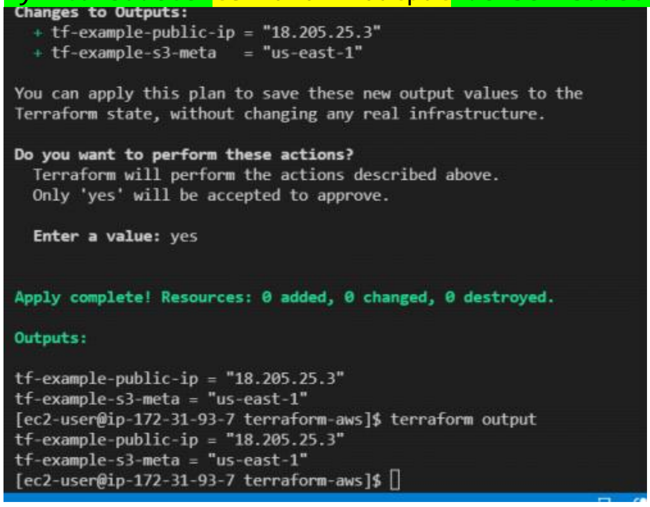
Engine: dot Format: svg Show raw output

Download
[root] root
[root] meta.count-boundary (EachMode fixup)
[root] provider["registry.terraform.io/hashicorp/aws"]
aws_instance.f-ec2
aws_s3_bucket.tf-s3
provider["registry.terraform.io/hashicorp/aws"]

- output command.
- Now add the followings to the `main.tf` file. Then run the commands `terraform apply or terraform refresh` and `terraform output`. `terraform output` command is used for reading an output from a state file. It reads an output variable from a Terraform state file and prints the value. With no additional arguments, output will display all the outputs for the (parent) root module. If NAME is not specified, all outputs are printed.

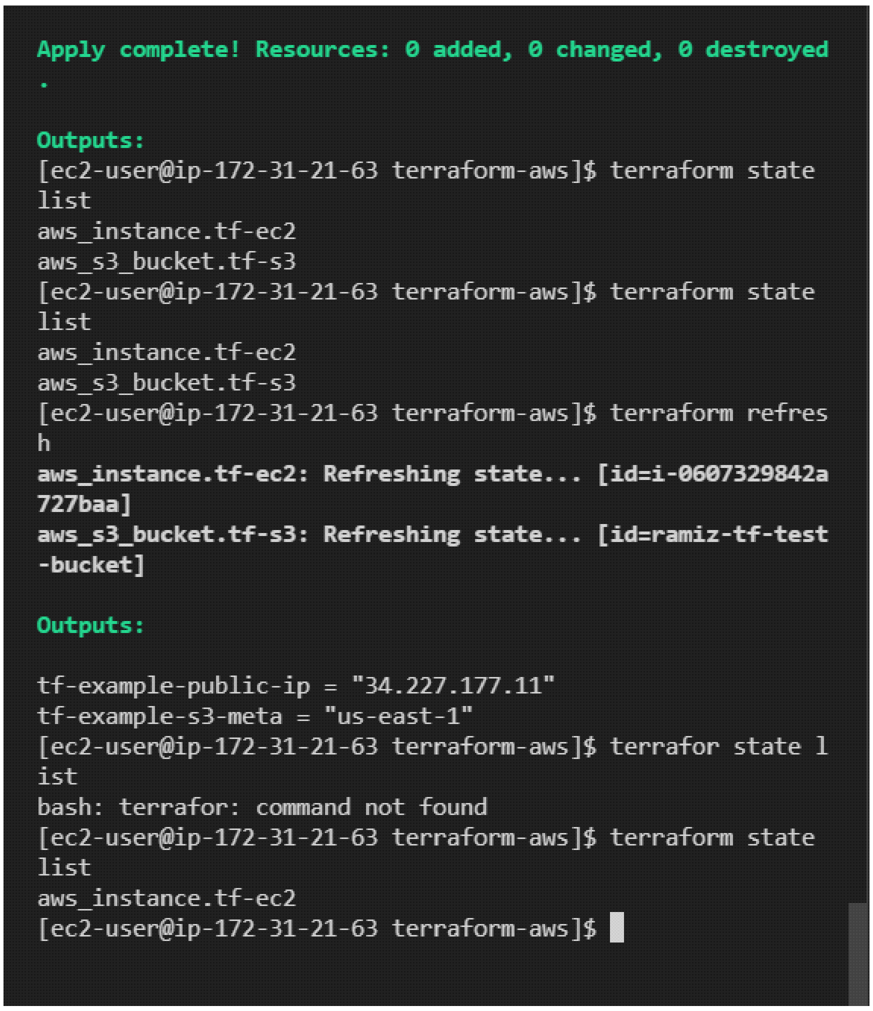
```
output "tf-example-s3-meta" {   ------- lokal isim naSIL ISTIYORSAK VEREBILIYORUZ
  value = aws_s3_bucket.tf-s3.region ----- VALUE DE NE ISTEYECEKSEK ONU YAZABILIYORUZ INSTANCE PUBLIC IP SINI ISTIYORUZ
} ----
output "tf-example-s3-meta" {
  value = aws_instance.f-ec2.public_ip
}
```

Daha sonra terraform apply ettik ve yapilan degisiklikleri butun terraform ile birlikte geldi Ayrica sadece terraform output dersek sadec son yaptigimiz out putlar gelecektir


```
Changes to Outputs:
  + tf-example-public-ip = "18.205.25.3"
  + tf-example-s3-meta   = "us-east-1"

You can apply this plan to save these new output values to the
Terraform state, without changing any real infrastructure.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes


Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

tf-example-public-ip = "18.205.25.3"
tf-example-s3-meta = "us-east-1"
[ec2-user@ip-172-31-93-7 terraform-aws]$ terraform output
tf-example-public-ip = "18.205.25.3"
tf-example-s3-meta = "us-east-1"
[ec2-user@ip-172-31-93-7 terraform-aws]$ 
```

```bash
output "tf-example-public-ip" {
  value = aws_instance.tf-ec2.public_ip
}
output "tf-example-s3-meta" {
  value = aws_s3_bucket.tf-s3.region
}


```

~~~
```bash
terraform apply
terraform output
terraform output -json
terraform output tf-example-public-ip
~~~
- refresh command.
- The `terraform refresh` command is used to update the state file with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file. First, check the current state of your resources with `terraform state list`. Then go to the AWS console and delete your S3 bucket `oliver-tf-test-bucket-addwhateveryouwant`. Display the state list again and refresh the state. Run the following commands.
```bash
$ terraform state list
aws_instance.tf-example-ec2
aws_s3_bucket.tf-example-s3
$ terraform state list
aws_instance.tf-example-ec2
aws_s3_bucket.tf-example-s3
$ terraform refresh
aws_instance.tf-example-ec2: Refreshing state... [id=i-02938164282a9c741]
aws_s3_bucket.tf-example-s3: Refreshing state... [id=oliver-tf-test-bucket]
Outputs:
tf-example-public-ip = "54.237.127.221"
tf-example-s3-meta = "us-east-1"
$ terraform state list
aws_instance.tf-example-ec2
```



```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed
.

Outputs:
[ec2-user@ip-172-31-21-63 terraform-aws]$ terraform state
list
aws_instance.tf-ec2
aws_s3_bucket.tf-s3
[ec2-user@ip-172-31-21-63 terraform-aws]$ terraform state
list
aws_instance.tf-ec2
aws_s3_bucket.tf-s3
[ec2-user@ip-172-31-21-63 terraform-aws]$ terraform refres
h
aws_instance.tf-ec2: Refreshing state... [id=i-0607329842a
727baa]
aws_s3_bucket.tf-s3: Refreshing state... [id=ramiz-tf-test
-bucket]

Outputs:

tf-example-public-ip = "34.227.177.11"
tf-example-s3-meta = "us-east-1"
[ec2-user@ip-172-31-21-63 terraform-aws]$ terrafor state l
ist
bash: terrafor: command not found
[ec2-user@ip-172-31-21-63 terraform-aws]$ terraform state
list
aws_instance.tf-ec2
[ec2-user@ip-172-31-21-63 terraform-aws]$
```

Refresh komutu ile gunceleme yaparak statemizin son halini göerebiliyoruz.

- Now, you can see the differences between files `terraform.tfstate` and `terraform.tfstate.backup`. From tfstate file S3 bucket is deleted but in backup file you can see the S3 bucket.
- Run terraform apply -auto-approve and create S3 bucket again.
```bash
terraform apply -auto-approve

Bu komut ile tekrardan s3 bucketimizi oluturuyoruz sildigimiz s3 bucket tekrardan geliyor
Satate olan degisiklikleri otomatik olarak uyguluyor
```
- Go to the `main.tf` file and make the changes. We will change the ami id with the Ubuntu instance. Also change the value of EC2 instance tag and name of the S3 bucket.
```bash
resource "aws_instance" "tf-ec2" {
    - ami           = "ami-0742b4e673072066f"
    + ami           = "ami-042e8287309f5df03"
    instance_type = "t2.micro"
    key_name      = "mk"    #<pem file>
    tags = {
      - Name = "tf-ec2"
      + Name = "tf-ec2-ubuntu"
  }
}
resource "aws_s3_bucket" "tf-s3" {
  + bucket = "oliver-tf-bucket-addwhateveryouwant-new"
  - bucket = "oliver-tf-bucket-addwhateveryouwant"
  acl    = "private"
}
```
- **Run the command** `terraform apply -auto-approve` and check the AWS console, the old S3 bucket and EC2 were destroyed and terraform created new ones.
```bash
terraform apply -auto-approve
```
Bu komut eger ki configurasyonda buyuk bir degisiklik olursa bunu uygular ubuntu nun ami sini eklemis tik esk instance sildi yerine yenisini kurdu

- Make the new changes in the `main.tf` file.
```bash
output "tf-example-private-ip" {
  value = aws_instance.tf-ec2.private_ip
}
```
- Run the command `terraform apply -refresh=false`.
Bu komut ile refresh etmeden sadece istedigimiz komutu uyguluyor bu uzun configurasyonlarda zamandankazanmak maksatlli uygulanabilir. Islemin daha kisa surmesi icin uygulanir

```bash
$ terraform apply -refresh=false
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
Terraform will perform the following actions:
Plan: 0 to add, 0 to change, 0 to destroy.
Changes to Outputs:
  + tf-example-private-ip = "172.31.22.95"
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
  Enter a value: yes
```

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```
```bash
terraform destroy
```

[Ayti.Tech | #15 – Terraform | Buluta Doğru – Feyizli Abilerden Bulut Sohbetleri]

Kisaca özetlersek ;

Ilk olarak Terraformu olusturdugumuz instance server a kurduk

- **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 -help**
- $ terraform -help apply
- $ terraform apply -help --- bu komutun özelde anlatiyor
- $ aws configure
- ### Create a role in IAM management console.

- $ **mkdir terraform-aws && cd terraform-aws && touch main.tf**

- $ terraform init ----- baslatama komutuidur Ayrica .terraform klasöru acar

- $ terraform plan
- $ terraform apply
- $ terraform state list

- terraform validate ---- **Sizin filenizin incelerek ve gecerli bir file mi degil mi onu söyler terraform plan demeden yapabiliriz**
- ```
- **Sizin filenizin incelerek ve gecerli bir file mi degil mi onu söyler terraform plan demeden yapabiliriz**

- **terraform fmt ----- file da bazi bosluk sorunlari var ise onlari duzenliyor**
- ```

- ************Terraform ile ilgili intelij idea yuklenebilir*************
- Console komutu ile ayri bir konsola aciyoruz ve olusturdugumuz file larin icerisine gidebiliyor file icerigini gorebiliyoruz

- ```bash
- terraform console
- > aws_instance.tf-ec2
- > aws_instance.tf-ec2.private_ip
- > min (1,2,3)
- > lower("HELLO")
- > file("${path.module}/cloud")
- > aws_s3_bucket.tf-s3
- > aws_s3_bucket.tf-s3.bucket
- > exit or (ctrl+c)

- aws_s3_bucket.tf-s3.bucket -----s3 bucketimizin ismini ögrenebiliyoruz
  ```

- terraform show ----state file daha okunakli bir sekilde gertirebiliyoruz

- - graph command.---state sekilsel olarak incelemeye yariyor cok buyuk yapilari uygulamada kullaniliyor internet adresine komutumuzu yapistirip nasil bir yapi oldugunu görebiliuyoruz

- https://dreampuf.github.io/GraphvizOnline

- $ terraform refresh **; Refresh komutu ile gunceleme yaparak statemizin son halini göerebiliyoruz. Örnegin konsolda silinen bir sey hala state de görunuyor olabilir ancak biz refresh yaparsak statenin son halini görebiliriz.**

- **terraform apply -auto-approve**

- Bu komut ile tekrardan s3 bucketimizi oluturuyoruz sildigimiz s3 bucket tekrardan geliyor

- **terraform apply -auto-approve**
- **```**
- **Bu komut eger ki configurasyonda buyuk bir degisiklik olursa bunu uygular ubuntu nun ami sini eklemis tik esk instance sildi yerine yenusuu kurdu**

- **`terraform apply -refresh=false`.**
- **Bu komut ile refresh etmeden sadece istedigimiz komutu uyguluyor bu uzun configurasyonlarda zamandankazanmak maksatlli uygulanabilir. Islemin daha kisa surmesi icin uygulanir**