Terraform Modules

Terraform Remote State

Overview

Modules are containers for multiple resources that are used together. A module consists of a collection of `.tf` and/or `.tf.json` files kept together in a directory. Modüller, birlikte kullanılan birden çok kaynagi içinde bulunduran konyteynirlardir. Modül, bir dizinde bir arada tutulan .tf ve/veya .tf.json dosyaları koleksiyonundan oluşur.

Modules are the main way to package and reuse resource configurations with Terraform. Modüller, kaynak yapılandırmalarını Terraform ile paketlemenin ve yeniden kullanmanın ana yoludur. Bir paket yada kalip gibi kaynak konfigürasyonlarini icinde barindirir. CloudFormation gibi dusunebiliriz.

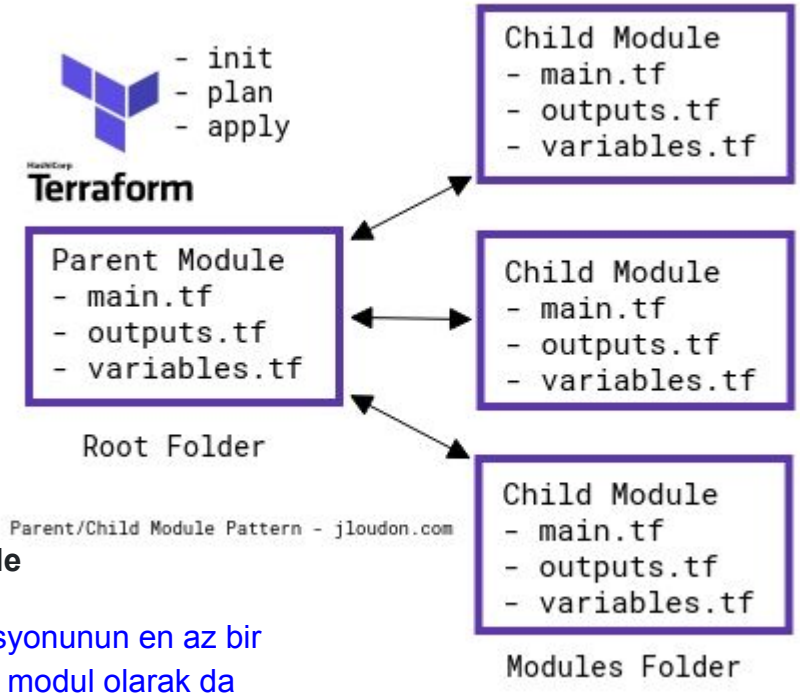
The Root Module

Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the `.tf` files in the main working directory. Her Terraform konfigürasyonunun, ana çalışma dizinindeki .tf dosyalarında tanımlanan kaynaklardan oluşan, kök modülü olarak bilinen en az bir modülü vardır. Bu ana klasor icerisinde bulunan .tf belgesinde tanimlanan kaynaklari barindirir

Child Modules

A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration. A module that has been called by another module is often referred to as a child module. Bir Terraform modülü (genellikle bir konfigürasyonun kök modülü), kaynaklarını konfigürasyona dahil etmek için diğer modülleri çağırabilir. Başka bir modül tarafından çağrılan bir modüle genellikle child modul (alt modül) denir.

Child modules can be called multiple times within the same configuration, and multiple configurations can use the same child module. Alt modüller aynı konfigürasyon içinde birden çok kez çağrılabilir ve birden çok konfigürasyon aynı alt modülü kullanabilir.



Child Modules

Başka bir modül tarafından çağrılan bir modüle genellikle alt modül denir. Alt modüller aynı konfigürasyon içinde birden çok kez çağrılabilir ve birden çok konfigürasyon aynı alt modülü kullanabilir.

The Root/Parent Module

Her Terraform konfigürasyonunun en az bir modulu vardir ve bu root modul olarak da bilinir. Root Module ana klasor icerisinde bulunan .tf belgeleri icerisinde tanimlanan kaynaklari barindirir.

Root/Parent modulu, child(alt) modullerdeki kaynaklari kullanarak konfigürasyonu tamamlar.

Published (yayinlanmis) Modules

In addition to modules from the local filesystem, Terraform can load modules from a public or private registry. This makes it possible to publish modules for others to use and to use modules that others have published. Yerel dosya sistemindeki modüllere ek olarak, Terraform, halka açık veya özel bir repodan/registry'den modüller yükleyebilir. Bu, başkalarının kullanması için modüller yayınlamayı ve başkalarının yayınladığı modülleri kullanmayı mümkün kılar.

The **Terraform Registry** hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure. These modules are free to use, and Terraform can download them automatically if you specify the appropriate source and version in a module call block. Terraform Registry, birçok türde ortak altyapıyı yapılandırmak için halka açık geniş bir Terraform modülü koleksiyonuna ev sahipliği yapar. Bu modüllerin kullanımı ücretsizdir ve bir modül çağrı bloğunda uygun kaynağı ve sürümü belirtirseniz Terraform bunları otomatik olarak indirebilir.

What are modules for?

Here are some of the ways that modules help solve the problems listed above: Modüllerin yukarıda listelenen sorunları çözmeye yardımcı olma yollarından bazıları şunlardır:

Organize configuration - Modules make it easier to navigate, understand, and update your configuration by keeping related parts of your configuration together. Even moderately complex infrastructure can require hundreds or thousands of lines of configuration to implement. By using modules, you can organize your configuration into logical components. Modüller, yapılandırmanızın ilgili kısımlarını bir arada tutarak yapılandırmanızda gezinmeyi, anlamayı ve yapılandırmayı güncellemeyi kolaylaştırır. Orta derecede karmaşık altyapı bile, uygulanması için yüzlerce veya binlerce satır yapılandırma gerektirebilir. Modülleri kullanarak yapılandırmanızı mantıksal bileşenler halinde düzenleyebilirsiniz. Yani, birbirine uygun bilesenlerin gruplandirilarak organize edilmesidir.

Encapsulate configuration-(kapsulleme yapilandirmasi) Another benefit of using modules is to encapsulate configuration into distinct logical components. Encapsulation can help prevent unintended consequences, such as a change to one part of your configuration accidentally causing changes to other infrastructure, and reduce the chances of simple errors like using the same name for two different resources. Modülleri kullanmanın bir başka faydası da konfigürasyonu farklı mantıksal bileşenlere yerleştirmektir. Kapsülleme, yapılandırmanızın bir bölümünde yanlışlıkla başka altyapıda değişikliklere neden olması gibi istenmeyen sonuçların önlenmesine yardımcı olabilir ve aynı adı iki farklı kaynak için kullanmak gibi basit hata olasılığını azaltır.

Re-use configuration -(Yapılandırmayı yeniden kullanın) Writing all of your configuration from scratch can be time-consuming and error-prone. Using modules can save time and reduce costly errors by re-using configuration written either by yourself, other members of your team, or other Terraform practitioners who have published modules for you to use. You can also share modules that you have written with your team or the general public, giving them the benefit of your hard work.Tüm yapılandırmanızı sıfırdan yazmak zaman alabilir ve hataya açık olabilir. Modülleri kullanmak, kendiniz, ekibinizin diğer üyeleri veya kullanmanız için modüller yayınlamış diğer Terraform uygulayıcıları tarafından yazılan yapılandırmayı yeniden kullanarak zamandan tasarruf edebilir ve maliyetli hataları azaltabilirsiniz. Ayrıca yazdığınız modülleri ekibinizle veya genel halkla paylaşabilir ve onlara sıkı çalışmanızın faydasını sağlayabilirsiniz.

Provide consistency and ensure best practices -Tutarlılık(standart) ve en iyi uygulamaları sağlama - Modules also help to provide consistency in your configurations. Not only does consistency make complex configurations easier to understand, it also helps to ensure that best practices are applied across all of your configurations. For instance, cloud providers give many options for configuring object storage services, such as Amazon S3 or Google Cloud Storage buckets. There have been many high-profile security incidents involving incorrectly secured object storage, and given the number of complex configuration options involved, it's easy to accidentally misconfigure these services.Modüller, yapılandırmalarınızda tutarlılık(standart) sağlamaya da yardımcı olur. Tutarlılık, yalnızca karmaşık yapılandırmaların anlaşılmasını kolaylaştırmakla kalmaz, aynı zamanda tüm yapılandırmalarınızda en iyi uygulamaların kullanilmasini sağlamaya da yardımcı olur. Örneğin, cloud providers (bulut saglayicilari) Amazon S3 veya Google Cloud Storage gibi depolama hizmetlerini yapılandırmak için birçok seçenek sunar. Birçok yüksek profilli güvenlik hatalari/kazalari yanlışlıkla nesne depolamalar ve karmaşık yapılandırma seçeneklerinin sayısı göz önüne alındığında, bu hizmetleri istemen veya farkinda olmadan yanlış yapılandırmak kolaydır.

Using modules can help reduce these errors. For example, you might create a module to describe how all of your organization's public website buckets will be configured, and another module for private buckets used for logging applications. Also, if a configuration for a type of resource needs to be updated, using modules allows you to make that update in a single place and have it be applied to all cases where you use that module. Modüllerin kullanılması bu hataların azaltılmasına yardımcı olabilir. Örneğin, kuruluşunuzun tüm genel web sitesi paketlerinin nasıl yapılandırılacağını açıklamak için bir modül ve uygulamaları günlüğe kaydetmek için kullanılan özel paketler için başka bir modül oluşturabilirsiniz. Ayrıca, bir kaynak türü için bir yapılandırmanın güncellenmesi gerekiyorsa, modülleri kullanmak, bu güncellemeyi tek bir yerde yapmanıza ve bu modülü kullandığınız tüm durumlara uygulanmasını sağlar.

What is a Terraform module?

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. You may have a simple set of Terraform configuration files such as: Bir Terraform modülü, tek bir dizindeki bir Terraform yapılandırma dosyasıdır. Bir veya daha fazla .tf dosyası içeren tek bir dizinden oluşan basit bir yapılandırma bile bir modüldür. Terraform komutlarını doğrudan böyle bir dizinden çalıştırdığınızda, kök modül olarak kabul edilir. Bu anlamda, her Terraform konfigürasyonu bir modülün parçasıdır. Aşağıdakiler gibi basit bir Terraform yapılandırma dosyanız olabilir:

```
.
├── LICENSE
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
```

In this case, when you run terraform commands from within the minimal-module directory, the contents of that directory are considered the root module. Bu durumda, terraform komutlarını minimal modül dizini içinden çalıştırdığınızda, bu dizinin içeriği kök modül olarak kabul edilir.

Which of the following is not an advantage of the terraform modules?

Select one:

A. Re-use configuration
B. Store your data in remote state
C. Provide consistency and ensure best practices

Build a Module

Terraform treats every configuration as a module. When you run terraform commands, the target directory containing Terraform configuration is treated as the root module. Terraform, her konfigürasyonu bir modül olarak ele alır. Terraform komutlarını çalıştırdığınızda, Terraform yapılandırmasını içeren hedef dizin kök modül olarak kabul edilir.

We will create a module to create an IAM User for different departments using Terraform module. Terraform modülünü kullanan farklı departmanlar için bir IAM Kullanıcısı oluşturacak bir modül olusturacagiz.

Create a directory.

```
$ mkdir modules
```

Change into that directory in your terminal. Next, create a terraform config file name `main.tf` Terminalinizdeki bu dizine geçin. Ardından, main.tf adında bir terraform yapılandırma dosyası oluşturun

```
$ cd modules
$ vim main.tf

#main.tf
provider "aws" {
 region = "us-east-1"
}
resource "aws_iam_user" "my-new-user" {
 name = "oliver-terraform-${var.environment}"
}
```

Create a variable file. Değişken bir dosya oluşturun.

```
$ vim variables.tf
----------
# variables.tf
variable "environment" {
 default = "default"
}
```

Create an output file. Bir çıktı dosyası oluşturun.

```
$ vim outputs.tf
----------
# outputs.tf
output "my-terraform-user" {
 value = aws_iam_user.my-new-user.name
}
```

Next, go to the previous folder and create a config file for the developer's department. Ardından, önceki klasöre gidin ve developer departmanı için bir yapılandırma dosyası oluşturun.

```
$ vim main.tf
```

Now create a config file and create an IAM user for the developers Şimdi developers için bir yapılandırma dosyası oluşturun ve bir IAM kullanıcısı oluşturun

```
# main.tf
module "usermodule" {
 source = "./modules"
 environment = "DEV"
}
```

Ensure that Terraform has downloaded all the necessary providers and modules by initializing it. Terraform'u başlatarak gerekli tüm sağlayıcıları ve modülleri indirdiğinden emin olun.

```
$ terraform init

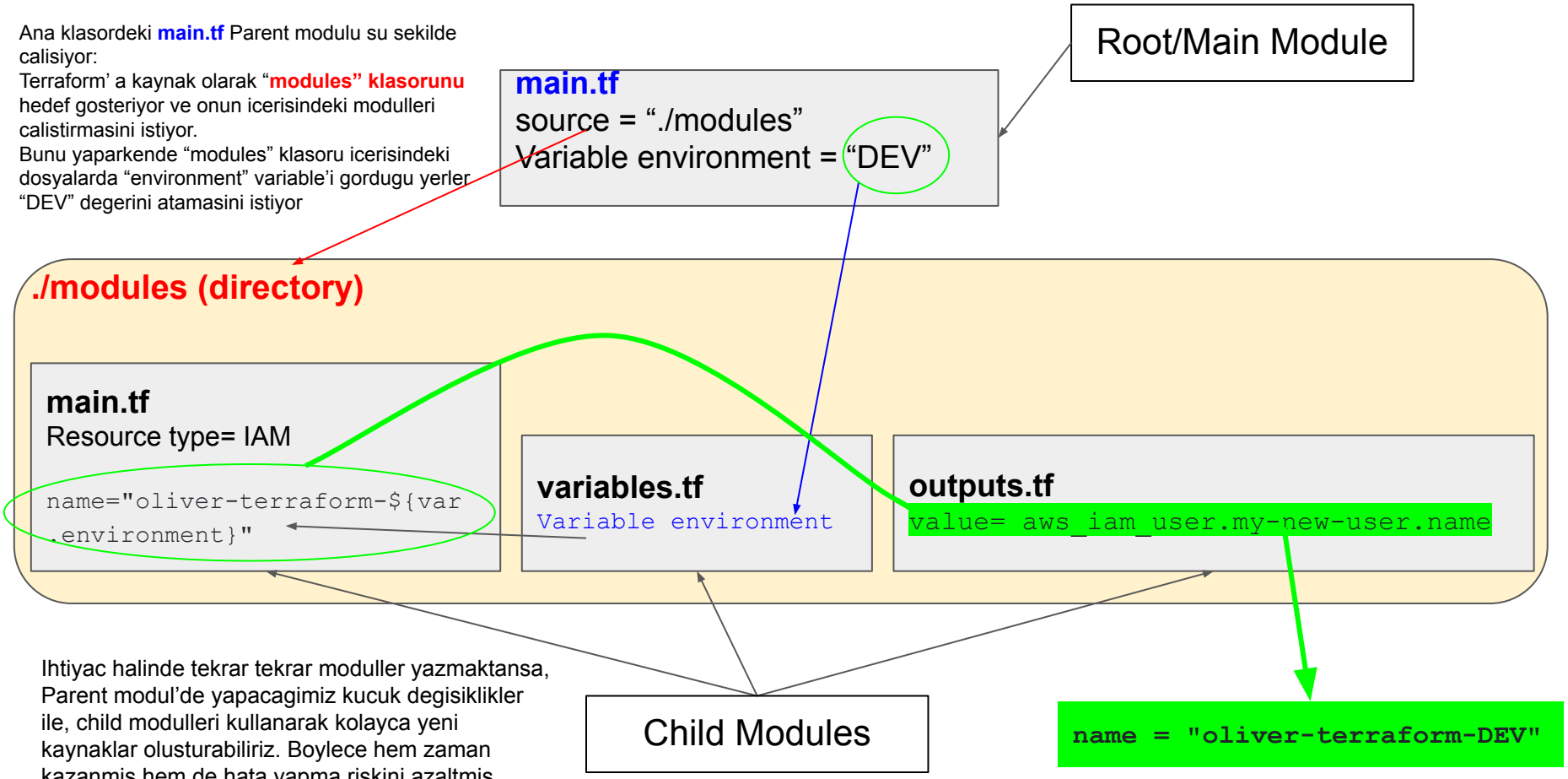
Initializing modules...
- usermodule in modules
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.27.0...
- Installed hashicorp/aws v3.27.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control
    repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
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.
```

Now that your new module is installed and configured, run `terraform apply` to provision your IAM User. Artık yeni modülünüz yüklenip yapılandırıldığına göre, IAM Kullanıcınızı sağlamak için terraform apply'ı çalıştırın.

```
$ terraform apply

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:
 # module.usermodule.aws_iam_user.my-new-user will be created
 + resource "aws_iam_user" "my-new-user" {
    + arn           = (known after apply)
    + force_destroy = false
    + id            = (known after apply)
    + name          = "oliver-terraform-DEV"
    + path          = "/"
    + unique_id     = (known after apply)
   }
Plan: 1 to add, 0 to change, 0 to destroy.
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
module.usermodule.aws_iam_user.my-new-user: Creating...
module.usermodule.aws_iam_user.my-new-user: Creation complete after 2s [id
    =oliver-terraform-DEV]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

After running `terraform apply`, your new user will be created. Terraform Apply çalıştırıldıktan sonra yeni kullanıcınız oluşturulacaktır.


Ana klasordeki main.tf Parent modulu su sekilde calisiyor:
Terraform' a kaynak olarak "modules" klasorunu hedef gosteriyor ve onun icerisindeki modulleri calistirmasini istiyor.
Bunu yaparkende "modules" klasoru icerisindeki dosyalarda "environment" variable'i gordugu yerler "DEV" degerini atamasini istiyor
Root/Main Module
main.tf
source = "./modules"
Variable environment = "DEV"
./modules (directory)
main.tf
Resource type= IAM
name="oliver-terraform-${var.environment}"
variables.tf
Variable environment
outputs.tf
value= aws_iam_user.my-new-user.name
Ihtiyac halinde tekrar tekrar moduller yazmaktansa, Parent modul'de yapacagimiz kucuk degisiklikler ile, child modulleri kullanarak kolayca yeni kaynaklar olusturabiliriz. Boylece hem zaman kazanmis hem de hata yapma riskini azaltmis oluruz.
Child Modules
name = "oliver-terraform-DEV"

⚠️**Note:** After Terraform 0.12, only the root module outputs appear in state snapshots. Child module outputs now exist only temporarily in memory, due to them now being implemented consistently with input variables and local values. If you want to display output when you use a module, you can use an output block with the module block. Terraform 0.12'den sonra, durum anlık görüntülerinde yalnızca kök modül çıktıları görünür. Alt modül çıkışları, artık giriş değişkenleri ve yerel değerlerle tutarlı bir şekilde uygulandıkları için yalnızca geçici olarak bellekte bulunmaktadır. Bir modül kullandığınızda çıktıyı görüntülemek istiyorsanız, modül bloğu ile bir çıktı bloğu kullanabilirsiniz.

- ```
  output "example" {
  value = module.usermodule.my-terraform-user
  }
  ```

⚠️**Note:** Do not forget to destroy the resources you created with the command terraform destroy terraform destroy komutu ile oluşturduğunuz kaynakları yok etmeyi unutmayınız.
  ```

Child modules can be called only one time within the same configuration.

Select one:

- True
- False

Where can you find a broad collection of publicly available Terraform modules ?

Select one:

A. Terraform Workspace
B. Amazon Terraform Registry
C. Terraform Cloud
D. Terraform Registry

What is the correct approach to call a child module?

```
module "webservers" {
    ____________________
}
```

"source" = "/dev-cluster" source = "/dev-cluster" source : "/dev-cluster" resource = "/dev-cluster"

Terraform Remote State

Remote State

Terraformu calistirdigimizda remote state'den son durumu alip, calismamiz bittiginde state dosyasini guncelleyip Remote State koyuyor. Bu islemi yaparken sadece bir kullaniciya izin vererek aynı anda başka kimsenin Terraform'u çalıştırmadığından emin olmamizi sagliyor.

By default, Terraform stores state locally in a file named terraform.tfstate. When working with Terraform in a team, the use of a local file makes Terraform usage complicated because each user must make sure they always have the latest state data before running Terraform and make sure that nobody else runs Terraform at the same time. Varsayılan olarak, Terraform state'i yerel olarak terraform.tfstate adlı bir dosyada saklar. Bir ekipte Terraform ile çalışırken, yerel bir dosyanın kullanılması Terraform kullanımını karmaşık hale getirir çünkü her kullanıcı Terraform'u çalıştırmadan önce her zaman en son state (durum) verilerine sahip olduğundan ve Terraform'u aynı anda başka kimsenin çalıştırmadığından emin olmalıdır.

With remote state, Terraform writes the state data to a remote data store, which can then be shared between all members of a team. Terraform supports storing state in Terraform Cloud, HashiCorp Consul, **Amazon S3**, Azure Blob Storage, Google Cloud Storage, Alibaba Cloud OSS, and more. Remote state ile Terraform, durum verilerini uzak bir veri deposuna yazar ve bu daha sonra bir ekibin tüm üyeleri arasında paylaşılabilir. Terraform, Terraform Cloud, HashiCorp Consul, Amazon S3, Azure Blob Storage, Google Cloud Storage, Alibaba Cloud OSS ve daha fazlasında depolama durumunu destekler.

Remote state is implemented by a **backend**, which you can configure in your configuration's root module. Uzak durum, yapılandırmanızın kök modülünde yapılandırabileceğiniz bir backend tarafından uygulanır.

Remote state allows you to share output values with other configurations. This allows your infrastructure to be decomposed into smaller components. Remote state, çıktı değerlerini diğer konfigürasyonlarla paylaşmanıza olanak tanır. Bu, altyapınızın daha küçük bileşenlere ayrılmasını sağlar.

Put another way, remote state also allows teams to share infrastructure resources in a read-only way without relying on any additional configuration store. Başka bir deyişle, Remote state, ekiplerin altyapı kaynaklarını herhangi bir ek yapılandırma deposuna dayanmadan salt okunur bir şekilde paylaşmasına da olanak tanır.

For fully-featured remote backends, Terraform can also use state locking to prevent concurrent runs of Terraform against the same state. Tam özellikli uzak arka uçlar için Terraform, Terraform'un aynı duruma karşı eşzamanlı çalışmasını önlemek için durum kilitlemeyi de kullanabilir.

Where does Terraform stores state by default?

Select one:

A. Amazon S3
B. Backend
C. Local

Backends

Each Terraform configuration can specify a backend, which defines where and how operations are performed, where state snapshots are stored, etc. Her Terraform konfigürasyonu, işlemlerin nerede ve nasıl gerçekleştirildiğini, durum anlık görüntülerinin (state snapshots) nerede saklandığını vb. tanımlayan bir arka uç belirtebilir.

Backends are responsible for storing state and providing an API for state locking. State locking is optional. Backends are responsible for supporting state locking if possible. Not all backends support locking. If supported by your backend, Terraform will lock your state for all operations that could write state. This prevents others from acquiring the lock and potentially corrupting your state. Backends, durumu depolamaktan ve durum kilitleme (State locking) için bir API sağlamaktan sorumludur. Durum kilitleme (State locking) isteğe bağlıdır. Arka uçlar, mümkünse State locking'i desteklemekten sorumludur. Tüm Backends'ler locking'i desteklemez. Backends tarafından destekleniyorsa, Terraform, durum yazabilecek tüm işlemler için durumunuzu kilitleyecektir. Bu, başkalarının kilidi almasını ve potansiyel olarak durumunuzu bozmasını önler.

Despite the state being stored remotely, all Terraform commands such as `terraform console`, the `terraform state` operations, `terraform taint`, and more will continue to work as if the state was local. Durumun uzaktan depolanmasına rağmen, `terraform console`, `terraform state` işlemleri, `terraform taint`, ve daha fazlası gibi tüm Terraform komutları, state yerelmiş gibi çalışmaya devam edecektir. `terraform taint`(Belli bir nesnenin bozuldugunu veya hasar gordugunu bildirir ve bunu kusurlu olarak isaretler. Bir sonraki plan'daonu degistirmeyi onerir)

Backends determine where state is stored. For example, the local (default) backend stores state in a local JSON file on disk. Backends, state'in nerede depolanacağını belirler. Örneğin, local (varsayılan/default) backend, durumu localdeki yerel bir JSON dosyasında saklar.

Amazon S3

S3 Stores the state as a given key in a given bucket on Amazon S3. This backend also supports state locking and consistency checking via Dynamo DB, which can be enabled by setting the `dynamodb_table` field to an existing DynamoDB table name. A single DynamoDB table can be used to lock multiple remote state files. Terraform generates key names that include the values of the `bucket` and `key` variables. S3 State'i, Amazon S3'teki belirli bir klasörde belirli bir anahtar/key olarak depolar. Bu arka uç/backend, `dynamodb_table` alanı mevcut bir DynamoDB tablo adına ayarlanarak etkinleştirilebilen Dynamo DB aracılığıyla state locking/durum kilitleme ve consistency checking/tutarlılık kontrolünü de destekler. Birden çok uzak durum dosyasını/multiple remote state files kilitlemek için tek bir DynamoDB tablosu kullanılabilir. Terraform, `bucket` ve `key` değişkenlerin değerlerini içeren anahtar adları oluşturur.

You can enable **Bucket Versioning** on the S3 bucket to allow for state recovery in the case of accidental deletions and human error. insan hatası ve yanlışlıkla silme durumunda, durum kurtarmaya izin vermek için S3 bucket Paket Sürüm Oluşturma'yı/Bucket Versioning'i etkinleştirebilirsiniz.

Terraform state can contain sensitive data, depending on the resources in use and your definition of "sensitive." The state contains resource IDs and all resource attributes. For resources such as databases, this may contain initial passwords. When using remote backend Amazon S3, you can also enable encryption on the S3 bucket to encrypt to data. Terraform state, kullanılan kaynaklara ve 'hassas' tanımınıza bağlı olarak hassas veriler içerebilir. State, kaynak kimliklerini ve tüm kaynak özniteliklerini içerir. Veritabanları gibi kaynaklar için bu, başlangıç parolalarını içerebilir. Remote backend Amazon S3'ü kullanırken, verileri şifrelemek için S3 klasöründe şifrelemeyi de etkinleştirebilirsiniz.

Configuration of S3 Backend

First, we assume that we have a bucket created called `mybucket`. İlk olarak, mybucket adında bir bucket oluşturduğumuzu varsayalım.

Next, add the following terraform block to your config file. Ardından, yapılandırma dosyanıza aşağıdaki terraform bloğunu ekleyin.

```
terraform {
 backend "s3" {
   bucket = "mybucket"
   key    = "path/to/my/key"
   region = "us-east-1"
 }
}
```

The Terraform state is written to the key `path/to/my/key`. Terraform durumu,/to/my/key anahtar yoluna yazılır.

AWS S3 Bucket module - Ornek

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.8.0"
    }
  }
}

provider "aws" {
  region  = "us-east-1"
}

resource "aws_s3_bucket" "tf-s3" {
  bucket = "bucket-name" # bu kismi degistirecegiz
}
```

What functionality is introduced to ensure that the terraform state file does not get corrupted when multiple users are trying to make use of the same file simultaneously?

Select one:

A. Remote State
B. Backend
C. Amazon S3
D. state locking

Which configuration block type is used to add backend to Terraform?

Select one:

A. provider
B. backend
C. terraform
D. Resource

Which one is not a Terraform remote state?

Select one:

A. Terraform Workspace
B. HashiCorp Consul
C. Amazon S3
D. Terraform Cloud