**Terraform Import**

**Import**

Terraform is able to import existing infrastructure. This allows you to take resources you've created by some other means and bring it under Terraform management.

This is a great way to slowly transition infrastructure to Terraform or to be able to be confident that you can use Terraform in the future if it potentially doesn't support every feature you need today.

The current implementation of Terraform import can only import resources into the state. It does not generate a configuration.

Because of this, prior to running terraform import it is necessary to write manually a resource configuration block for the resource, to which the imported object will be mapped.

**⚠ Note:**

If you import existing objects into Terraform, be careful to import each remote object to only one Terraform resource address. If you import the same object multiple times, Terraform may exhibit unwanted behavior.

**Usage**

Terraform import [options] ADDRESS ID

Import will find the existing resource from ID and import it into your Terraform state at the given ADDRESS.

ADDRESS must be a valid resource address. Because any resource address is valid, the import command can import resources into modules as well as directly into the root of your state.

ID is dependent on the resource type being imported. For example, for AWS instances it is the instance ID (i-abcd1234) but for AWS Route53 zones it is the zone ID (Z12ABC4UGMOZ2N). Please reference the provider documentation for details on the ID format. If you're unsure, feel free to just try an ID. If the ID is invalid, you'll just receive an error message.

**Import an Existing Resource**

However, you may need to manage an infrastructure that wasn’t created by Terraform. Terraform import solves this problem by loading supported resources into your Terraform workspace’s state. The import command doesn’t automatically generate the configuration to manage the infrastructure, though. Because of this, importing existing infrastructure into Terraform is a multi-step process.

Bringing existing infrastructure under Terraform’s control involves five main steps:

1. Identify the existing infrastructure to be imported.
2. Import infrastructure into your Terraform state.
3. Write Terraform configuration that matches that infrastructure.
4. Review the Terraform plan to ensure the configuration matches the expected state and infrastructure.
5. Apply the configuration to update your Terraform state.

**Create an AWS EC2 instance**

Go to your AWS account and create an EC2 instance (Amazon Linux 2 AMI, t2.micro, your pem file, tag (Name = learn-tf-import)). You can use default values when creating.

**Import the EC2 instance into Terraform**

Create a directory and change into that directory. Then write Terraform configuration that matches that EC2 instance. Write the arguments according to the EC2 instance that you created before. They must have the same values with the instance.

$ mkdir import-tf && cd import-tf

$ vim import-ec2.tf

# import-ec2.tf

provider "aws" {

region = "us-east-1"

}

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

ami = "ami-047a51fa27710816e"

instance\_type = "t2.micro"

key\_name = "northvirginia"

// Write your pem file name

vpc\_security\_group\_ids = ["vpc-64ffc61e"]

subnet\_id = "subnet-c89a71e9"

tags = {

Name = "learn-tf-import"

}

Next, initialize terraform in the current directory.

$ terraform init

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.

Next, run the command below to import the instance you created before. Take the instance id from your AWS console.

$ terraform import aws\_instance.example-ec2 i-0dfa805978aa33e2f

aws\_instance.example-ec2: Importing from ID "i-0dfa805978aa33e2f"...

aws\_instance.example-ec2: Import prepared!

Prepared aws\_instance for import

aws\_instance.example-ec2: Refreshing state... [id=i-0dfa805978aa33e2f]

Import successful!

The resources that were imported are shown above. These resources are now in

your Terraform state and will henceforth be managed by Terraform.

Now you will see that the terraform.tfstate was created in your current directory. So succeed to import an existing instance to your terraform state and you can manage the EC2 instance with terraform anymore.