Import existing resources into Terraform and generate code with examples

Leslie Alldridge · Follow

5 min read · Jul 8, 2023

> *These examples assume you're familiar with initialising a basic Terraform project and running commands such as* `terraform plan` *and* `terraform apply`

Table of Contents:

Why do we import resources?

If you're new to Terraform or learning a cloud provider (for example AWS) sometimes it's easier to use the web console when creating your infrastructure. Once you've created your desired resources they can be imported into Terraform which allows you to leverage the benefits of Infrastructure as Code (IaC).

Even skilled Engineers may find a new role or cloud provider daunting. They may click around in the UI and reach a point where something is working and then import their resources as Terraform to manage it going forward. This can also be a faster feedback loop compared to Terraform plan + apply commands being run over multiple iterations.

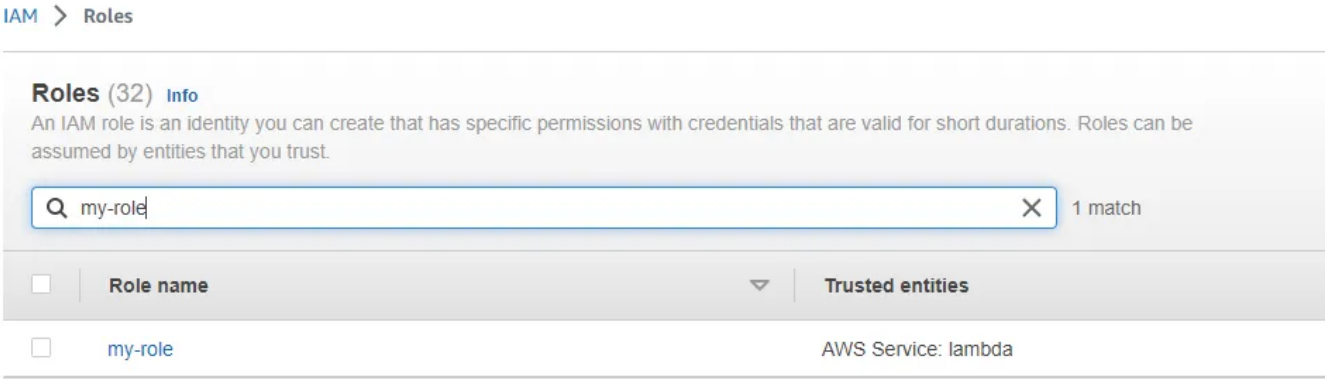
Another reason to import will be if you're moving from another IaC provider to Terraform — you wouldn't want to take production offline during this process so importing is the way to go. Some companies won't have any IaC and you may be tasked with setting this up — again, importing is the way to go.

Import resources on Terraform v1.5.0 and newer

The release of v1.5.0 bought with it some exciting changes to importing in Terraform. I recommend upgrading to this version or newer as it requires less effort to import resources.

In this example I'll be importing an AWS IAM role and policy. However, it's the same process for any resource.

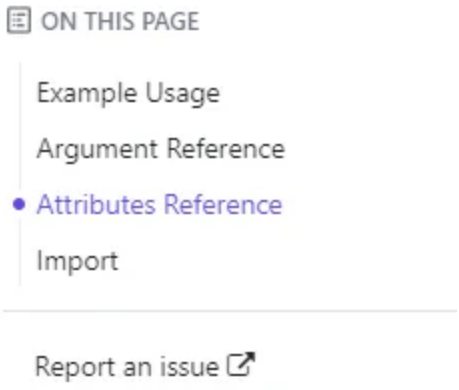
Identify resources to import



The role I'm going to import

Add an import resource block to your Terraform configuration

Locate the relevant Terraform documentation page, for me that's aws_iam_role. Scroll to the bottom of the page to see import details. To get there without scrolling you can click the menu (top right):



Documentation menu — click Import

We import this resource using the name (my-role)

Therefore, my Terraform code will be:

```
import {
  to = aws_iam_role.my_role
  id = "my-role"
}
```

`to` refers to the Terraform resource you want to import the configuration into and `id` is the attribute we need to import by (in this case it's the IAM role name). `to` can include indexes from `for_each` and `count` such as `[0]`

Assuming you have a Terraform project already initialised with `terraform init` we can go ahead and generate the code for the IAM role resource automatically. In the terminal you can run `terraform plan -generate-config-out='generated.tf'` (*Note: the single quotes are optional but without them it won't work on Windows using Powershell*).

Now I have a new file named `generated.tf` on my machine with the following contents:

```
# __generated__ by Terraform
# Please review these resources and move them into your main configuration files

# __generated__ by Terraform from "my-role"
resource "aws_iam_role" "my_role" {
  assume_role_policy    = "{\"Statement\":[{\"Action\":\"sts:AssumeRole\",\"Effe
  description           = "Allows Lambda functions to call AWS services on your
  force_detach_policies = false
  managed_policy_arns   = ["arn:aws:iam::xxx:policy/service-role/AWSLambdaBasicE
  max_session_duration  = 3600
  name                  = "my-role"
  name_prefix           = null
  path                  = "/"
  permissions_boundary  = null
  tags                  = {}
  tags_all              = {}
}
```

Some of the fields can be removed such as `tags` and `name_prefix` but you can follow the official documentation to discover which arguments are mandatory for your specific resource(s).

Terraform code can be split across many files so you can run a `terraform plan` followed by `terraform apply` and your imported resource will be persisted to state.

Depending on your workflow you can create a pull request and the Terraform changes will show your reviewer resources that will be imported. This is fantastic for environments where you don't want to / don't have the ability to directly manipulate Terraform state from your local machine and/or need a second opinion.

```
aws_iam_role.my_role: Importing... [id=my-role]
aws_iam_role.my_role: Import complete [id=my-role]
```

Now I can run a `terraform state list` and it shows me:

```
aws_iam_role.my_role
```

That's it! The IAM role is now managed in Terraform — The massive win being, I didn't write any of it.

Link to official documentation: Terraform Import

Import resources on Terraform v1.4.x and older

In older versions of Terraform users had to import resources with the command line. The import happened immediately so it was difficult to raise change requests or have a reviewer double checking unless you were pair programming together. It was easy to make mistakes and quite labour intensive.

The first step here is to write the resource in Terraform using your best guess and following the documentation for guidance.

I've come up with the following:

```
resource "aws_iam_role" "my_role" {
  assume_role_policy   = "{\"Statement\":[{\"Action\":\"sts:AssumeRole\",\"Effec
  managed_policy_arns  = ["arn:aws:iam::xxx:policy/service-role/AWSLambdaBasicEx
  max_session_duration = 3600
  name                 = "my-role"
}
```

Next, use the Terraform command `import` to import your resource (where `my-role` is my IAM role name):

```
terraform import aws_iam_role.my_role my-role
```

```
aws_iam_role.my_role: Importing from ID "my-role"...
aws_iam_role.my_role: Import prepared!
  Prepared aws_iam_role for import
aws_iam_role.my_role: Refreshing state... [id=my-role]

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.
```

The import is complete, now we run a `terraform plan` to see how close our code from earlier was...

```
aws_iam_role.my_role: Refreshing state... [id=my-role]

Terraform used the selected providers to generate the following execution plan.
  ~ update in-place
```

```
Terraform will perform the following actions:

  # aws_iam_role.my_role will be updated in-place
  ~ resource "aws_iam_role" "my_role" {
      - description           = "Allows Lambda functions to call AWS services on
        id                    = "my-role"
        name                  = "my-role"
        tags                  = {}
        # (9 unchanged attributes hidden)
    }

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

As you can see I left out the description field. I can either apply this plan and empty the description field, or go back and write it into my Terraform code.

Description was left out on purpose to show you the process can be more challenging compared to importing in Terraform v1.5.0 and newer.

I hope you've enjoyed learning more about importing resources in Terraform. Please let me know in the comments if this post has been helpful to you!