Create Resource Dependencies

developer.hashicorp.com/terraform/tutorials/certification-associate-tutorials/dependencies

In this tutorial, you will learn about dependencies between resources and modules. Most of the time, Terraform infers dependencies between resources based on the configuration given, so that resources are created and destroyed in the correct order. Occasionally, however, Terraform cannot infer dependencies between different parts of your infrastructure, and you will need to create an explicit dependency with the `depends_on` argument.

Prerequisites

You can complete this tutorial using the same workflow with either Terraform OSS or Terraform Cloud. Terraform Cloud is a platform that you can use to manage and execute your Terraform projects. It includes features like remote state and execution, structured plan output, workspace resource summaries, and more.

Select the **Terraform OSS** tab to complete this tutorial using Terraform OSS.

This tutorial assumes that you are familiar with the Terraform and Terraform Cloud workflows. If you are new to Terraform, complete the Get Started tutorials first. If you are new to Terraform Cloud, complete the Terraform Cl oud Get Started tutorials first.

For this tutorial, you will need:

- Terraform v1.2+ installed locally.
- a Terraform Cloud account and organization.
- Terraform Cloud locally authenticated.
- the AWS CLI.
- a Terraform Cloud variable set configured with your AWS credentials.

This tutorial assumes that you are familiar with the Terraform workflow. If you are new to Terraform, complete the [Get Started tutorials(/collections/terraform/aws-get-started) first.

For this tutorial, you will need:

- Terraform v1.2+ installed locally.
- AWS Credentials configured for use with Terraform.

Clone example repository

Clone the example repository for this tutorial.

```
$ git clone https://github.com/hashicorp/learn-terraform-dependencies.git
```

Initialize workspace

Set the `TF_CLOUD_ORGANIZATION` environment variable to your Terraform Cloud organization name. This will configure your Terraform Cloud integration.

```
$ export TF_CLOUD_ORGANIZATION=
```

Initialize your configuration. Terraform will automatically create the `learn-terraform-dependencies` workspace in your Terraform Cloud organization.

```
$ terraform init
Initializing Terraform Cloud...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v4.17.1...
- Installed hashicorp/aws v4.17.1 (signed by HashiCorp)
Terraform Cloud has been successfully initialized!
You may now begin working with Terraform Cloud. Try running "terraform plan" to
see any changes that are required for your infrastructure.
If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.
```

Note: This tutorial assumes that you are using a tutorial-specific Terraform Cloud organization with a global variable set of your AWS credentials. Review the Create a Credential Variable Set for detailed guidance. If you are using a scoped variable set, assign it to your new workspace now.

Open your `terraform.tf` file and comment out the `cloud` block that configures the Terraform Cloud integration.

terraform.tf

```
terraform {
  /*
  cloud {
    workspaces {
      name = "learn-terraform-dependencies"
    }
  }
  */

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.17.1"
    }
  }

  required_version = ">= 1.2"
}
```

Initialize this configuration.

```
$ terraform init
Initializing the backend...
##...
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.
```

Manage implicit dependencies

The most common source of dependencies is an implicit dependency between two resources or modules.

Review the configuration in `main.tf`. It declares two EC2 instances and an Elastic IP address.


main.tf

```
provider "aws" {
  region = var.aws_region
}

data "aws_ami" "amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}

resource "aws_instance" "example_a" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"
}

resource "aws_instance" "example_b" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"
}

resource "aws_eip" "ip" {
  vpc      = true
  instance = aws_instance.example_a.id
}
```

The `aws_eip` resource type allocates and associates an Elastic IP to an EC2 instance. Since the instance must exist before the Elastic IP can be created and attached, Terraform must ensure that `aws_instance.example_a` is created before it creates `aws_eip.ip`. Meanwhile, `aws_instance.example_b` can be created in parallel to the other resources.

Apply the configuration. Respond to the confirmation prompt with a `yes`.

```
$ terraform apply
##...
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_eip.ip will be created
  + resource "aws_eip" "ip" {

##...

Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions in workspace "learn-terraform-dependencies"?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Terraform provisions your resources in order, and reports on its progress as it deploys your resources. The output will be similar to the following.

```
aws_instance.example_a: Creating...
aws_instance.example_b: Creating...
aws_instance.example_b: Still creating... [10s elapsed]
aws_instance.example_a: Still creating... [10s elapsed]
aws_instance.example_a: Still creating... [20s elapsed]
aws_instance.example_b: Still creating... [20s elapsed]
aws_instance.example_b: Still creating... [30s elapsed]
aws_instance.example_a: Still creating... [30s elapsed]
aws_instance.example_b: Creation complete after 32s [id=i-0d485fc8d11d27c87]
aws_instance.example_a: Creation complete after 33s [id=i-07c07aebe269be8a5]
aws_eip.ip: Creating...
aws_eip.ip: Creation complete after 1s [id=eipalloc-0ef86dc6fbb50ccf8]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

As shown above, Terraform waited until the creation of EC2 instance `example_a` was complete before creating the Elastic IP address.

Terraform automatically infers when one resource depends on another by studying the resource attributes used in interpolation expressions. In the example above, the reference to `aws_instance.example_a.id` in the definition of the `aws_eip.ip` block creates an

implicit dependency.

Terraform uses this dependency information to determine the correct order in which to create the different resources. To do so, it creates a dependency graph of all of the resources defined by the configuration. In the example above, Terraform knows that the EC2 Instance must be created before the Elastic IP.

Manage explicit dependencies

Implicit dependencies are the primary way that Terraform understands the relationships between your resources. Sometimes there are dependencies between resources that are *not* visible to Terraform, however. The `depends_on` argument is accepted by any resource or module block and accepts a list of resources to create *explicit dependencies* for.

To illustrate this, assume you have an application running on your EC2 instance that expects to use a specific Amazon S3 bucket. This dependency is configured inside the application, and thus not visible to Terraform. You can use `depends_on` to explicitly declare the dependency. You can also specify multiple resources in the `depends_on` argument, and Terraform will wait until all of them have been created before creating the target resource.

Tip: Since Terraform will wait to create the dependent resource until after the specified resource is created, adding explicit dependencies can increase the length of time it takes for Terraform to create your infrastructure.

Add the following to `main.tf`.


main.tf

```
resource "aws_s3_bucket" "example" { }

resource "aws_instance" "example_c" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"

  depends_on = [aws_s3_bucket.example]
}

module "example_sqs_queue" {
  source  = "terraform-aws-modules/sqs/aws"
  version = "3.3.0"

  depends_on = [aws_s3_bucket.example, aws_instance.example_c]
}
```

Tip: The order in which resources are declared in your configuration files has no effect on the order in which Terraform creates or destroys them.

This configuration includes a reference to a new module, `terraform-aws-modules/sqs/aws`. Modules must be installed before Terraform can use them.

Run `terraform get` to install the module.

```
$ terraform get
Downloading registry.terraform.io/terraform-aws-modules/sqs/aws 3.3.0 for
example_sqs_queue...
- example_sqs_queue in .terraform/modules/example_sqs_queue
```

Now run `terraform apply` to apply the changes. Respond to the confirmation prompt with a `yes`.

```
$ terraform apply
##...

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create
 <= read (data resources)

Terraform will perform the following actions:

  # aws_instance.example_c will be created
  + resource "aws_instance" "example_c" {

##...

Plan: 3 to add, 0 to change, 0 to destroy.


Do you want to perform these actions in workspace "learn-terraform-dependencies"?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Terraform will print output similar to the following.

```
  Enter a value: yes

aws_s3_bucket.example: Creating...
aws_s3_bucket.example: Creation complete after 2s [id=terraform-
20220608191906096600000001]
aws_instance.example_c: Creating...
aws_instance.example_c: Still creating... [10s elapsed]
aws_instance.example_c: Still creating... [20s elapsed]
aws_instance.example_c: Still creating... [30s elapsed]
aws_instance.example_c: Still creating... [40s elapsed]
aws_instance.example_c: Creation complete after 42s [id=i-0fcf06245633ab00e]
module.example_sqs_queue.aws_sqs_queue.this[0]: Creating...
module.example_sqs_queue.aws_sqs_queue.this[0]: Still creating... [10s elapsed]
module.example_sqs_queue.aws_sqs_queue.this[0]: Still creating... [20s elapsed]
module.example_sqs_queue.aws_sqs_queue.this[0]: Creation complete after 26s
[id=https://sqs.us-west-1.amazonaws.com/561656980159/terraform-
20220608191950087600000003]
module.example_sqs_queue.data.aws_arn.this[0]: Reading...
module.example_sqs_queue.data.aws_arn.this[0]: Read complete after 0s
[id=arn:aws:sqs:us-west-1:561656980159:terraform-20220608191950087600000003]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

Since both the instance and the SQS Queue are dependent upon the S3 Bucket, Terraform waits until the bucket is created to begin creating the other two resources.

Destroy resources

Both implicit and explicit dependencies affect the order in which resources are destroyed as well as created.

Clean up the resources you created in this tutorial using Terraform. Respond to the confirmation prompt with a `yes`.

```
$ terraform destroy
##...

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_eip.ip will be destroyed
  - resource "aws_eip" "ip" {

##...

Plan: 0 to add, 0 to change, 6 to destroy.

Changes to Outputs:

Do you really want to destroy all resources in workspace "learn-terraform-
dependencies"?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

Notice that the SQS Queue, Elastic IP address, and the `example_c` EC2 instance are destroyed before the resources they depend on are.

Terraform will print output similar to the following.

```
  Enter a value: yes

aws_eip.ip: Destroying... [id=eipalloc-0ef86dc6fbb50ccf8]
module.example_sqs_queue.aws_sqs_queue.this[0]: Destroying... [id=https://sqs.us-
west-1.amazonaws.com/561656980159/terraform-20220608191950087600000003]
aws_instance.example_b: Destroying... [id=i-0d485fc8d11d27c87]
aws_eip.ip: Destruction complete after 1s
aws_instance.example_a: Destroying... [id=i-07c07aebe269be8a5]
module.example_sqs_queue.aws_sqs_queue.this[0]: Destruction complete after 2s
aws_instance.example_c: Destroying... [id=i-0fcf06245633ab00e]
aws_instance.example_b: Still destroying... [id=i-0d485fc8d11d27c87, 10s elapsed]
aws_instance.example_a: Still destroying... [id=i-07c07aebe269be8a5, 10s elapsed]
aws_instance.example_c: Still destroying... [id=i-0fcf06245633ab00e, 10s elapsed]
aws_instance.example_b: Still destroying... [id=i-0d485fc8d11d27c87, 20s elapsed]
aws_instance.example_a: Still destroying... [id=i-07c07aebe269be8a5, 20s elapsed]
aws_instance.example_c: Still destroying... [id=i-0fcf06245633ab00e, 20s elapsed]
aws_instance.example_b: Still destroying... [id=i-0d485fc8d11d27c87, 30s elapsed]
aws_instance.example_b: Destruction complete after 30s
aws_instance.example_a: Destruction complete after 30s
aws_instance.example_c: Destruction complete after 30s
aws_s3_bucket.example: Destroying... [id=terraform-20220608191906096600000001]
aws_s3_bucket.example: Destruction complete after 1s

Apply complete! Resources: 0 added, 0 changed, 6 destroyed.
```

If you used Terraform Cloud for this tutorial, after destroying your resources, delete the `learn-terraform-dependencies` workspace from your Terraform Cloud organization.

Next steps

Now that you understand how to manage implicit and explicit dependencies, explore the following resources.

- Read the Terraform documentation for the depends_on meta-argument.
- Read how Terraform uses the dependency graph to manage infrastructure.
- Learn how to provision infrastructure created with Terraform.