Infrastructure as a code.
Terraform.
Lection 2.

INFRASTRUCTURE AS CODE. Terraform remote state and backends.

The remote backend stores Terraform state and may be used to run operations in Terraform Cloud.

When using full remote operations, operations like terraform plan or terraform apply can be executed in Terraform Cloud's run environment, with log output streaming to the local terminal. Remote plans and applies use variable values from the associated Terraform Cloud workspace.

Terraform Cloud can also be used with local operations, in which case only state is stored in the Terraform Cloud backend.

https://www.terraform.io/docs/language/settings/backends/remote.html

INFRASTRUCTURE AS CODE. Implicit and implicit dependencies

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:

The Terraform CLI, version 0.13 or later.
AWS Credentials configured for use with Terraform.

INFRASTRUCTURE AS CODE.
Terraform.
Implicit dependencies example

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

Create a directory named ***learn-terraform-dependencies*** and paste this configuration into a file named ***main.tf***

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "2.69.0"
    }
  }
}
provider aws {
  region = "us-west-1"
}
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
}
```

INFRASTRUCTURE AS CODE. Terraform. Implicit dependencies example

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.

First, initialize this directory for use with ***Terraform***.

```
$ terraform init
```

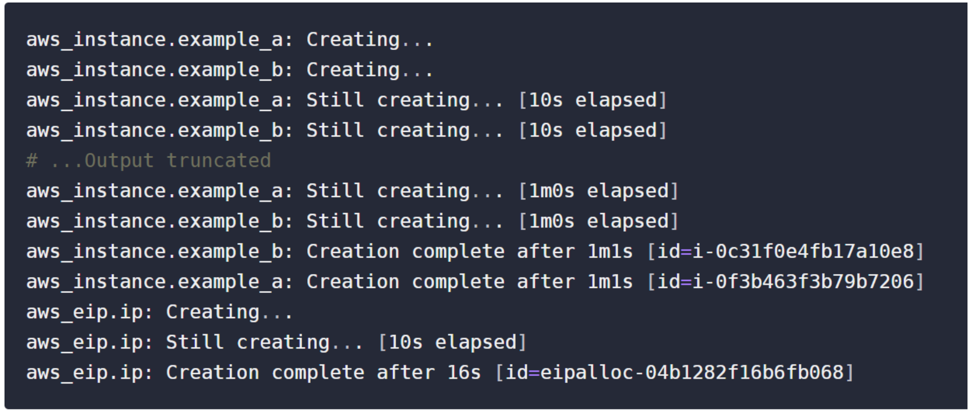
Next, apply the configuration.

```
$ terraform apply
```

Respond to the confirmation prompt with yes.

INFRASTRUCTURE AS CODE. Terraform. Implicit dependencies example

You can see the order Terraform provisions the resources in the output of the apply step. The output will look similar to the following. As shown below, 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 a***ws_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**.



```
aws_instance.example_a: Creating...
aws_instance.example_b: Creating...
aws_instance.example_a: Still creating... [10s elapsed]
aws_instance.example_b: Still creating... [10s elapsed]
# ...Output truncated
aws_instance.example_a: Still creating... [1m0s elapsed]
aws_instance.example_b: Still creating... [1m0s elapsed]
aws_instance.example_b: Creation complete after 1m1s [id=i-0c31f0e4fb17a10e8]
aws_instance.example_a: Creation complete after 1m1s [id=i-0f3b463f3b79b7206]
aws_eip.ip: Creating...
aws_eip.ip: Still creating... [10s elapsed]
aws_eip.ip: Creation complete after 16s [id=eipalloc-04b1282f16b6fb068]
```

INFRASTRUCTURE AS CODE. Terraform. Explicit dependencies example

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.

INFRASTRUCTURE AS CODE. Terraform. Explicit dependencies example

```
resource "aws_s3_bucket" "example" {
  acl    = "private"
}

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 = "2.1.0"

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

INFRASTRUCTURE AS CODE. Terraform. Explicit dependencies example

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 terraform-aws-modules/sqs/aws 2.1.0 for example_sqs_queue...
- example_sqs_queue in .terraform/modules/example_sqs_queue
```

Now run ***terraform apply*** to apply the changes.

```
$ terraform apply
```

INFRASTRUCTURE AS CODE. Terraform. Explicit dependencies example

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.

```
aws_s3_bucket.example: Creating...
aws_s3_bucket.example: Still creating... [10s elapsed]
# ...Output truncated
aws_s3_bucket.example: Creation complete after 1m0s [id=terraform-20200813175124184300000001]
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 44s [id=i-08a44071a2517179f]
module.example_sqs_queue.aws_sqs_queue.this[0]: Creating...
module.example_sqs_queue.aws_sqs_queue.this[0]: Creation complete after 6s [id=https://sqs.us-west-
1.amazonaws.com/561656980159/terraform-20200813175223563000000002]
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-
20200813175223563000000002]

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

INFRASTRUCTURE AS CODE. Terraform. Explicit dependencies example

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.

$ terraform destroy

Respond to the confirmation prompt with a yes.

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

INFRASTRUCTURE AS CODE. Terraform. Variables

Prerequisites

According our goal, you should have a directory named
learn-terraform-aws-instance
with the following configuration in a file called ***main.tf***.
Ensure that your configuration matches this,
and that you have run ***terraform init*** in the
learn-terraform-aws-instance directory.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
}
resource "aws_instance" "example" {
  ami           = "ami-
08d70e59c07c61a3a"
  instance_type = "t2.micro"
  tags = {
    Name = "ExampleInstance"
  }
}
```

INFRASTRUCTURE AS CODE. Terraform. Variables

Set the instance name with a variable

The configuration includes a number of hard-coded values. Terraform variables allow you to write configuration that is flexible and easier to re-use.

Add a variable to define the instance name.

Create a new file called **variables.tf** with a block defining a new **instance_name** variable.

```
variable "instance_name" {
  description = "Value of the Name tag for the EC2 instance"
  type        = string
  default     = "ExampleInstance"
}
```

Note: Terraform loads all files in the current directory ending in **.tf**, so you can name your configuration files however you choose.

INFRASTRUCTURE AS CODE. Terraform. Variables

In **main.tf**, update the **aws_instance** resource block to use the new variable.

```
 resource "aws_instance" "example" {
   ami           = "ami-08d70e59c07c61a3a"
   instance_type = "t2.micro"

   tags = {
-    Name = "ExampleInstance"
+    Name = var.instance_name
   }
 }
```

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

INFRASTRUCTURE AS CODE. Terraform. Variables

In **main.tf**, update the **aws_instance** resource block to use the new variable.

```
 resource "aws_instance" "example" {
   ami           = "ami-08d70e59c07c61a3a"
   instance_type = "t2.micro"

   tags = {
#    Name = "ExampleInstance"
     Name = var.instance_name
   }
 }
```

Apply the configuration. Respond to the confirmation prompt with a yes.
Another one way to set the variable is put it in command line with option **–var:**

```
$ terraform apply -var 'instance_name=YetAnotherName'
```

INFRASTRUCTURE AS CODE. Terraform. Variables

Output EC2 instance configuration

Create a file called **outputs.tf** in your **learn-terraform-aws-instance directory**.

Add outputs to the new file for your EC2 instance's ID and IP address.

```
output "instance_id" {
  description = "ID of the EC2 instance"
  value       = aws_instance.example.id
}

output "instance_public_ip" {
  description = "Public IP address of the EC2 instance"
  value       = aws_instance.example.public_ip
}
```

INFRASTRUCTURE AS CODE. Terraform. Outputs

Terraform prints output values to the screen when you apply your configuration. Query the **outputs** with the ***terraform output*** command.

```
$ terraform output
instance_id = "i-0bf954919ed765de1"
instance_public_ip = "54.186.202.254"
```

You can use Terraform **outputs** to connect your Terraform projects with other parts of your infrastructure, or with other Terraform projects.

INFRASTRUCTURE AS CODE. Terraform. Modules

As you manage your infrastructure with Terraform, you will create increasingly complex configurations. There is no limit to the complexity of a single Terraform configuration file or directory, so it is possible to continue writing and updating your configuration files in a single directory. However, if you do, you may encounter one or more problems:

- Understanding and navigating the configuration files will become increasingly difficult.
- Updating the configuration will become more risky, as an update to one section may cause unintended consequences to other parts of your configuration.
- There will be an increasing amount of duplication of similar blocks of configuration, for instance when configuring separate dev/staging/production environments, which will cause an increasing burden when updating those parts of your configuration.
- You may wish to share parts of your configuration between projects and teams, and will quickly find that cutting and pasting blocks of configuration between projects is error prone and hard to maintain.

So main goal of creating and using Terraform modules is to simplify your current workflow.

INFRASTRUCTURE AS CODE. Terraform. Modules

Create Terraform configuration

For example, you will use **modules** to create an example AWS environment using a Virtual Private Cloud (VPC) and two EC2 instances. You can create it by manually building the directory structure and files using the following commands to clone this GitHub repo.

Clone the GitHub repository.

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

Change into that directory in your terminal.

```
$ cd learn-terraform-modules
```

Check out the ec2-instances tag into a local branch.

```
$ git checkout tags/ec2-instances -b ec2-instances
```

INFRASTRUCTURE AS CODE. Terraform. Modules

```
# Terraform configuration
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
    }
  }
}
provider "aws" {
  region = "us-west-2"
}
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.21.0"
  name = var.vpc_name
  cidr = var.vpc_cidr
  azs             = var.vpc_azs
  private_subnets = var.vpc_private_subnets
  public_subnets  = var.vpc_public_subnets
  enable_nat_gateway = var.vpc_enable_nat_gateway
  tags = var.vpc_tags
}
```

```
#continue of code

module "ec2_instances" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "2.12.0"

  name           = "my-ec2-cluster"
  instance_count = 2

  ami                    = "ami-0c5204531f799e0c6"
  instance_type          = "t2.micro"
  vpc_security_group_ids = [module.vpc.default_security_group_id]
  subnet_id              = module.vpc.public_subnets[0]

  tags = {
    Terraform   = "true"
    Environment = "dev"
  }
}
```

References

https://learn.hashicorp.com/tutorials/terraform/module-use?in=terraform/modules
https://github.com/adv4000/terraform-lessons/tree/master/Lesson-21

Q&A

Thank you!