Provisioning AWS Infrastructure with Terraform

By

Stanley Stephen

Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/
Email: s.stanley.mca@gmail.com

1 Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

Provisioning AWS Infrastructure with Terraform?

In this tutorial, we are going to discuss in detail on how to provision AWS infrastructure with Terraform.

Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform can determine what changed and create incremental execution plans which can be applied.

With Terraform installed, you're ready to create your first infrastructure.

You will provision an Amazon Machine Image (AMI) on Amazon Web Services (AWS) in this learning since AMIs are widely used.

Prerequisites

To follow this tutorial you will need:

- An AWS account
- The AWS CLI installed
- Your AWS credentials are configured locally.

1. With your account created and the CLI installed to configure the AWS CLI.

```
$ aws configure
```

1. Follow the prompts to input your AWS Access Key ID and Secret Access Key, which you'll find on this page.
2. The configuration process creates a file at `~/.aws/credentials` on macOS and Linux or `%UserProfile%\.aws\credentials` on Windows, where your credentials are stored.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

Note: This tutorial will provide resources that qualify under the AWS free-tier. If your account doesn't qualify under the AWS free-tier, we're not responsible for any charges that you may incur.

Write configuration

The set of files used to describe infrastructure in Terraform is known as a Terraform *configuration*. You'll write your first configuration now to launch a single AWS EC2 instance.

Each configuration should be in its own directory. Create a directory for the new configuration.

```
$ mkdir learn-terraform-aws-instance
```

Change into the directory.

```
$ cd learn-terraform-aws-instance
```

Create a file for the configuration code.

```
$ touch example.tf
```

Paste the configuration below into example.tf and save it. Terraform loads all files in the working directory that end in .tf.

```
provider "aws" {
  profile = "default"
  region  = "us-east-1"
}resource "aws_instance" "example" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}
```

This is a complete configuration that Terraform is ready to apply. In the following sections, we'll review each block of the configuration in more detail.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

Providers

The `provider` block configures the named provider, in our case `aws`, which is responsible for creating and managing resources. A provider is a plugin that Terraform uses to translate the API interactions with the service. A provider is responsible for understanding API interactions and exposing resources. Because Terraform can interact with any API, you can represent almost any infrastructure type as a resource in Terraform.

The `profile` an attribute in your provider block refers to Terraform to the AWS credentials stored in your AWS Config File, which you created when you configured the AWS CLI. HashiCorp recommends that you never hard-code credentials into `*.tf` configuration files. We are explicitly defining the `default` AWS config profile here to illustrate how Terraform should access sensitive credentials.

Note: If you leave out your AWS credentials, Terraform will automatically search for saved API credentials (for example, in `~/.aws/credentials`) or IAM instance profile credentials. This is cleaner when files are checked into source control or if there is more than one admin user.

Multiple provider blocks can exist if a Terraform configuration manages resources from different providers. You can even use multiple providers together. For example, you could pass the ID of an AWS instance to a monitoring resource from DataDog.

Resources

The `resource` the block defines a piece of infrastructure. A resource might be a physical component such as an EC2 instance, or it can be a logical resource such as a Heroku application.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

The resource block has two strings before the block: the resource type and the resource name. In the example, the resource type is aws_instance and the name is example. The prefix of the type maps to the provider. In our case, "aws_instance" automatically tells Terraform that it is managed by the "aws" provider.

The arguments for the resource are within the resource block. The arguments could be things like machine sizes, disk image names, or VPC IDs. Our providers reference documents the required and optional arguments for each resource provider. For your EC2 instance, you specified an AMI for Ubuntu and requested a t2.micro instance so you qualify under the free tier.

Initialize the directory

When you create a new configuration — or check out an existing configuration from version control — you need to initialize the directory with terraform init.

Terraform uses a plugin-based architecture to support hundreds of infrastructure and service providers. Initializing a configuration directory downloads and installs providers used in the configuration, which in this case is the aws provider. Subsequent commands will use local settings and data during initialization.

Initialize the directory.

```
$ terraform init
Initializing the backend...Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (terraform-providers/aws) 2.10.0...The following providers do not have any version constraints in configuration,
so the latest version was installed.To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.* provider.aws: version = "~> 2.10"Terraform has been successfully
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

```
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.
```

Terraform downloads the `aws` provider and installs it in a hidden subdirectory of the current working directory. The output shows which version of the plugin was installed.

Format and validate the configuration

We recommend using consistent formatting in files and modules written by different teams. The `terraform fmt` the command automatically updates configurations in the current directory for easy readability and consistency.

Format your configuration. Terraform will return the names of the files it formatted. In this case, your configuration file was already formatted correctly, so Terraform won't return any file names.

```
$ terraform fmt
```

If you are copying configuration snippets or just want to make sure your configuration is syntactically valid and internally consistent, the built-in `terraform validate` the command will check and report errors within modules, attribute names, and value types.

Validate your configuration. If your configuration is valid, Terraform will return a success message.

```
$ terraform validate
Success! The configuration is valid.
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

Create infrastructure

In the same directory as the `example.tf` the file you created, run `terraform apply`. You should see an output similar to the one shown below, though we've truncated some of the output to save space.

Note: Terraform 0.11 and earlier require running `terraform plan` before `terraform apply`. Use `terraform version` to confirm your running version.

```
$ terraform apply## ... Output truncated ...An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + createTerraform will perform the following actions:# aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                          = "ami-12345678"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)## ... Output truncated ...Plan: 1 to add, 0 to change, 0 to destroy.
```

Tip: If your configuration fails to apply, you may have customized your region or removed your default VPC. Refer to the troubleshooting section at the bottom of this guide for help.

This output shows the *execution plan*, describing which actions Terraform will take to change real infrastructure to match the configuration.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

The output format is similar to the diff format generated by tools such as Git. The output has a `+` next to `aws_instance.example`, meaning that Terraform will create this resource. Beneath that, it shows the attributes that will be set. When the value displayed is, it means that the value won't be known until the resource is created.

Terraform will now pause and wait for your approval before proceeding. If anything in the plan seems incorrect or dangerous, it is safe to abort here with no changes made to your infrastructure.

In this case, the plan is acceptable, so type `yes` at the confirmation prompt to proceed. Executing the plan will take a few minutes since Terraform waits for the EC2 instance to become available.

```
## ... Output truncated ...Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.Enter a value: yesaws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 1m50s [id=i-0bbf06244e44211d1]Apply
complete! Resources: 1 added, 0 changed, 0 destroyed.
```

You've now created infrastructure using Terraform! Visit the EC2 console to see the created EC2 instance. Make sure you're looking at the same region that was configured in the provider configuration!

Inspect state

When you applied your configuration, Terraform wrote data into a file called `terraform.tfstate`. This file now contains the IDs and properties of the resources Terraform created so that it can manage or destroy those resources going forward.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

You must save your state file securely and distribute it only to trusted team members who need to manage your infrastructure. In production, we recommend storing your state remotely. Remote stage storage enables collaboration using Terraform but is beyond the scope of this tutorial.

Inspect the current state using terraform show.

```
$ terraform show
# aws_instance.example:
resource "aws_instance" "example" {
    ami                          = "ami-12345678"
    arn                          = "arn:aws:ec2:us-east-1:130490850807:instance/i-
0bbf06244e44211d1"
    associate_public_ip_address  = true
    availability_zone            = "us-east-1b"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-0bbf0gdhjuske789"
    instance_state               = "running"
    instance_type                = "t2.micro"
    ipv6_address_count           = 0
    ipv6_addresses               = []
    monitoring                   = false
    primary_network_interface_id = "eni-0f1ceashsji56h076"
    private_dns                  = "ip-172-31-69-121.ec2.internal"
    private_ip                   = "172.31.61.141"
    public_dns                   = "ec2-54-124-14-244.compute-1.amazonaws.com"
    public_ip                    = "54-124-14-244"
    security_groups              = [
        "default",
    ]
    source_dest_check            = true
    subnet_id                    = "subnet-1ffgj87d5"
    tenancy                      = "default"
    volume_tags                  = {}
    vpc_security_group_ids       = [
        "sg-5255f429",
```


Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

```
  ]credit_specification {
    cpu_credits = "standard"
  }root_block_device {
    delete_on_termination = false
    iops                  = 100
    volume_id             = "vol-0079esdjbs567fg"
    volume_size           = 8
    volume_type           = "gp1"
  }
}
```

When Terraform created this EC2 instance, it also gathered a lot of information about it. These values can be referenced to configure other resources or outputs, which we discuss more later on in this track.

Manually Managing State

Terraform has a built-in command called `terraform state` for advanced state management. For example, if you have a long state file, you may want a list of the resources in the state, which you can get by using the `list` subcommand.

```
$ terraform state list
aws_instance.example
```

Troubleshooting

If `terraform validate` was successful and your apply still failed, you may be encountering a common error.

- **If you use a region other than `us-east-1`**, you will also need to change yours `ami` since AMI IDs are region-specific. Choose an AMI ID specific to your region by following these instructions, and modify `example.tf` with this ID. Then re-run `terraform apply`.

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com

- **If you do not have a default VPC in your AWS account in the correct region**, navigate to the AWS VPC Dashboard in the web UI, create a new VPC in your region, and associate a subnet and security group to that VPC. Then add the security group ID (vpc_security_group_ids) and subnet ID (subnet_id) into your aws_instance resource, and replace the values with the ones from your new security group and subnet.

```
resource "aws_instance" "example" {  ami = "ami-12345678"  instance_type        =
 "t2.micro" + vpc_security_group_ids = ["sg-0077..."] + subnet_id             = "subnet-...a..."
}}
```

- Save the changes to example.tf, and re-run terraform apply.
- Remember to add these lines to your configuration for the rest of the get started to track. For more information, review this document from AWS on working with VPCs.

For more information on the concepts we used in this learning:

- Terraform documentation - Read about the format of the configuration files
- Providers - Learn more about Terraform
- Use cases section - Find examples of Terraform configurations using multiple providers in the documentation
- Terraform blog post - Terraform: Beyond the Basics with AWS
- AWS Provider doc - To learn more about AWS authentication.
- CLI state command documentation - For more information about the terraform state command and subcommands for moving or removing resources from state

Stanley Stephen, M.C.A.,
Linkedin: https://www.linkedin.com/in/contactstanley/
Github: https://github.com/stanleymca/Learn_from_Stanley/AWS_Infra_with_Terraform.pdf
Email: s.stanley.mca@gmail.com