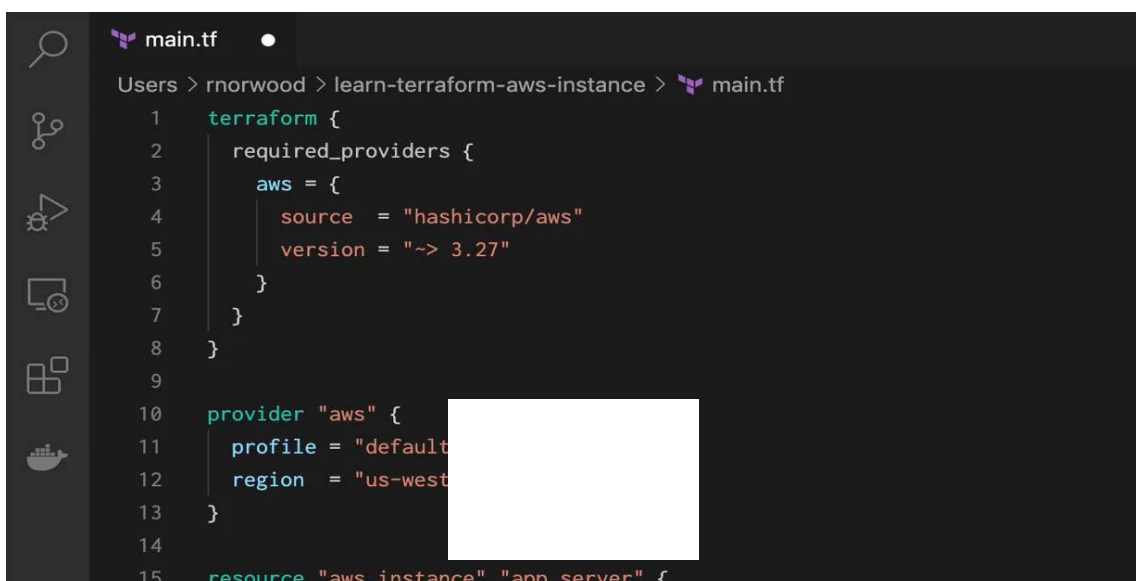


Change Infrastructure

🕒 4 MIN

PRODUCTS USED



```
main.tf
Users > rnorwood > learn-terraform-aws-instance > main.tf
1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 3.27"
6     }
7   }
8 }
9
10 provider "aws" {
11   profile = "default"
12   region = "us-west"
13 }
14
15 resource "aws_instance" "app_server" {
```

2:38

In the last tutorial, you created your first infrastructure with Terraform: a single EC2 instance on AWS. In this tutorial, you will modify that resource, and learn how to apply changes to your Terraform projects.

Infrastructure is continuously evolving, and Terraform helps you manage that change. As you change Terraform configurations, Terraform builds an execution plan that only modifies what is necessary to reach your desired state.

When using Terraform in production, we recommend that you use a version control system to manage your configuration files, and store your state in a remote backend such as Terraform Cloud or Terraform Enterprise.

Prerequisites

This tutorial assumes that you are continuing from the previous tutorials. If not, follow the steps below before continuing.

- Install the Terraform CLI (0.14.9+), and the AWS CLI (configured with a default profile), as [described in the last tutorial](#).
- Create a directory named `learn-terraform-aws-instance` and paste the following configuration into a file named `main.tf`.

```
terraform {  
  required_providers {  
    aws = {  
      source  = "hashicorp/aws"  
      version = "~> 3.27"  
    }  
  }  
  
  required_version = ">= 0.14.9"  
}  
  
provider "aws" {  
  profile = "default"  
  region  = "us-west-2"  
}  
  
resource "aws_instance" "app_server" {  
  ami           = "ami-830c94e3"  
  instance_type = "t2.micro"  
  
  tags = {  
    Name = "ExampleAppServerInstance"  
  }  
}
```

Copy 

- Initialize the configuration.

```
$ terraform init
```

Copy 

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

```
$ terraform apply
```

Copy 

Once you have successfully applied the configuration, you can continue with the rest of this tutorial.

Configuration

Now update the `ami` of your instance. Change the `aws_instance.app_server` resource under the provider block in `main.tf` by replacing the current AMI ID with a new one.

Tip: The below snippet is formatted as a diff to give you context about which parts of your configuration you need to change. Replace the content displayed in red with the content displayed in green, leaving out the leading `+` and `-` signs.

Note: The new AMI ID used in this configuration is specific to the `us-west-2` region. If you are working in a different region, be sure to select an appropriate AMI for that region by following [these instructions](#).

```
resource "aws_instance" "app_server" {  
-   ami           = "ami-830c94e3"  
+   ami           = "ami-08d70e59c07c61a3a"  
    instance_type = "t2.micro"  
}
```

This update changes the AMI to an Ubuntu 16.04 AMI. The AWS provider knows that it cannot change the AMI of an instance after it has been created, so Terraform will destroy the old instance and create a new one.

Apply Changes

After changing the configuration, run `terraform apply` again to see how Terraform will apply this change to the existing resources.

```
$ terraform apply
```

Copy 

```
aws_instance.app_server: Refreshing state... [id=i-01e03375ba238b38
```

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

```
# aws_instance.app_server must be replaced
-/+ resource "aws_instance" "app_server" {
    ~ ami                        = "ami-830c94e3" -> "ami-08d76
    ~ arn                      = "arn:aws:ec2:us-west-2:56165
##...
```

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

```
Do you want to perform these actions?
```

```
Terraform will perform the actions described above.
```

```
Only 'yes' will be accepted to approve.
```

```
Enter a value:
```

The prefix `-/+` means that Terraform will destroy and recreate the resource, rather than updating it in-place. Terraform can update some attributes in-place (indicated with the `~` prefix), but changing the AMI

for an EC2 instance requires recreating it. Terraform handles these details for you, and the execution plan displays what Terraform will do.

Additionally, the execution plan shows that the AMI change is what forces Terraform to replace the instance. Using this information, you can adjust your changes to avoid destructive updates if necessary.

Once again, Terraform prompts for approval of the execution plan before proceeding. Answer `yes` to execute the planned steps.

```
Enter a value: yes
```

```
aws_instance.app_server: Destroying... [id=i-01e03375ba238b384]
aws_instance.app_server: Still destroying... [id=i-01e03375ba238b384]
aws_instance.app_server: Still destroying... [id=i-01e03375ba238b384]
aws_instance.app_server: Still destroying... [id=i-01e03375ba238b384]
aws_instance.app_server: Still destroying... [id=i-01e03375ba238b384]
aws_instance.app_server: Destruction complete after 42s
aws_instance.app_server: Creating...
aws_instance.app_server: Still creating... [10s elapsed]
aws_instance.app_server: Still creating... [20s elapsed]
aws_instance.app_server: Still creating... [30s elapsed]
aws_instance.app_server: Still creating... [40s elapsed]
aws_instance.app_server: Creation complete after 50s [id=i-0fd4a359]
```

```
Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

As indicated by the execution plan, Terraform first destroyed the existing instance and then created a new one in its place. You can use `terraform show` again to have Terraform print out the new values associated with this instance.

Was this tutorial helpful?

NEXT

Destroy Infrastructure

>

<

PREVIOUS

Build Infrastructure



- System Status
- Cookie Manager
- Terms of Use
- Security
- Privacy

stdin: is not a tty