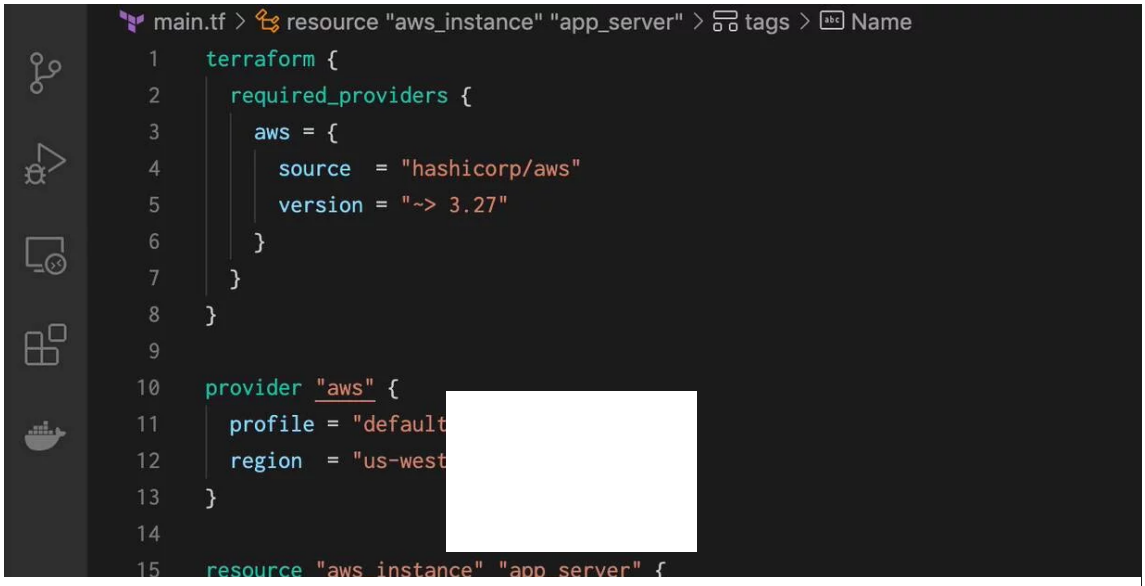
Define Input Variables

3 MIN PRODUCTS USED Terraform



2:41

You now have enough Terraform knowledge to create useful configurations, but the examples so far have used hard-coded values. Terraform configurations can include variables to make your configuration more dynamic and flexible.

Prerequisites

After following the other tutorials in this collection, you will have a directory named `learn-terraform-aws-instance` with the following configuration in a file called `main.tf`.

```
terraform {
```

Copy

```
  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-08d70e59c07c61a3a"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

Ensure that your configuration matches this, and that you have run `terraform init` in the `learn-terraform-aws-instance` directory.

Set the instance name with a variable

The current 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.

Copy

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

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

In `main.tf`, update the `aws_instance` resource block to use the new variable. The `instance_name` variable block will default to its default value ("ExampleAppServerInstance") unless you declare a different value.

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

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

Apply your configuration

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

Copy

```
$ terraform apply

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

Terraform will perform the following actions:
```

```
  # aws_instance.app_server will be created
  + resource "aws_instance" "app_server" {
      + ami                          = "ami-08d70e59c07c61a3a"
      + arn                          = (known after apply)
##...

Plan: 1 to add, 0 to change, 0 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: yes

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-0bf9549]

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

Now apply the configuration again, this time overriding the default instance name by passing in a variable using the `-var` flag. Terraform will update the instance's `Name` tag with the new name. Respond to the confirmation prompt with `yes`.

```
$ terraform apply -var "instance_name=YetAnotherName"
aws_instance.app_server: Refreshing state... [id=i-0bf954919ed765de

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_instance.app_server will be updated in-place
```

Copy

```
  ~ resource "aws_instance" "app_server" {
        id                           = "i-0bf954919ed765de1"
      ~ tags                         = {
          ~ "Name" = "ExampleAppServerInstance" -> "YetAnotherName"
        }
        # (26 unchanged attributes hidden)




        # (4 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 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: yes

aws_instance.app_server: Modifying... [id=i-0bf954919ed765de1]
aws_instance.app_server: Modifications complete after 7s [id=i-0bf9

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

Setting variables via the command-line will not save their values. Terraform supports many ways to use and set variables so you can avoid having to enter them repeatedly as you execute commands. To learn more, follow our in-depth tutorial, Customize Terraform Configuration with Variables.

Was this tutorial helpful?

Yes No

NEXT
Query Data with Outputs

PREVIOUS
Destroy Infrastructure