


Query Data Sources

 developer.hashicorp.com/terraform/tutorials/certification-associate-tutorials/data-sources

Terraform data sources let you dynamically fetch data from APIs or other Terraform state backends. Examples of data sources include machine image IDs from a cloud provider or Terraform outputs from other configurations. Data sources make your configuration more flexible and dynamic and let you reference values from other configurations, helping you scope your configuration while still referencing any dependent resource attributes. In Terraform Cloud, workspaces let you share data between workspaces.

In this tutorial, you will use data sources to make your configuration more dynamic. First, you will use Terraform to create an AWS VPC and security groups. Next, you will use the `aws_availability_zones` data source to make your configuration deployable across any region. You will then deploy application infrastructure defined by a separate Terraform configuration, and use the `terraform_remote_state` data source to query information about your VPC. Finally, you will use the `aws_ami` data source to configure the correct AMI for the current region.

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 [Get Started collection](#) first. If you are new to Terraform Cloud, complete the [Terraform Cloud 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 [Get Started collection](#) first.

For this tutorial, you will need:

- Terraform v1.2+ installed locally.
- the [AWS CLI](#).

- AWS Credentials configured for use with Terraform.

Note: Some of the infrastructure in this tutorial may not qualify for the AWS free tier. Destroy the infrastructure at the end of the guide to avoid unnecessary charges. We are not responsible for any charges that you incur.

Clone example repositories

The example configuration for this tutorial is hosted in two GitHub repositories.

1. The **VPC repository** contains the configuration to deploy a VPC and security groups for your application.

Clone the VPC repository.

```
$ git clone https://github.com/hashicorp/learn-terraform-data-sources-vpc.git
```

2. The **application repository** contains the configuration to deploy an example application consisting of a load balancer and an EC2 instance.

Clone the application repository.

```
$ git clone https://github.com/hashicorp/learn-terraform-data-sources-app.git
```

Initialize VPC workspace

Change to the VPC repository directory.

```
$ cd learn-terraform-data-sources-vpc
```

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-data-sources-vpc** workspace in your Terraform Cloud organization.

```
$ terraform init
Initializing modules...
```

```
Initializing Terraform Cloud...
```

```
Initializing provider plugins...
```

- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.17.1

```
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.



learn-terraform-data-sources-vpc/terraform.tf

```
terraform {
  /*
  cloud {
    workspaces {
      name = "learn-terraform-data-sources-vpc"
    }
  }
  */

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

  required_version = ">= 1.2"
}
```

Initialize your 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.
```

Update VPC region

The VPC configuration uses a variable called `aws_region` with a default value of `us-east-1` to set the region.

However, changing the value of the `aws_region` variable will not successfully change the region because the VPC configuration includes an `azs` argument to set Availability Zones, which is a hard-coded list of availability zones in the `us-east-1` region.



learn-terraform-data-sources-vpc/main.tf

```
module "vpc" {
  ##...
  azs          = ["us-east-1a", "us-east-1b", "us-east-1c", "us-east-1d", "us-
east-1e"]
  ##...
}
```

Use the `aws_availability_zones` data source to load the available AZs for the current region. Add the following to `main.tf`.



learn-terraform-data-sources-vpc/main.tf

```
data "aws_availability_zones" "available" {
  state = "available"

  filter {
    name   = "zone-type"
    values = ["availability-zone"]
  }
}
```

The `aws_availability_zones` data source is part of the AWS provider and retrieves a list of availability zones based on the arguments supplied. In this case, the `state` argument limits the availability zones to only those that are currently available.

You can reference data source attributes with the pattern `data.<NAME>.<ATTRIBUTE>`. Update the VPC configuration to use this data source to set the list of availability zones.



learn-terraform-data-sources-vpc/main.tf

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.14.0"

  cidr = var.vpc_cidr_block

  azs                = data.aws_availability_zones.available.names
  private_subnets   = slice(var.private_subnet_cidr_blocks, 0, 2)
  public_subnets    = slice(var.public_subnet_cidr_blocks, 0, 2)

  ##...
}
```

Configure the VPC workspace to output the region, which the application workspace will require as an input. Add a data source to `main.tf` to access region information.



learn-terraform-data-sources-vpc/main.tf

```
data "aws_region" "current" { }
```

Add an output for the region to `outputs.tf`.



learn-terraform-data-sources-vpc/outputs.tf

```
output "aws_region" {
  description = "AWS region"
  value       = data.aws_region.current.name
}
```

Create infrastructure

Apply this configuration, setting the value of `aws_region` to `us-west-1`. Respond to the confirmation prompt with a `yes`.

```

$ terraform apply -var aws_region=us-west-1
##...

Plan: 34 to add, 0 to change, 0 to destroy.

##...

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

  Enter a value: yes

##...

Apply complete! Resources: 34 added, 0 changed, 0 destroyed.

Outputs:

app_security_group_ids = [
  "sg-00fc397fb1066b140",
]
aws_region = "us-west-1"
lb_security_group_ids = [
  "sg-0ab0e3a1416bac068",
]
private_subnet_ids = [
  "subnet-0e9855907f0bab6f4",
  "subnet-074a96820b50023bc",
]
public_subnet_ids = [
  "subnet-0303938fcbcd0d16",
  "subnet-012e5c8724dfa5a0e",
]

```

Tip: In this scenario, you could use the `aws_region` variable to define the output parameter instead of using the data source. However, there are multiple ways to configure the AWS region. Using the `aws_region` data source will get the AWS provider's current region no matter how it was configured.

Configure Terraform remote state

Now that you deployed your network resources, go to the `learn-terraform-data-sources-app` directory.

```
$ cd ../learn-terraform-data-sources-app
```

This directory contains the Terraform configuration for your application.

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

```
$ terraform init
```

Initializing modules...

Initializing Terraform Cloud...

Initializing provider plugins...

- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Using previously-installed hashicorp/random v3.3.1
- Using previously-installed hashicorp/aws v4.17.1

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.



learn-terraform-data-sources-app/terraform.tf

```
terraform {
  /*
  cloud {
    workspaces {
      name = "learn-terraform-data-sources-app"
    }
  }
  */

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

  required_version = ">= 1.2"
}
```

Initialize your 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.
```

Like the VPC workspace, this configuration includes hard-coded values for the `us-east-1` region. You can use the `terraform_remote_state` data source to use another Terraform workspace's output data.

Add a `terraform_remote_state` data source to the `main.tf` file inside the `learn-terraform-data-sources-app` directory, replacing `YOUR_ORG` with your own Terraform Cloud organization name.



learn-terraform-data-sources-app/main.tf

```
data "terraform_remote_state" "vpc" {
  backend = "remote"

  config = {
    organization = "YOUR_ORG"
    workspaces = {

      name      = "learn-terraform-data-sources-vpc"
    }
  }
}
```

This remote state block uses the remote backend to load state data from the workspace and organization in the `config` section.

For security purposes, you must explicitly allow Terraform Cloud workspaces to access one another's state. To allow your `learn-terraform-data-sources-app` workspace to access your `learn-terraform-data-sources-vpc` workspace's state:

1. Log in to [Terraform Cloud](#).
2. Choose the organization you are using for this tutorial.
3. Navigate to your `learn-terraform-data-sources-vpc` workspace.
4. In the workspace's **General Settings**, find the **Remote state sharing** section.
5. Select the **Share with specific workspaces** option and choose the `learn-terraform-data-sources-app` workspace.
6. Click the **Save settings** button.



learn-terraform-data-sources-app/main.tf


```
data "terraform_remote_state" "vpc" {
  backend = "local"

  config = {
    path = "../learn-terraform-data-sources-vpc/terraform.tfstate"
  }
}
```

This remote state block uses the local backend to load state data from the path in the `config` section. Terraform remote state also supports a remote backend type for use with remote systems, such as Terraform Cloud or Consul.

Now, update your `aws` provider configuration in `main.tf` to use the same region as the VPC configuration instead of a hardcoded region.



learn-terraform-data-sources-app/main.tf

```
provider "aws" {
  region = data.terraform_remote_state.vpc.outputs.aws_region
}
```

The VPC configuration also included outputs for subnet and security group IDs. Configure the load balancer security group and subnet arguments for the `elb` module with those values.



learn-terraform-data-sources-app/main.tf

```
module "elb_http" {
  ##...
  security_groups = data.terraform_remote_state.vpc.outputs.lb_security_group_ids
  subnets        = data.terraform_remote_state.vpc.outputs.public_subnet_ids
  ##...
}
```

Note: Terraform's remote state data source can only load "root-level" output values from the source workspace, it cannot directly access values from resources or modules in the source workspace. To retrieve those values, you must add a corresponding output to the source workspace.

Scale EC2 instances

You can use values from data sources just like any other Terraform values, including by passing them to functions. The configuration in `main.tf` only uses a single EC2 instance. Update the configuration to use the `instances_per_subnet` variable to provision multiple EC2 instances per subnet.



learn-terraform-data-sources-app/main.tf

```
resource "aws_instance" "app" {
  ##...
  count = var.instances_per_subnet *
  length(data.terraform_remote_state.vpc.outputs.private_subnet_ids)

  ami = "ami-04d29b6f966df1537"
  ##...
}
```

Now when you apply this configuration, Terraform will provision `var.instances_per_subnet` instances for each private subnet configured in your VPC workspace.

Configure region-specific AMIs

The AWS instance configuration also uses a hard-coded AMI ID, which is only valid for the `us-east-1` region. Use an `aws_ami` data source to load the correct AMI ID for the current region. Add the following to `main.tf`.



learn-terraform-data-sources-app/main.tf

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

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

Replace the hard-coded AMI ID with the one loaded from the new data source.



learn-terraform-data-sources-app/main.tf

```
resource "aws_instance" "app" {
  count = var.instances_per_subnet *
  length(data.terraform_remote_state.vpc.outputs.private_subnet_ids)

  ami = data.aws_ami.amazon_linux.id

  ##...
}
```

Configure EC2 subnet and security groups

Finally, update the EC2 instance configuration to use the subnet and security group configuration from the VPC workspace.



learn-terraform-data-sources-app/main.tf

```
resource "aws_instance" "app" {  
  ##...  
  
  subnet_id =  
  data.terraform_remote_state.vpc.outputs.private_subnet_ids[count.index %  
length(data.terraform_remote_state.vpc.outputs.private_subnet_ids)]  
  vpc_security_group_ids =  
  data.terraform_remote_state.vpc.outputs.app_security_group_ids  
  
  ##...  
}
```

Apply the configuration and Terraform will provision the application infrastructure.
Respond to the confirmation prompt with a **yes**.

```
$ terraform apply  
##...
```

Plan: 10 to add, 0 to change, 0 to destroy.

```
Changes to Outputs:  
+ lb_url = (known after apply)  
+ web_instance_count = 4
```

```
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
```

```
##...
```

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.

Outputs:

```
lb_url = "http://lb-D0f-tutorial-example-1971328425.us-west-2.elb.amazonaws.com/"  
web_instance_count = 4
```

After a few minutes, the load balancer health checks will pass, and will return the example response.

```
$ curl $(terraform output -raw lb_url)  
<html><body><div>Hello, world!</div></body></html>
```

Tip: It can take several minutes for the load balancer to become available. If the curl command returns an error, try again after a few minutes.

Clean up your infrastructure

Before moving on, destroy the infrastructure you created in this tutorial.

In the application directory, destroy the application infrastructure. Respond to the confirmation prompt with **yes**.

```
$ terraform destroy
##...
```

Plan: 0 to add, 0 to change, 10 to destroy.

Changes to Outputs:

```
- lb_url = "http://lb-D0f-tutorial-example-1971328425.us-west-2.elb.amazonaws.com/" -> null
- web_instance_count = 4 -> null
```

Do you really want to destroy all resources in workspace "learn-terraform-data-sources-app"?

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

```
##...
```

Destroy complete! Resources: 10 destroyed.

Note: You must destroy the application workspace before the VPC workspace. Since the resources in the application workspace depend on those in the VPC workspace, the AWS API will return an error if you attempt to destroy the VPC first.

Now change to the VPC directory.

```
$ cd ../learn-terraform-data-sources-vpc
```

Destroy this infrastructure as well. Once again, respond to the confirmation prompt with **yes**.

```
$ terraform destroy -var aws_region=us-west-1
```

```
##...
```

```
Plan: 0 to add, 0 to change, 34 to destroy.
```

```
Changes to Outputs:
```

```
- app_security_group_ids = [
  - "sg-0214d055921c25c8e",
] -> null
- aws_region              = "us-west-2" -> null
- lb_security_group_ids  = [
  - "sg-03f34e1dd93483bd9",
] -> null
- private_subnet_ids     = [
  - "subnet-07b77ef2e9c386a17",
  - "subnet-098f226b620943eac",
] -> null
- public_subnet_ids      = [
  - "subnet-034fc6327ae353f",
  - "subnet-0a9a7558a4eaa4640",
] -> null
```

```
Do you really want to destroy all resources?
```

```
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
```

```
##...
```

```
Destroy complete! Resources: 34 destroyed.
```

If you used Terraform Cloud for this tutorial, after destroying your resources, delete the [learn-terraform-data-sources-vpc](#) and [learn-terraform-data-sources-app](#) workspaces from your Terraform Cloud organization.

Next steps

In this tutorial, you used data sources to make your configuration more dynamic. You deployed two separate configurations for your application and network resources and used the `terraform_remote_state` data source to share data between them. You also replaced region-specific configuration with dynamic values from AWS provider data sources.

Now that you have used Terraform data sources, check out the following resources for more information.

- Read the [Terraform Data Sources documentation](#).
- [Connect Terraform Cloud Workspaces](#) with run triggers, and use outputs from one workspace to configure another workspace.
- [Inject secrets into Terraform using the Vault provider](#).