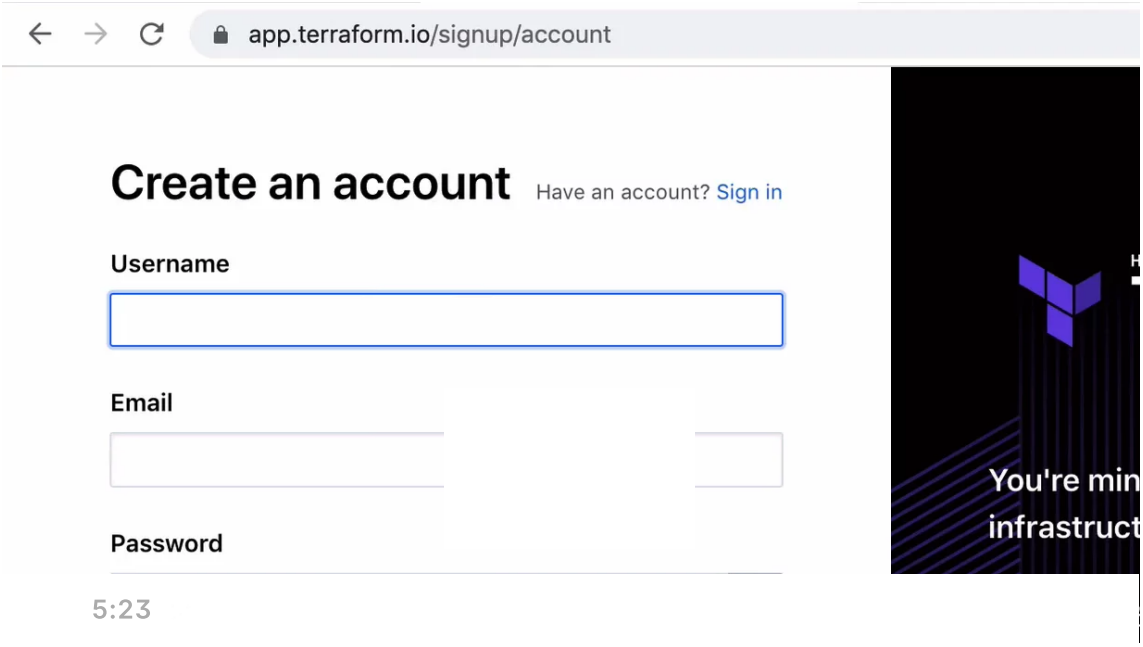
Store Remote State

7 MIN | PRODUCTS USED Terraform

This tutorial also appears in: Associate Tutorials.



Now you have built, changed, and destroyed infrastructure from your local machine. This is great for testing and development, but in production environments you should keep your state secure and encrypted, where your teammates can access it to collaborate on infrastructure. The best way to do this is by running Terraform in a remote environment with shared access to state.

Remote backends allow Terraform to use a shared storage space for state data. The Terraform Cloud remote backend also allows teams to easily version, audit, and collaborate on infrastructure changes. Terraform Cloud

also securely stores variables, including API tokens and access keys. It provides a safe, stable environment for long-running Terraform processes.

In this tutorial you will migrate your state to Terraform Cloud.

Prerequisites

This tutorial assumes that you have completed the previous tutorials in this collection. If you have not, create a directory named `learn-terraform-aws-instance` and paste this code into a file named `main.tf`.

Copy

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

Run `terraform init` to initialize your configuration directory and download the required providers. It is safe to re-run this command even if you have already done so in this directory.

Copy

```
$ terraform init
```

Next, apply your configuration. Type `yes` to confirm the proposed changes.

```
$ terraform apply
```

Copy

Terraform will provision an AWS EC2 instance and store data about the resource in a local state file.

Configure the remote backend

If you do not have a Terraform Cloud account, please sign up for free here and create an organization. Make a note of the organization's name. For more detailed instructions on how to sign up and create an organization, see this tutorial.

Next, modify `main.tf` to add a remote backend block to your Terraform configuration, and replace `<ORG_NAME>` with your organization name.

```
terraform {
+ backend "remote" {
+   organization = "<ORG_NAME>"
+   workspaces {
+     name = "Example-Workspace"
+   }
+ }
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }
}
```

Login to Terraform Cloud

Next, log into your Terraform Cloud account with the Terraform CLI in your terminal.

```
$ terraform login
Terraform will request an API token for app.terraform.io using your

If login is successful, Terraform will store the token in plain tex
the following file for use by subsequent commands:
    /Users/<USER>/.terraform.d/credentials.tfrc.json

Do you want to proceed?
  Only 'yes' will be accepted to confirm.

  Enter a value:
```

Confirm with a `yes` and follow the workflow in the browser window that will automatically open. You will need to paste the generated API key into your Terminal when prompted. For more detail on logging in, follow the Authenticate the CLI with Terraform Cloud tutorial.

Initialize Terraform

Now that you have configured your remote backend, run `terraform init` to re-initialize your configuration and migrate your state file to Terraform Cloud. Enter "yes" when prompted to confirm the migration.

```
$ terraform init

Initializing the backend...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "local"
  newly configured "remote" backend. No existing state was found in
  configured "remote" backend. Do you want to copy this state to th
```

```
  backend? Enter "yes" to copy and "no" to start with an empty stat
  Enter a value: yes

Releasing state lock. This may take a few moments...

Successfully configured the backend "remote"! Terraform will automa
use this backend unless the backend configuration changes.
...
```

Now that Terraform has migrated the state file to Terraform Cloud, delete the local state file.

```
$ rm terraform.tfstate
```

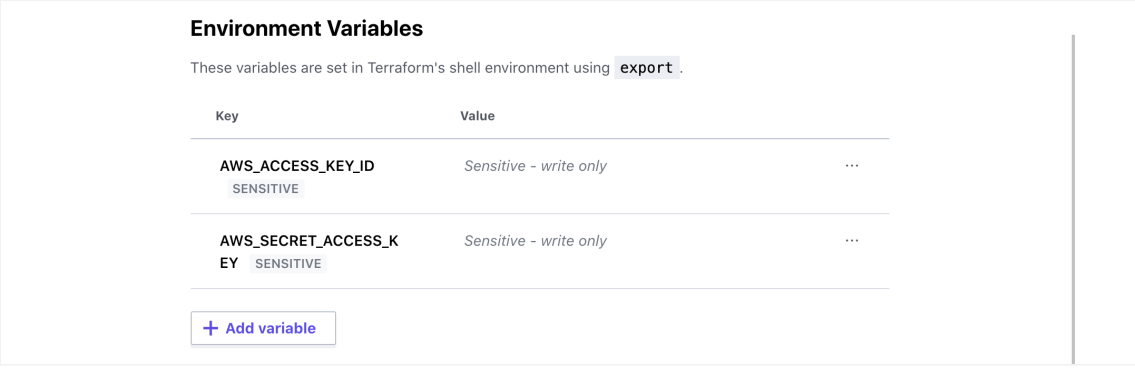
Copy

When using Terraform Cloud as a remote backend with the CLI-driven workflow, you can choose to have Terraform run remotely, or on your local machine. When using local execution, Terraform Cloud will execute Terraform on your local machine and remotely store your state file in Terraform Cloud. For this tutorial, you will use the remote execution mode.

Set workspace variables

The `terraform init` step created the `Example-Workspace` workspace in your Terraform Cloud organization. Your workspace needs to be configured with your AWS credentials to authenticate the AWS provider.

Navigate to your `Example-Workspace` workspace in Terraform Cloud and select the "Variables" tab. Add your `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` in the "Environment Variables" section, making sure to mark them as "Sensitive".



Apply the configuration

Now, run `terraform apply` to trigger a run in Terraform Cloud. Terraform will show that there are no changes to be made.

```
$ terraform apply

## ...
No changes. Infrastructure is up-to-date.
```

Copy

This means that Terraform did not detect any differences between your configuration and real physical resources that exist. As a result, Terraform doesn't need to do anything.

Terraform is now storing your state remotely in Terraform Cloud. Remote state storage makes collaboration easier and keeps state and secret information off your local disk. Remote state is loaded only in memory when it is used.

If you want to move back to local state, you can remove the backend configuration block from your configuration and run `terraform init` again. Terraform will once again ask if you want to migrate your state back to local.

Destroy your infrastructure

Make sure to run `terraform destroy` to clean up the resources you created in these tutorials. Terraform will execute this run in the remote backend and stream the output to your terminal window. When prompted, remember to confirm with a `yes`. You can also confirm the operation by visiting your workspace in the Terraform Cloud web UI and confirming the run.

```
$ terraform destroy
Running apply in the remote backend. Output will stream here. Press
will cancel the remote apply if it's still pending. If the apply st
will stop streaming the logs, but will not stop the apply running r

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/hashicorp-learn/rita-test/runs/run-AcG

Waiting for the plan to start...

Terraform v0.15.3
on linux_amd64
Configuring remote state backend...
Initializing Terraform configuration...
aws_instance.app_server: Refreshing state... [id=i-039d6d420ad46af1

Terraform used the selected providers to generate the following exe
plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_instance.app_server will be destroyed
```

Next Steps

This concludes the getting started tutorials for Terraform. Now you can use Terraform to create and manage your infrastructure.

For more hands-on experience with the Terraform configuration language, resource provisioning, or importing existing infrastructure, review the tutorials below.

- Configuration Language - Get more familiar with variables, outputs, dependencies, meta-arguments, and other language features to write more sophisticated Terraform configurations.
- Modules - Organize and re-use Terraform configuration with modules.
- Provision - Use Packer or Cloud-init to automatically provision SSH keys and a web server onto a Linux VM created by Terraform in AWS.
- Import - Import existing infrastructure into Terraform.

To read more about available configuration options, explore the Terraform documentation.

Learn more about Terraform Cloud

Although Terraform Cloud can act as a standard remote backend to support Terraform runs on local machines, it works even better as a remote run environment. It supports two main workflows for performing Terraform runs:

- A VCS-driven workflow, in which it automatically queues plans whenever changes are committed to your configuration's VCS repo.
- An API-driven workflow, in which a CI pipeline or other automated tool can upload configurations directly.

For a hands-on introduction to the Terraform Cloud VCS-driven workflow, follow the Terraform Cloud getting started tutorials. Terraform Cloud also

offers commercial solutions which include team permission management, policy enforcement, agents, and more.

Was this tutorial helpful?

Yes No

PREVIOUS
Query Data with Outputs