Migrate State to Terraform Cloud

developer.hashicorp.com/terraform/tutorials/certification-associate-tutorials/cloud-migrate

When using open source Terraform, you are responsible for maintaining a state file as the source of truth for your cloud infrastructure. Terrafrom Cloud offers secure remote state storage, make it easier to collaborate on infrastructure development. You can migrate your state to Terraform Cloud without interrupting services or recreating your existing infrastructure.

In this tutorial, you will migrate a local state file to Terraform Cloud.

Warning: When uploading a state file to Terraform Cloud using the steps in this tutorial, always use the same version of the Terraform CLI you used to create the resources. Using a newer version of Terraform may update the state file and cause state file corruption.

Prerequisites

This tutorial assumes that you have the following:

- The Terraform CLI version 1.1 or higher installed locally
- A Terraform Cloud account

Note: Terraform versions older than 1.1 use the `remote` backend block to configure the CLI workflow and migrate state.

Create state

Clone the example configuration for this tutorial. This configuration uses the `random` provider to generate a random pet name.

```
$ git clone https://github.com/hashicorp/learn-state-migration
```

Next, change into the directory.

```
$ cd learn-state-migration
```

Open `main.tf` to review the configuration. It uses an input varaible to determine the length of the generated string and outputs the value.

main.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.3.2"
    }
  }
  required_version = ">= 1.1.0"
}


variable "name_length" {
  description = "The number of words in the pet name"
  default     = "3"
}

resource "random_pet" "pet_name" {
  length    = var.name_length
  separator = "-"
}

output "pet_name" {
  value = random_pet.pet_name.id
}
```

Initialize the directory.

```
$ terraform init
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/random from the dependency lock file
- Installing hashicorp/random v3.3.2...
- Installed hashicorp/random v3.3.2 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```

Now apply the configuration, typing `yes` at the prompt to confirm the operation.

```
$ terraform apply

Terraform used the selected providers to generate the following execution plan.
Resource actions are
indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # random_pet.pet_name will be created
  + resource "random_pet" "pet_name" {
      + id        = (known after apply)
      + length    = 3
      + separator = "-"
    }

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

Changes to Outputs:
  + pet_name = (known after apply)

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

random_pet.pet_name: Creating...
random_pet.pet_name: Creation complete after 0s [id=mostly-joint-lacewing]

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

Outputs:

pet_name = "mostly-joint-lacewing"
```

Terraform displays the generated name in the outputs.

Configure Terraform Cloud integration

Now that you have a local state file, you need to create a `cloud` code block in your configuration.

To use Terraform Cloud as a backend for your configuration, you must include a `cloud` block in your configuration.

Add the `cloud` block to your configuration as shown below, replacing `ORGANIZATION-NAME` with your own Terraform Cloud organization name.

main.tf

```
terraform {
  cloud {
    organization = "ORGANIZATION-NAME"
    workspaces {
      name = "learn-terraform-cloud-migrate"
    }
  }

  required_version = ">= 1.1.0"

  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.3.2"
    }
  }
}
```

While the organization defined in the `cloud` stanza must already exist, the workspace does not have to; Terraform Cloud will create it if necessary. If you use an existing workspace, it must not have any existing states.

Note: Terraform Cloud workspaces behave differently from Terraform CLI workspaces. Terraform CLI workspaces allow multiple state files to exist within a single directory, letting you use one configuration for multiple environments. Terraform Cloud workspaces contain everything needed to manage a given set of infrastructure, and function like separate working directories.

Authenticate with Terraform Cloud

After configuring your Terraform Cloud integration, you must authenticate to Terraform Cloud to use it for remote operations.

Run `terraform login` and follow the prompts to log in, typing `yes` at the confirmation prompt.

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

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

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

  Enter a value:
```

For more detailed instructions on logging in, review the login tutorial.

Migrate the state file

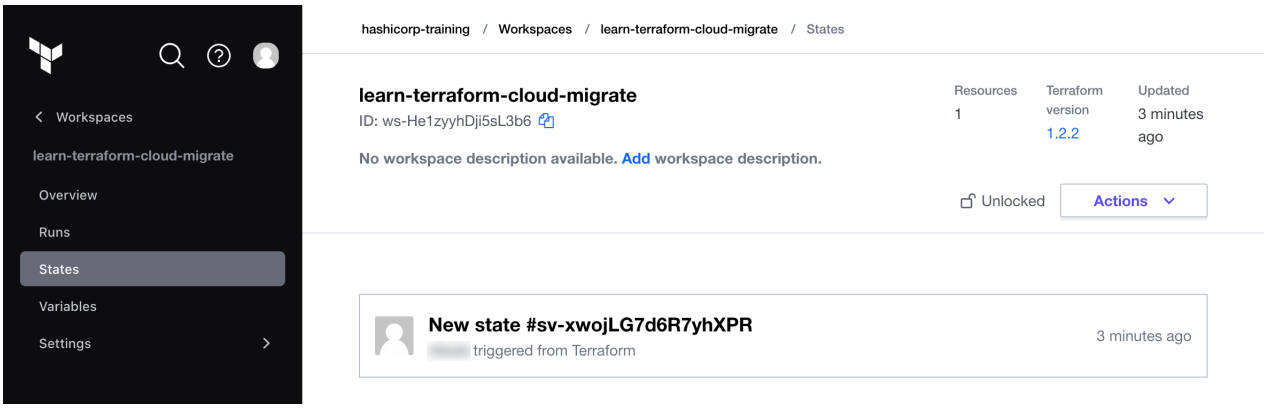
To migrate your existing state file to Terraform Cloud, you must reinitialize your configuration to update the backend.

Reinitialize your configuration. Terraform detects your updated backend and confirms that you wish to migrate your state file to Terraform Cloud. Type `yes` to confirm the migration.

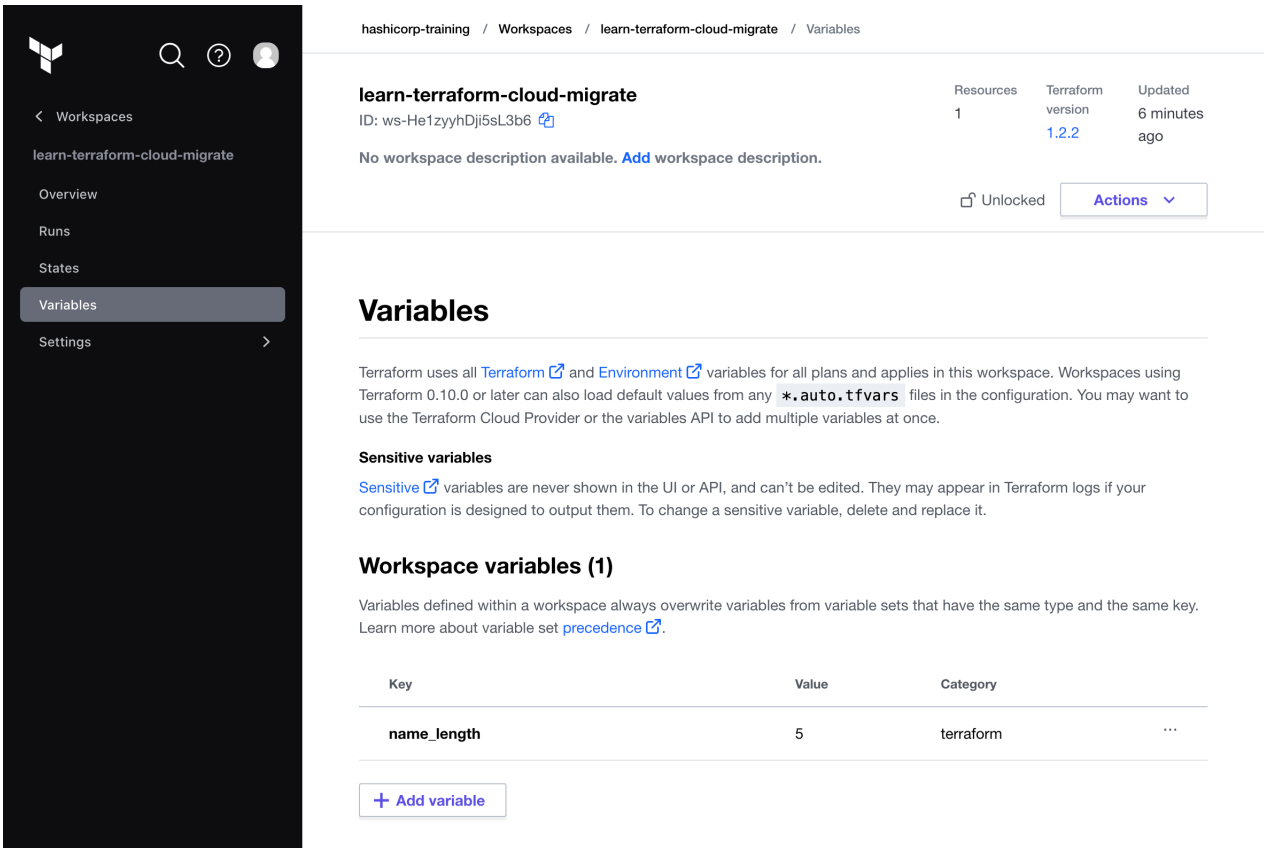
```
$ terraform init
Initializing Terraform Cloud...
Do you wish to proceed?
  As part of migrating to Terraform Cloud, Terraform can optionally copy your
  current workspace state to the configured Terraform Cloud workspace.

  Answer "yes" to copy the latest state snapshot to the configured
  Terraform Cloud workspace.

  Answer "no" to ignore the existing state and just activate the configured
  Terraform Cloud workspace with its existing state, if any.

  Should Terraform migrate your existing state?

  Enter a value: yes


Initializing provider plugins...
- Reusing previous version of hashicorp/random from the dependency lock file
- Using previously-installed hashicorp/random v3.0.1

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.
```

Configure the Terraform Cloud workspace

After migrating your state to Terraform Cloud, log in to the Terraform Cloud web UI and navigate to your `learn-terraform-cloud-migrate` workspace. Then, go to the workspace's **States** page. Terraform Cloud lists the state you migrated to your new workspace.



Your configuration relies on an input variable. Navigate to the workspace's **Variables** page and create a new Terraform variable named `name_length` with a value of `5`.



If the configuration relied on a cloud provider, you would set the provider credentials on this page as well.

Initiate a run in the new workspace

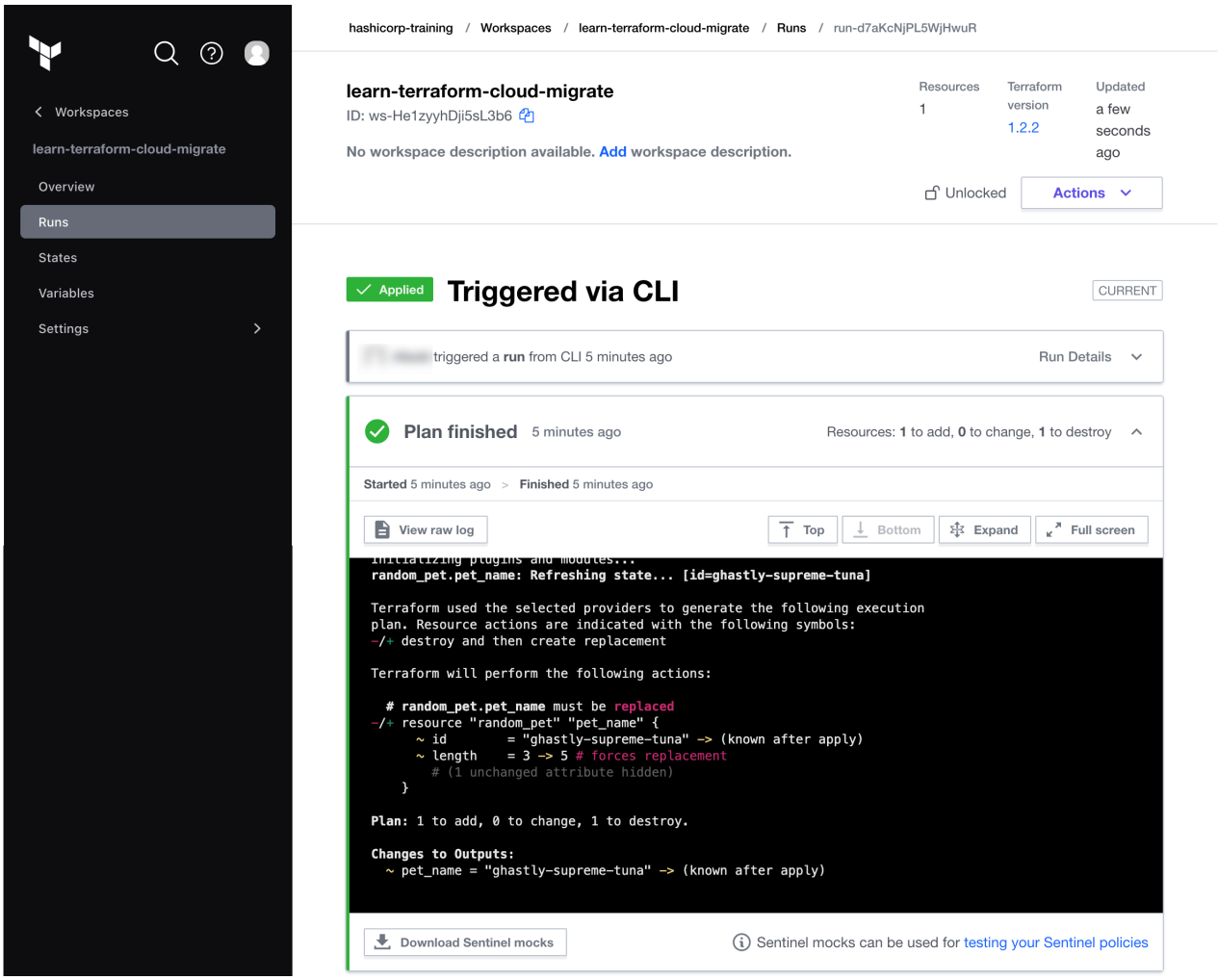
After verifying that Terraform migrated your state to Terraform Cloud, remove your local state file.

```
$ rm terraform.tfstate
```

Trigger a new run. Terraform will propose replacing your resource to reflect the update to the `name_length` input variable. Confirm the operation by typing `yes`.

```
$ terraform apply
Running apply in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will cancel the remote apply if it's still pending. If the apply started it
will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/hashicorp-training/learn-terraform-cloud-
migrate/runs/run-d7aKcNjPL5WjHwuR

Waiting for the plan to start...

Terraform v1.2.2
on linux_amd64
Initializing plugins and modules...
random_pet.pet_name: Refreshing state... [id=ghastly-supreme-tuna]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # random_pet.pet_name must be replaced
-/+ resource "random_pet" "pet_name" {
      ~ id        = "ghastly-supreme-tuna" -> (known after apply)
      ~ length    = 3 -> 5 # forces replacement
        # (1 unchanged attribute hidden)
    }

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

Changes to Outputs:
  ~ pet_name = "ghastly-supreme-tuna" -> (known after apply)

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

  Enter a value: yes

random_pet.pet_name: Destruction complete after 0s
random_pet.pet_name: Creating...
random_pet.pet_name: Creation complete after 0s [id=possibly-eminently-sadly-
inspired-mongoose]

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

Outputs:

pet_name = "possibly-eminently-sadly-inspired-mongoose"
```

Terraform streams the logs to your local console, and also displays the run details in the workspace UI.



Destroy your infrastructure

Run `terraform destroy` to clean up your resources.

```
$ terraform destroy
Running apply in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will cancel the remote apply if it's still pending. If the apply started it
will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/hashicorp-training/learn-terraform-cloud-
migrate/runs/run-StNegAY8UrBCT6FB

Waiting for the plan to start...

Terraform v1.2.2
on linux_amd64
Initializing plugins and modules...
random_pet.pet_name: Refreshing state... [id=possibly-eminently-sadly-inspired-
mongoose]

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

Terraform will perform the following actions:

  # random_pet.pet_name will be destroyed
  - resource "random_pet" "pet_name" {
      - id        = "possibly-eminently-sadly-inspired-mongoose" -> null
      - length    = 5 -> null
      - separator = "-" -> null
    }

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

Changes to Outputs:
  - pet_name = "possibly-eminently-sadly-inspired-mongoose" -> null

Do you really want to destroy all resources in workspace "learn-terraform-cloud-
migrate"?
  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

random_pet.pet_name: Destruction complete after 0s

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

You may also optionally delete your Terraform Cloud workspace from your workspace's settings page. Review the Destroy resources and workspace tutorial for detailed guidance.

Next steps

In this tutorial, you migrated a state file from your local machine to a Terraform Cloud workspace. To learn more about related concepts and Terraform Cloud features, review the following resources:

- Review the documentation on Migrating State from Multiple Local Workspaces
- Follow the tutorial on connecting workspaces using Terraform Cloud run triggers
- Learn how to manage permissions in Terraform Cloud