

Deploy Infrastructure with Terraform and CircleCI

- ❖ CircleCI is a continuous integration and delivery (CI/CD) platform for automating software builds, tests, and deployments.
- ❖ The CI/CD paradigm establishes version control repositories as the source of truth for your deployments.
- ❖ It also helps teams quickly ship new features and fixes by defining pipelines that help ensure the stability and resilience of your services through testing and automation.
- ❖ You can build deployment pipelines of varying complexity to satisfy your organization's requirements for production deployments.
- ❖ Using Terraform to manage your infrastructure as code enables the benefits of the CI/CD workflow for infrastructure deployments.
- ❖ Since your infrastructure is codified, your team can collaborate and review it and deploy it using automated pipelines instead of manual orchestration.
- ❖ To automate Terraform operations in a remote environment, you need to configure remote state storage so Terraform can access and manage your project's state across runs.
- ❖ In this hands-on project, you will use CircleCI and Terraform to deploy an S3-backed web application.
- ❖ You will configure and review an automated Terraform workflow and use Terraform Cloud for remote state storage.

Prerequisites

This project assumes that tutorial you are familiar with the Terraform and Terraform Cloud workflows. If you are new to Terraform, fork and complete projects in this repo https://github.com/joshking1/Terraform_Cloud.git.

To complete this hands-on project, you will need the following:

- A [GitHub account](#)
- A [CircleCI account](#). Sign up with your GitHub account so CircleCI can build and deploy from your GitHub repositories. Review the [CircleCI getting started guide](#) for an introduction to the workflow if you are unfamiliar.
- An [AWS account](#).
- A [Terraform Cloud account](#).

Create a Terraform Cloud token

To authenticate with Terraform Cloud to store your project's Terraform state, you need to configure your Terraform Cloud integration with a Terraform Cloud API token.

Navigate to your organization settings, then select the **Teams** page.

Tip: If you are using a free Terraform Cloud account, create a token for your default Owners team. Otherwise, choose or create a new team with permissions to manage workspaces.

Under the **Team API Token** section, click **Create a team token**.

Organization settings

General

Plan & billing

Tags

Teams

Users

Variable sets

Security

API tokens

Authentication

SSH keys

Version control

General

Events

Providers

Team: owners

Visibility

☐ Visible

Visible to every member of this organization

☒ Secret

Only visible to team members and organization owners

[Update team visibility](#)

Team API Token

Treat this token like a password, as it can be used to access your account without a username, password, or two-factor authentication.

No tokens created for this team.

[Create a team token](#)

Store this token in a secure place as Terraform Cloud will not display it again. Later in this tutorial, you will set an environment variable in your CircleCI project to this token value for your build to use.

Fork and clone example configuration

<https://github.com/joshking1/terraform-circle-ci-pipeline.git>

This repository contains example configuration to deploy Terramino, a Terraform-skinned Tetris game, to an AWS S3 bucket using Terraform and CircleCI.

Once you have forked the repository, clone it to your own machine.

Example, run a git clone as follow

```
# git clone https://github.com/joshking1/terraform-circle-ci-pipeline.git
```

Navigate to repository by running the command

```
# cd terraform-circle-ci-pipeline
```

Review Terraform configuration

Open the main.tf file.

The Terraform configuration in this repository deploys Terramino, a Terraform-skinned Tetris application, to an AWS S3 bucket.

Open the main.tf file.

The Terraform configuration in this repository deploys Terramino, a Terraform-skinned Tetris application, to an AWS S3 bucket.

main.tf

```
provider      {
  region = var.region

  default_tags {
    tags = {
      hashicorp-learn = "circleci"
    }
  }
}

resource      "randomid" {}

resource      "app" {
  tags = {
    Name = "App Bucket"
  }

  bucket      = "${var.  }.${var.  }.${random_uuid.randomid.result}"
  force_destroy = true
}
```

```
resource "aws_s3_bucket_object" "app" {
  acl      = "public-read"
  key      = "index.html"
  bucket   = aws_s3_bucket.app.id
  content  = file("./assets/index.html")
  content_type = "text/html"
}
```

```
resource "aws_s3_bucket" "bucket" {
  bucket = aws_s3_bucket.app.id
  acl    = "public-read"
}
```

```
resource "aws_s3_bucket_object" "terramino" {
  bucket = aws_s3_bucket.app.bucket

  index_document {
    suffix = "index.html"
  }

  error_document {
    key = "error.html"
  }
}
```

This configuration uses a terraform.tfvars file to set values for the input variables of your configuration. Open the terraform.tfvars file to review its contents.

```
terraform.tfvars
region = "us-east-1"
label  = "hashicorp"
app    = "terramino"
```

Next, open the terraform.tf file. The configuration in this file defines the required provider and Terraform versions for this configuration. It also includes an empty cloud block.

```
main.tf
terraform {
  cloud {}

  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 4.4.0"
    }
  }

  required_version = ">= 1.2.0"
}
```

The cloud block configures a Terraform Cloud integration for CLI-driven Terraform Cloud runs. As of Terraform 1.2.0, you can configure the cloud block using environment variables that let you dynamically determine which Terraform Cloud organization or workspace to deploy to. Terraform also lets you use an environment variable to pass a Terraform Cloud token to your CI/CD system. Later in this tutorial, you will configure environment variables in your CircleCI project to configure your Terraform Cloud integration.

Review CircleCI jobs and workflow

Navigate to the .circleci subdirectory

```
$ cd. circleci
```

Copy

CircleCI pipelines consist of jobs and workflows. A job is a collection of steps executed together. A workflow is a sequence of jobs that progresses based on the result of the individual steps. For example, your CircleCI configuration may define a testing job that runs tests on your infrastructure and a deployment job that ships your changes. You can then compose these jobs into a workflow, specifying that if the test job succeeds, CircleCI should trigger the deployment job.

Each job declares an executor, which defines the environment CircleCI will perform the job's steps in. All of the jobs in this configuration use the Docker executor and reference a Docker image that contains the latest version of the Terraform binary.

Open the `config.yml` file in your file editor to review the jobs and workflows for this configuration.

This configuration defines four jobs: `plan-apply`, `apply`, `plan-destroy`, and `destroy`. It composes these jobs into an automated Terraform workflow.

Review the plan-apply job

The `plan-apply` job copies your repository using the `checkout` step, runs `terraform init` to initialize your configuration, and generates a plan file named `tfapply` by running `terraform plan -out`.

The job uses the `hashicorp/terraform:light` Docker image, which contains the latest version of the Terraform binary. The `persist_to_workspace` step saves the initialized configuration and your environment variables for use in the following jobs. `persist_to_workspace` allows you to run this initialized Terraform

configuration throughout the rest of the jobs in the workflow as if they are running on the same machine.

.circleci/config.yml

jobs:

plan-apply:

working_directory: /tmp/project

docker:

- image: docker.mirror.hashicorp.services/hashicorp/terraform:light

steps:

- checkout

- run:

name: terraform init & plan

command: |

terraform init -input=false

terraform plan -out tfapply -var-file terraform.tfvars

- persist_to_workspace:

root: .

paths:

Review the apply job

The apply job invokes the attach_workspace step to load the persisted workspace from the plan job, giving this job access to its artifacts, including the execution plan captured in the tfapply file. It then runs terraform apply using the -auto-approve flag to avoid prompting for user input.

apply:

docker:

- image: docker.mirror.hashicorp.services/hashicorp/terraform:light

steps:

- attach_workspace:

at: .


```
- run:
  name: terraform
  command: |
    terraform apply -auto-approve tfapply
- persist_to_workspace:
  root: .
  paths:
```

Review the plan-destroy and destroy jobs

Similarly, to the plan-apply and apply jobs, the plan-destroy job creates an execution plan and the destroy job executes that saved plan to remove all of the infrastructure tracked in the project's state file. It uses the persisted workspace to access the execution plan across jobs.

plan-destroy:

```
docker:
- image: docker.mirror.hashicorp.services/hashicorp/terraform:light
steps:
- attach_workspace:
  at: .
- run:
  name: terraform create destroy plan
  command: |
    terraform plan -destroy -out tfdestroy -var-file terraform.tfvars
- persist_to_workspace:
  root: .
  paths:
    - .
```

destroy:

```
docker:
- image: docker.mirror.hashicorp.services/hashicorp/terraform:light
```

```
steps:
- attach_workspace:
  at: .
- run:
  name: terraform destroy
  command: |
    terraform apply -auto-approve tfdestroy
```

You should not run these jobs unattended because it may lead to disruption to your services.

Review the workflow

The last block in the configuration composes the jobs into a workflow. Workflows define order, precedence, and dependencies to perform the jobs within the pipeline.

```
workflows:
  version: 2
  plan_approve_apply:
    jobs:
      - plan-apply
      - hold-apply:
          type: approval
          requires:
            - plan-apply
      - apply:
          requires:
            - hold-apply
      - plan-destroy:
          requires:
            - apply
      - hold-destroy:
          type: approval
          requires:
            - plan-destroy
```

- **destroy:**
requires:
 - hold-destroy

Notice that hold-apply and hold-destroy have type: approval in this workflow. This means that for every run, CircleCI will generate a plan and wait for approval before running the apply job. CircleCI will also generate a plan to destroy the infrastructure and wait for approval before running the destroy job.

Create Terraform Cloud workspace

This configuration uses Terraform Cloud for your project's state storage. You will create a new workspace to use for this project and configure it for local execution. When using local execution with Terraform Cloud, the Terraform operations occur in the environment that runs the Terraform CLI (in this case, the Docker executor configured for your build), and Terraform Cloud stores the state file for shared access across builds and runs.

In the Terraform Cloud UI, create a new CLI-driven workspace named learn-terraform-circleci.

On the workspace overview page, click on the current Execution Mode to navigate to the general settings.

The screenshot shows the Terraform Cloud workspace overview page for a workspace named "learn-terraform-circleci". The workspace ID is "ws-szwMU6cgnYuQQr4u". It shows 0 resources, Terraform version 1.2.0, and was updated a few seconds ago. The page has tabs for Overview, Runs, States, Variables, and Settings. The Overview tab is active. Below the tabs, there's a section titled "Waiting for configuration" with a status "Checking for configuration". A message states: "This workspace currently has no Terraform configuration files associated with it. Terraform Cloud is waiting for the configuration to be uploaded." On the right, there are settings for "Execution mode" (set to "Remote" and highlighted with a red box) and "Auto apply" (set to "Off").

learn-terraform-circleci
ID: ws-szwMU6cgnYuQQr4u [🔗](#)

No workspace description available. [Add workspace description.](#)

Resources: 0 Terraform version: 1.2.0 Updated: a few seconds ago

Overview Runs States Variables Settings ▾ 🔓 Unlocked Actions ▾

Waiting for configuration Checking for configuration 🔄

This workspace currently has no Terraform configuration files associated with it. Terraform Cloud is waiting for the configuration to be uploaded.

⚡ Execution mode: **Remote** ⚙️ Auto apply: Off

Execution Mode

If you change the execution mode any in progress runs will be discarded.

☐ **Remote**

Your plans and applies occur on Terraform Cloud's infrastructure. You and your team have the ability to review and collaborate on runs within the app.

☒ **Local**

Your plans and applies occur on machines you control. Terraform Cloud is only used to store and synchronize state.

☐ **Agent**

Terraform Cloud will manage the plans and applies your agents execute.

Remote state sharing

Choose whether this workspace should share state with the entire organization, or only with specific approved workspaces. The `terraform_remote_state` data source relies on state sharing to access workspace outputs.

☒ **Share with specific workspaces**

Select workspaces to share with

☐ **Share with all workspaces in this organization**

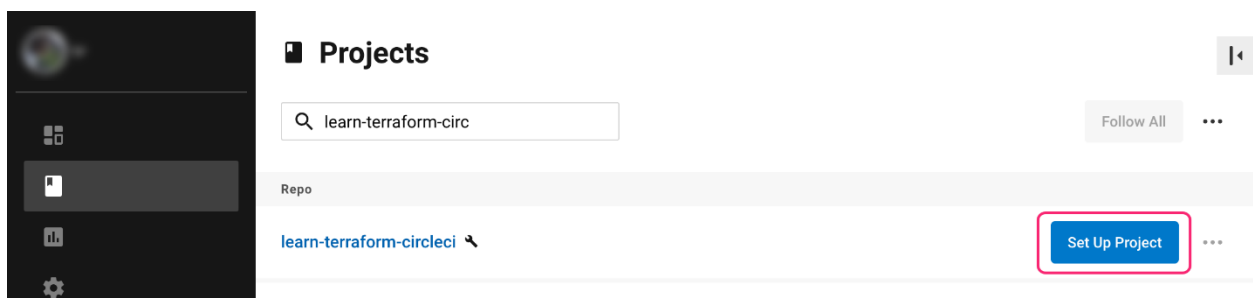
Save settings

Configure CircleCI project

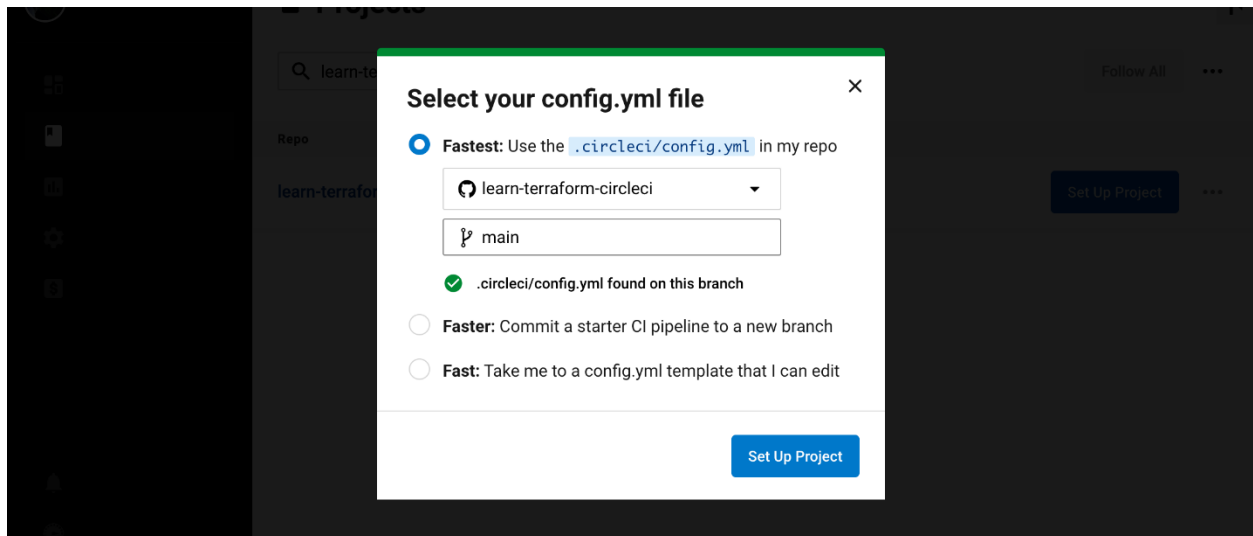
Navigate to your CircleCI dashboard. Make sure that you are in the correct organization with access to your GitHub account by confirming the organization in the top left corner.

Then, select Projects in the left sidebar.

Search for your forked learn-terraform-circleci repository. Then, click the Set Up Project button

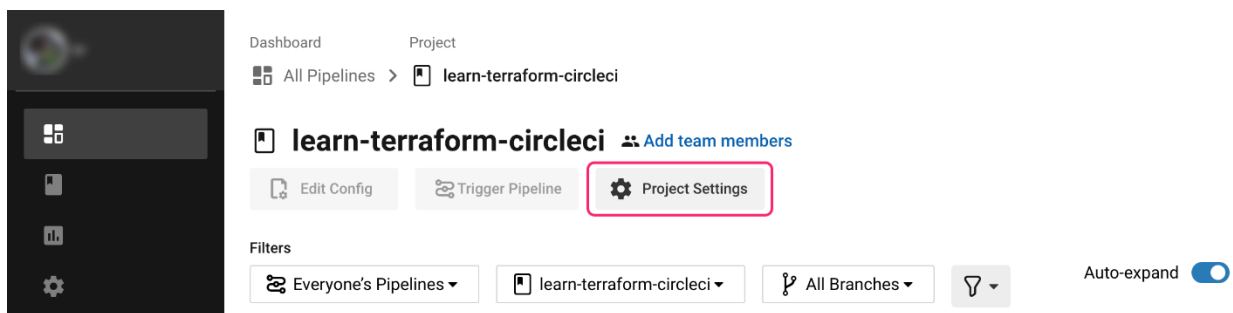


Select the **Fastest** configuration option to use the CircleCI configuration file in the repository. Enter the main branch as the branch to track. Then, click **Set Up Project**.



CircleCI will automatically attempt to run the job and fail because the project needs your AWS credentials and Terraform Cloud integration details.

Navigate to the project's **Project Settings**, then select **Environment Variables** from the sidebar.



Set the following environment variables, which Terraform will access in your build environment to configure both the AWS provider and the Terraform Cloud integration for your project:

1. Set `AWS_ACCESS_KEY_ID` to the generated key for the AWS user running this job. To generate an access key and secret access key file, log in to your AWS account and [create them in IAM](#).
2. Set `AWS_SECRET_ACCESS_KEY` to the secret access key you generated above.
3. Set `TF_CLOUD_ORGANIZATION` to your Terraform Cloud organization name
4. Set `TF_WORKSPACE` to `learn-terraform-circleci`, the Terraform Cloud workspace you created and configured earlier in this tutorial.
5. Set `TF_TOKEN_app_terraform_io` to the API token you created at the beginning of the tutorial.

When complete, your environment variables page will list the configured variables.

<

Project Settings

learn-terraform-circleci

Organization Settings

Overview

Triggers

Advanced

Environment Variables

SSH Keys

API Permissions

Jira Integration

Slack Integration

Insights Snapshot Badge

Status Badges

Webhooks

Environment Variables

Environment variables let you add sensitive data (e.g. API keys) to your jobs rather than placing them in the repository. The value of the variables cannot be read or edited in the app once they are set.

If you're looking to share environment variables across projects, try [Contexts](#).

Name	Value	Add Environment Variable	Import Variables
AWS_ACCESS_KEY_ID	xxxxUP6Q		×
AWS_SECRET_ACCESS_KEY	xxxxlBgR		×
TF_CLOUD_ORGANIZATION	xxxxning		×
TF_TOKEN_app_terraform_io	xxxxNHJI		×
TF_WORKSPACE	xxxxleci		×

Define your Terraform variables

In your file editor, open `terraform.tfvars` and update the `label` variable value to `hashicorp.fun`. Then, save the file.

terraform.tfvars

Copy

```
region = "us-east-1"
```

```
label = "hashicorp.fun"
```

```
app = "terramino"
```

Trigger CircleCI workflow

Stage your changes to your GitHub repository.

```
$ git add terraform.tfvars
```

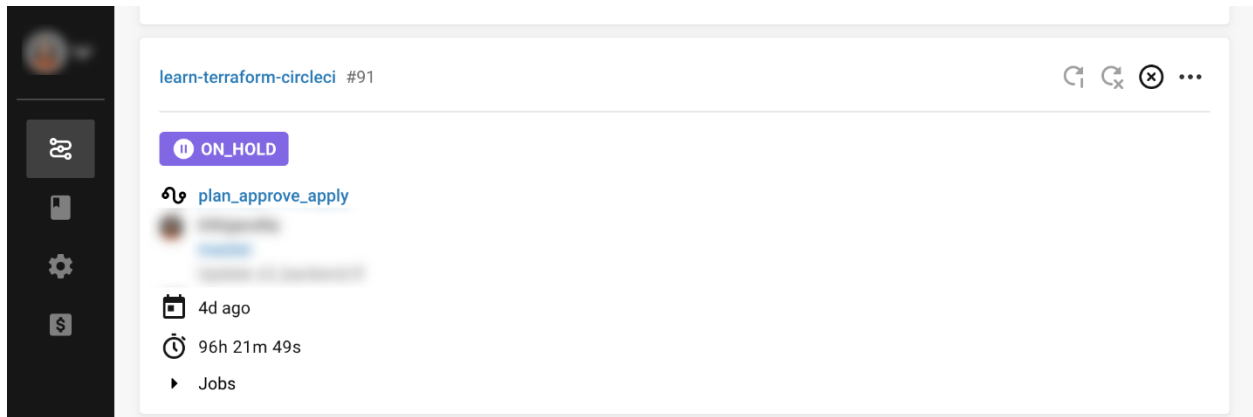
Commit these changes with a message.

```
$ git commit -m "Update variable definitions"
```

Finally, push these changes to your forked repository's main branch to kick off a CircleCI run.

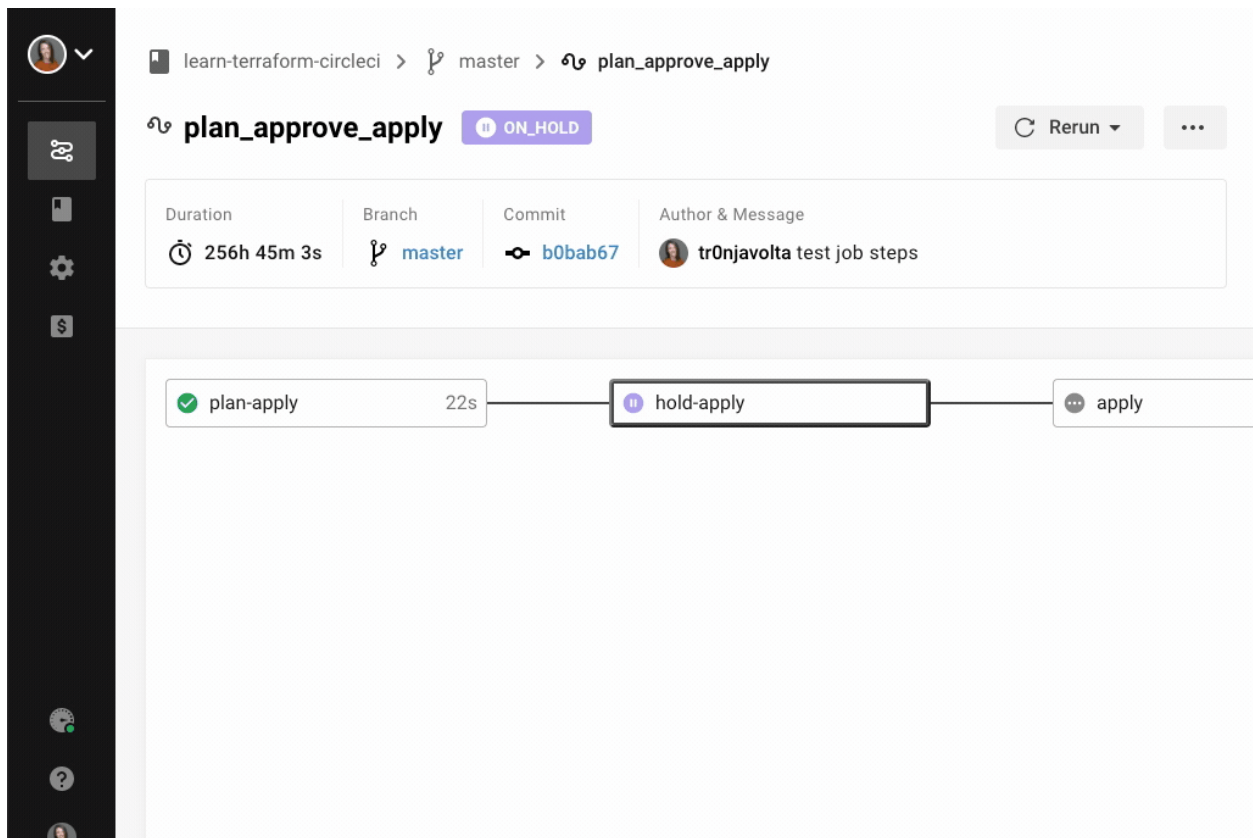
```
$ git push
```

The CircleCI web UI should indicate that your build started. The steps for this deployment will initialize your Terraform directory, plan the Terraform deployment, and wait for your approval to apply the planned changes.

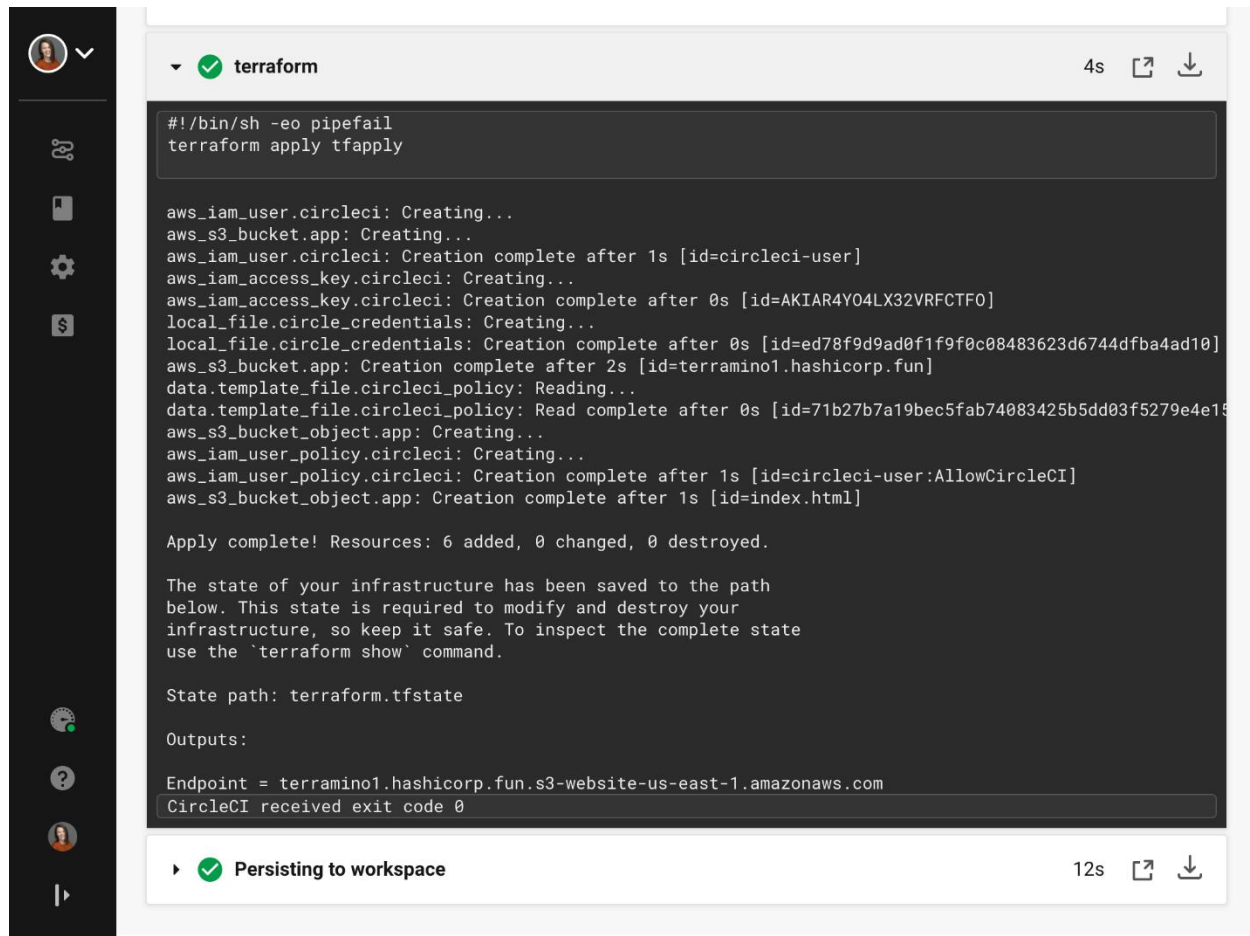


Review the output for the plan-apply job, which shows the proposed execution plan to create your resources. First, click on the **plan-apply** step in the workflow, then expand the **terraform init && plan** step to review the execution plan.

Then, return to the workflow overview by clicking the **plan_approve_apply** workflow at the top and approve it.



Once the deployment job is complete, your workflow will hold so that you can review the destroy plan before it starts the destroy job. Before you approve the destroy operation, review the output for the apply job and navigate to the endpoint in the output to test out your application.



```
#!/bin/sh -eo pipefail
terraform apply tfapply

aws_iam_user.circleci: Creating...
aws_s3_bucket.app: Creating...
aws_iam_user.circleci: Creation complete after 1s [id=circleci-user]
aws_iam_access_key.circleci: Creating...
aws_iam_access_key.circleci: Creation complete after 0s [id=AKIAR4Y04LX32VRFCTF0]
local_file.circle_credentials: Creating...
local_file.circle_credentials: Creation complete after 0s [id=ed78f9d9ad0f1f9f0c08483623d6744dfba4ad10]
aws_s3_bucket.app: Creation complete after 2s [id=terramino1.hashicorp.fun]
data.template_file.circleci_policy: Reading...
data.template_file.circleci_policy: Read complete after 0s [id=71b27b7a19bec5fab74083425b5dd03f5279e4e15]
aws_s3_bucket_object.app: Creating...
aws_iam_user_policy.circleci: Creating...
aws_iam_user_policy.circleci: Creation complete after 1s [id=circleci-user:AllowCircleCI]
aws_s3_bucket_object.app: Creation complete after 1s [id=index.html]

Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the 'terraform show' command.

State path: terraform.tfstate

Outputs:

Endpoint = terramino1.hashicorp.fun.s3-website-us-east-1.amazonaws.com
CircleCI received exit code 0
```

► Persisting to workspace 12s

Any changes to your GitHub repository will trigger another run of this workflow. As with any Terraform deployment, terraform will determine which resources to recreate or update in place depending on configuration and the provider.

Navigate to the Terraform Cloud workspace to verify that it stores the state for your project and lists the resources it manages on the workspace overview page.

learn-terraform-circleci

ID: ws-szwMU6cgnYuQQr4u

Resources

5

Terraform version

1.2.0

Updated

a few seconds ago

No workspace description available. [Add workspace description.](#)

Overview

States

Settings

Unlocked

Actions

Resources

5

Outputs

1

Current as of the most recent state version.

Filter resources

NAME	PROVIDER	TYPE	MODULE	UPDATED ↓
app	hashicorp/aws	aws_s3_bucke...	root	May 15 2022
bucket	hashicorp/aws	aws_s3_bucke...	root	May 15 2022
app	hashicorp/aws	aws_s3_bucke...	root	May 15 2022
terrmino	hashicorp/aws	aws_s3_bucke...	root	May 15 2022
randomid	hashicorp/rando...	random_uuid	root	May 15 2022

⚡ Execution mode: Local

⚙️ Auto apply: Off

Metrics

Metrics will appear once your next run is applied.

Tags (0)

Add a tag

Tags have not been added to this workspace.

Run triggers

Destroy the infrastructure

The plan-destroy step in the workflow generated and saved a plan to destroy your application. The hold-destroy job is a manual gate step that gives you to review the plan before the workflow destroys your resources. Click on the hold step and then choose "approve" to move on to the destroy job in this workflow.

While terraform is destroying your infrastructure, navigate to the plan-destroy job in the CircleCI web UI to observe the output in the **terraform create destroy plan** dropdown.