

<https://www.hashicorp.com>

The Core Terraform Workflow

[JUMP TO SECTION](#) ▾

The core Terraform workflow has three steps:

1. **Write** – Author infrastructure as code.
2. **Plan** – Preview changes before applying.
3. **Apply** – Provision reproducible infrastructure.

This guide walks through how each of these three steps plays out in the context of working as an individual practitioner, how they evolve when a team is collaborating on infrastructure, and how Terraform Cloud enables this workflow to run smoothly for entire organizations.

Working as an Individual Practitioner

Let's first walk through how these parts fit together as an individual working on infrastructure as code.

Write

You write Terraform configuration just like you write code: in your editor of choice. It's common practice to store your work in a version controlled repository even when you're just operating as an individual.

```
# Create repository
$ git init my-infra && cd my-infra

Initialized empty Git repository in ../../my-infra/.git/

# Write initial config
$ vim main.tf

# Initialize Terraform
$ terraform init

Initializing provider plugins...
# ...
Terraform has been successfully initialized!
```

As you make progress on authoring your config, repeatedly running plans can help flush out syntax errors and ensure that your config is coming together as you expect.

```
# Make edits to config
$ vim main.tf

# Review plan
$ terraform plan

# Make additional edits, and repeat
$ vim main.tf
```

This parallels working on application code as an individual, where a tight feedback loop between editing code and running test commands is useful.

Plan

When the feedback loop of the Write step has yielded a change that looks good, it's time to commit your work and review the final plan.

```
$ git add main.tf
$ git commit -m 'Managing infrastructure as code!'

[main (root-commit) f735520] Managing infrastructure as code!
1 file changed, 1 insertion(+)
```

Because `terraform apply` will display a plan for confirmation before proceeding to change any infrastructure, that's the command you run for final review.

```
$ terraform apply

An execution plan has been generated and is shown below.
# ...
```

Apply

After one last check, you are ready to tell Terraform to provision real infrastructure.

```
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: 1 added, 0 changed, 0 destroyed.
```

At this point, it's common to push your version control repository to a remote location for safekeeping.

```
$ git remote add origin https://github.com/*user*/repo*.git
$ git push origin main
```

This core workflow is a loop; the next time you want to make changes, you start the process over from the beginning.

Notice how closely this workflow parallels the process of writing application code or scripts as an individual? This is what we mean when we talk about Terraform enabling infrastructure as code.

Working as a Team

Once multiple people are collaborating on Terraform configuration, new steps must be added to each part of the core workflow to ensure everyone is working together smoothly. You'll see that many of these steps parallel the workflow changes we make when we work on application code as teams rather than as individuals.

Write

While each individual on a team still makes changes to Terraform configuration in their editor of choice, they save their changes to version control *branches* to avoid colliding with each other's work. Working in branches enables team members to resolve mutually incompatible infrastructure changes using their normal merge conflict workflow.

```
$ git checkout -b add-load-balancer  
  
Switched to a new branch 'add-load-balancer'
```

Running iterative plans is still useful as a feedback loop while authoring configuration, though having each team member's computer able to run them becomes more difficult with time. As the team and the infrastructure grows, so does the number of sensitive input variables (e.g. API Keys, SSL Cert Pairs) required to run a plan.

To avoid the burden and the security risk of each team member arranging all sensitive inputs locally, it's common for teams to migrate to a model in which Terraform operations are executed in a shared Continuous Integration (CI) environment. The work needed to create such a CI environment is nontrivial, and is outside the scope of this core workflow overview, but a full deep dive on this topic can be found in our [Running Terraform in Automation](https://learn.hashicorp.com/tutorials/terraform/automate-terraform?in=terraform/automation&utm_source=WEBSITE&utm_medium=WEB_IO&utm_offer=ARTICLE_PAGE&utm_content=DOCS) (https://learn.hashicorp.com/tutorials/terraform/automate-terraform?in=terraform/automation&utm_source=WEBSITE&utm_medium=WEB_IO&utm_offer=ARTICLE_PAGE&utm_content=DOCS) guide.

This longer iteration cycle of committing changes to version control and then waiting for the CI pipeline to execute is often lengthy enough to prohibit using speculative plans as a feedback loop while authoring individual Terraform configuration changes. Speculative plans are still useful before new Terraform changes are applied or even merged to the main development branch, however, as we'll see in a minute.


Plan

For teams collaborating on infrastructure, Terraform's plan output creates an opportunity for team members to review each other's work. This allows the team to ask questions, evaluate risks, and catch mistakes before any potentially harmful changes are made.

The natural place for these reviews to occur is alongside pull requests within version control--the point at which an individual proposes a merge from their working branch to the shared team branch. If team members review proposed config changes alongside speculative plan output, they can evaluate whether the intent of the change is being achieved by the plan.

The problem becomes producing that speculative plan output for the team to review. Some teams that still run Terraform locally make a practice that pull requests should include an attached copy of speculative plan output generated by the change author. Others arrange for their CI system to post speculative plan output to pull requests automatically.

Initial resources #1

 Open hashiadmin wants to merge 1 commit into `master` from `initial-resources`

Conversation 0 Commits 1 Files changed 1

hashiadmin commented 10 minutes ago

Adding initial resources for my application

Initial resources ... 5bfb27b

hashiadmin commented 2 minutes ago

Here is the Terraform plan output for this change:

```

-----

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:

+ null_resource.foo
  id: <computed>

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

-----

```

In addition to reviewing the plan for the proper expression of its author's intent, the team can also make an evaluation whether they want this change to happen now. For example, if a team notices that a certain change could result in service disruption, they may decide to delay merging its pull request until they can schedule a maintenance window.

Apply

Once a pull request has been approved and merged, it's important for the team to review the final concrete plan that's run against the shared team branch and the latest version of the state file.

This plan has the potential to be different than the one reviewed on the pull request due to issues like merge order or recent infrastructural changes. For example, if a manual change was made to your infrastructure since the plan was reviewed, the plan might be different when you merge.

It is at this point that the team asks questions about the potential implications of applying the change. Do we expect any service disruption from this change? Is there any part of this change that is high risk? Is there anything in our system that we should be watching as we apply this? Is there anyone we need to notify that this change is happening?

Depending on the change, sometimes team members will want to watch the apply output as it is happening. For teams that are running Terraform locally, this may involve a screen share with the team. For teams running Terraform in CI, this may involve gathering around the build log.

Just like the workflow for individuals, the core workflow for teams is a loop that plays out for each change. For some teams this loop happens a few times a week, for others, many times a day.

The Core Workflow Enhanced by Terraform Cloud

While the above described workflows enable the safe, predictable, and reproducible creating or changing of infrastructure, there are multiple collaboration points that can be streamlined, especially as teams and organizations scale. We designed Terraform Cloud to support and enhance the core Terraform workflow for anyone collaborating on infrastructure, from small teams to large organizations. Let's look at how Terraform Cloud makes for a better experience at each step.

Write

Terraform Cloud provides a centralized and secure location for storing input variables and state while also bringing back a tight feedback loop for speculative plans for config authors. Terraform configuration interacts with Terraform Cloud via the "remote" backend (</docs/language/settings/backends/remote.html>).

```
terraform {  
  backend "remote" {  
    organization = "my-org"  
    workspaces {  
      prefix = "my-app-"  
    }  
  }  
}
```

Once the backend is wired up, a Terraform Cloud API key is all that's needed by team members to be able to edit config and run speculative plans against the latest version of the state file using all the remotely stored input variables.

```
$ terraform workspace select my-app-dev  
Switched to workspace "my-app-dev".  
  
$ terraform plan  
  
Running plan remotely in Terraform Enterprise.  
  
Output will stream here. To view this plan in a browser, visit:  
  
https://app.terraform.io/my-org/my-app-dev/.../  
  
Refreshing Terraform state in-memory prior to plan...  
  
# ...  
  
Plan: 1 to add, 0 to change, 0 to destroy.
```

With the assistance of this plan output, team members can each work on authoring config until it is ready to propose as a change via a pull request.

Plan

Once a pull request is ready for review, Terraform Cloud makes the process of reviewing a speculative plan easier for team members. First, the plan is automatically run when the pull request is created. Status updates to the pull request indicate while the plan is in progress.

Once the plan is complete, the status update indicates whether there were any changes in the speculative plan, right from the pull request view.

Add a resource #4



hashiadmin wants to merge 1 commit into `master` from `adding-bar`



Conversation 0



Commits 1



Checks 0



Files changed 1



hashiadmin commented 28 minutes ago

Owner



Add a second resource, bar



Add a resource

Verified

✓ 4232b14

Add more commits by pushing to the `adding-bar` branch on `core-workflow-guide/my-app-terraform`.



All checks have passed

[Hide all checks](#)

1 successful check



atlas/my-org/my-app-terraform — Terraform plan: 1 to add, 0 to change, 0 to...

[Details](#)



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

For certain types of changes, this information is all that's needed for a team member to be able to approve the pull request. When a teammate needs to do a full review of the plan, clicking the link to Terraform Cloud brings up a view that allows them to quickly analyze the full plan details.

✓ PLANNED

Add a resource

hashiadmin triggered a pull request run (#3) from GitHub 3 minutes ago

Run Details

Plan finished 3 minutes ago

Resources: 1 to add, 0 to change, 0 to destroy

Queued 3 minutes ago > Started 3 minutes ago > Finished 3 minutes ago

View raw log

Top

Bottom

Expand

Full Screen

The refreshed state will be used to calculate this plan, but will not be persisted to local or remote state storage.

null_resource.foo: Refreshing state... (ID: 618339531673542557)

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:

- + null_resource.bar
id: <computed>

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

ⓘ

Heads up: This run was started from a [pull request](#) and cannot be applied. To apply these changes the pull request must first be merged into the default branch.

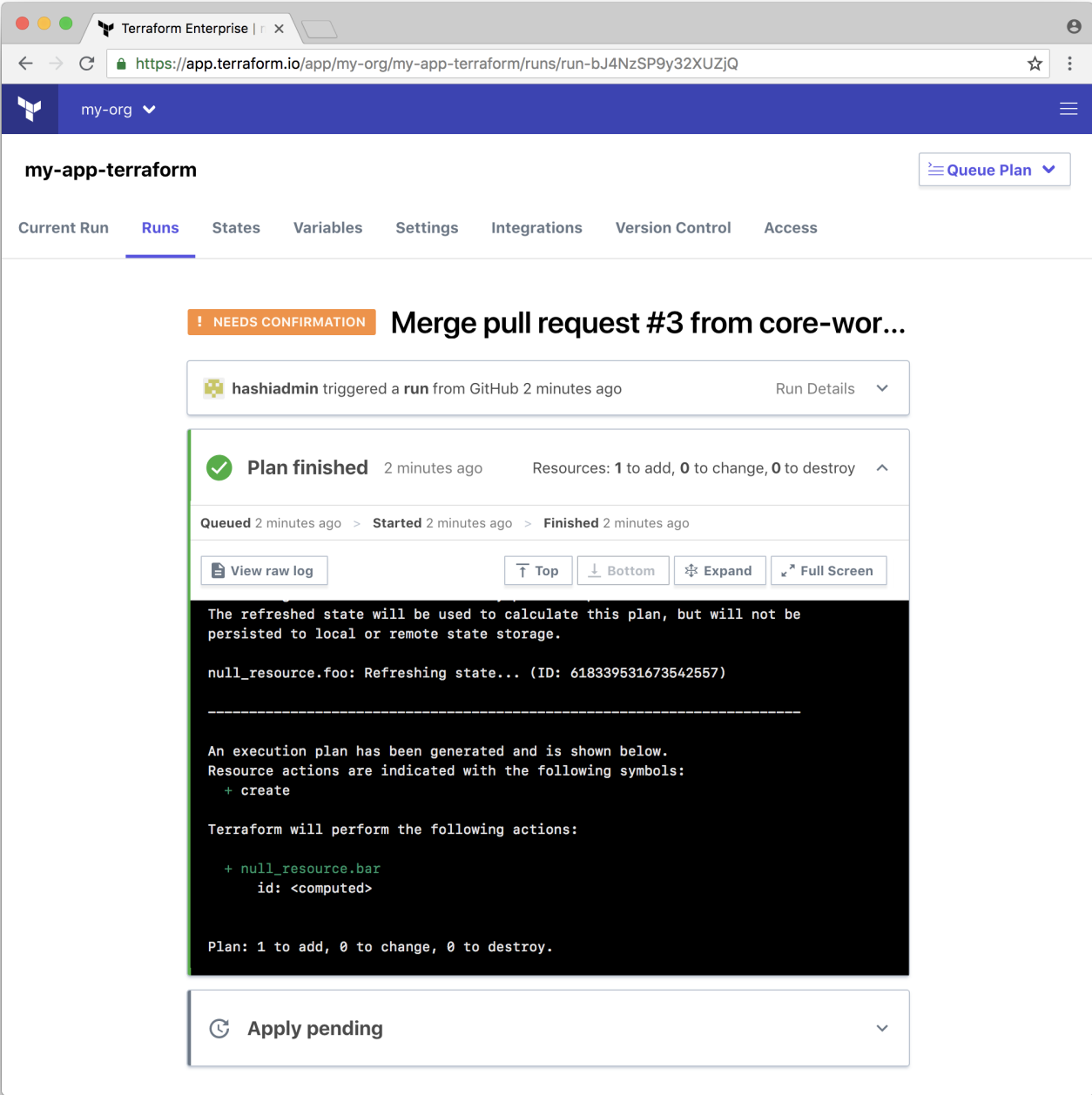
Comment: Leave feedback or record a decision.

Add Comment

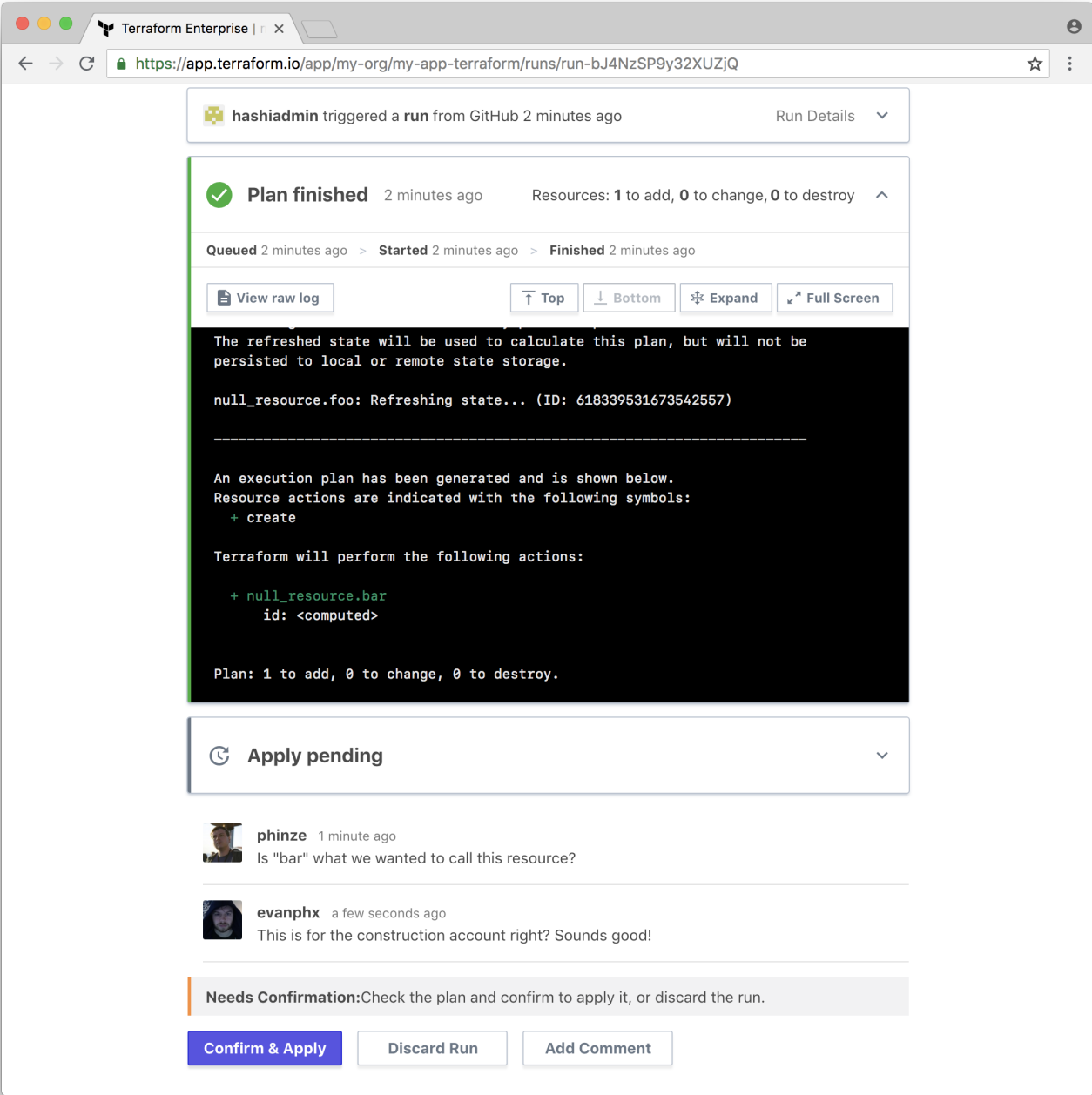
This page allows the reviewer to quickly determine if the plan is matching the config author’s intent and evaluate the risk of the change.

Apply

After merge, Terraform Cloud presents the concrete plan to the team for review and approval.



The team can discuss any outstanding questions about the plan before the change is made.



Once the Apply is confirmed, Terraform Cloud displays the progress live to anyone who'd like to watch.

Apply running a few seconds ago

Queued a few seconds ago > Started a few seconds ago

[View raw log](#) [Top](#) [Bottom](#) [Expand](#) [Full Screen](#)

```
Terraform v0.11.8

Initializing plugins and modules...
2018/08/17 22:08:32 [DEBUG] Using modified User-Agent: Terraform/0.11.8 TFE/d012b76
null_resource.bar: Creating...
null_resource.bar: Creation complete after 0s (ID: 3423346902457632441)

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

phinze 1 minute ago
Is "bar" what we wanted to call this resource?

evanphx a few seconds ago
This is for the construction account right? Sounds good!

phinze a few seconds ago
It is! Ok cool let's ship this.

Run confirmed

Comment: Leave feedback or record a decision.

[Add Comment](#)

Conclusion

There are many different ways to use Terraform: as an individual user, a single team, or an entire organization at scale. Choosing the best approach for the density of collaboration needed will provide the most return on your investment in the core Terraform workflow. For organizations using Terraform at scale, Terraform Cloud introduces new layers that build on this core workflow to solve problems unique to teams and organizations.