HashiCorp
(https://www.hashicorp.com)

Command: apply

JUMP TO SECTION

Hands-on: Try the Terraform: Get Started (https://learn.hashicorp.com/collections/terraform/aws-get-started?utm_source=WEBSITE&utm_medium=WEB_IO&utm_offer=ARTICLE_PAGE&utm_content=DOCS) collection on HashiCorp Learn.

The `terraform apply` command executes the actions proposed in a Terraform plan.

The most straightforward way to use `terraform apply` is to run it without any arguments at all, in which case it will automatically create a new execution plan (as if you had run `terraform plan`) and then prompt you to approve that plan, before taking the indicated actions.

Another way to use `terraform apply` is to pass it the filename of a saved plan file you created earlier with `terraform plan -out=...`, in which case Terraform will apply the changes in the plan without any confirmation prompt. This two-step workflow is primarily intended for when 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).

Usage

Usage: `terraform apply [options] [plan file]`

The behavior of `terraform apply` differs significantly depending on whether you pass it the filename of a previously-saved plan file.

Automatic Plan Mode

In the default case, with no saved plan file, `terraform apply` creates its own plan of action, in the same way that `terraform plan` (/docs/cli/commands/plan.html) would.

Terraform will propose the plan to you and prompt you to approve it before taking the described actions, unless you waive that prompt by using the `-auto-approve` option.

When performing its own plan, `terraform apply` supports all of the same planning modes (/docs/cli/commands/plan.html#planning-modes) and planning options (/docs/cli/commands/plan.html#planning-options) that `terraform plan` would accept, so you can customize how Terraform will create the plan.

Saved Plan Mode

If you pass the filename of a previously-saved plan file, `terraform apply` performs exactly the steps specified by that plan file. It does not prompt for approval; if you want to inspect a plan file before applying it, you can use `terraform show` (/docs/cli/commands/show.html).

When using a saved plan, none of the planning modes or planning options linked above are supported; these options only affect Terraform's decisions about which actions to take, and the plan file contains the final results of those decisions.

Plan Options

When run without a saved plan file, `terraform apply` supports all of `terraform plan`'s planning modes and planning options. For details, see:

- Planning Modes (/docs/cli/commands/plan.html#planning-modes)
- Planning Options (/docs/cli/commands/plan.html#planning-options)

Apply Options

The following options allow you to change various details about how the apply command executes and reports on the apply operation. If you are running `terraform apply` *without* a previously-saved plan file, these options are *in addition to* the planning modes and planning options described for `terraform plan` (/docs/cli/commands/plan.html).

- `-auto-approve` - Skips interactive approval of plan before applying. This option is ignored when you pass a previously-saved plan file, because Terraform considers you passing the plan file as the approval and so will never prompt in that case.
- `-compact-warnings` - Shows any warning messages in a compact form which includes only the summary messages, unless the warnings are accompanied by at least one error and thus the warning text might be useful context for the errors.
- `-input=false` - Disables all of Terraform's interactive prompts. Note that this also prevents Terraform from prompting for interactive approval of a plan, so Terraform will conservatively assume that you do not wish to apply the plan, causing the operation to fail. If you wish to run Terraform in a non-interactive context, see 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) for some different approaches.
- `-json` - Enables the machine readable JSON UI (/docs/internals/machine-readable-ui.html) output. This implies `-input=false`, so the configuration must have no unassigned variable values to continue. To enable this flag, you must also either enable the `-auto-approve` flag or specify a previously-saved plan.
- `-lock=false` - Don't hold a state lock during the operation. This is dangerous if others might concurrently run commands against the same workspace.
- `-lock-timeout=DURATION` - Unless locking is disabled with `-lock=false`, instructs Terraform to retry acquiring a lock for a period of time before returning an error. The duration syntax is a number followed by a time unit letter, such as "3s" for three seconds.
- `-no-color` - Disables terminal formatting sequences in the output. Use this if you are running Terraform in a context where its output will be rendered by a system that cannot interpret terminal formatting.
- `-parallelism=n` - Limit the number of concurrent operation as Terraform walks the graph (/docs/internals/graph.html#walking-the-graph). Defaults to 10.

For configurations using the `local` backend (/docs/language/settings/backends/local.html) only, `terraform apply` also accepts the legacy options `-state`, `-state-out`, and `-backup` (/docs/language/settings/backends/local.html#command-line-arguments).

Passing a Different Configuration Directory

Terraform v0.13 and earlier also accepted a directory path in place of the plan file argument to `terraform apply`, in which case Terraform would use that directory as the root module instead of the current working directory.

That usage was deprecated in Terraform v0.14 and removed in Terraform v0.15. If your workflow relies on overriding the root module directory, use the `-chdir` global option (./#switching-working-directory-with-chdir) instead, which works across all commands and makes Terraform consistently look in the given directory for all files it would normaly read or write in the current working directory.

If your previous use of this legacy pattern was also relying on Terraform writing the `.terraform` subdirectory into the current working directory even though the root module directory was overridden, use the `TF_DATA_DIR` environment variable (/docs/cli/config/environment-variables.html#tf_data_dir) to direct Terraform to write the `.terraform` directory to a location other than the current working directory.