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

Terraform Settings

JUMP TO SECTION

The special `terraform` configuration block type is used to configure some behaviors of Terraform itself, such as requiring a minimum Terraform version to apply your configuration.

Terraform Block Syntax

Terraform settings are gathered together into `terraform` blocks:

```
terraform {
  # ...
}
```

Each `terraform` block can contain a number of settings related to Terraform's behavior. Within a `terraform` block, only constant values can be used; arguments may not refer to named objects such as resources, input variables, etc, and may not use any of the Terraform language built-in functions.

The various options supported within a `terraform` block are described in the following sections.

Configuring a Terraform Backend

The nested `backend` block configures which backend Terraform should use.

The syntax and behavior of the `backend` block is described in Backend Configuration (/docs/language/settings/backends/configuration.html).

Specifying a Required Terraform Version

The `required_version` setting accepts a version constraint string, (/docs/language/expressions/version-constraints.html) which specifies which versions of Terraform can be used with your configuration.

If the running version of Terraform doesn't match the constraints specified, Terraform will produce an error and exit without taking any further actions.

When you use child modules (/docs/language/modules/index.html), each module can specify its own version requirements. The requirements of all modules in the tree must be satisfied.

Use Terraform version constraints in a collaborative environment to ensure that everyone is using a specific Terraform version, or using at least a minimum Terraform version that has behavior expected by the configuration.

The `required_version` setting applies only to the version of Terraform CLI. Terraform's resource types are implemented by provider plugins, whose release cycles are independent of Terraform CLI and of each other. Use the `required_providers` block (/docs/language/providers/requirements.html) to manage the expected versions for each provider you use.

Specifying Provider Requirements

The `required_providers` block specifies all of the providers required by the current module, mapping each local provider name to a source address and a version constraint.

```
terraform {
  required_providers {
    aws = {
      version = ">= 2.7.0"
      source = "hashicorp/aws"
    }
  }
}
```

For more information, see Provider Requirements (/docs/language/providers/requirements.html).

Experimental Language Features

The Terraform team will sometimes introduce new language features initially via an opt-in experiment, so that the community can try the new feature and give feedback on it prior to it becoming a backward-compatibility constraint.

In releases where experimental features are available, you can enable them on a per-module basis by setting the `experiments` argument inside a `terraform` block:

```
terraform {
  experiments = [example]
}
```

The above would opt in to an experiment named `example`, assuming such an experiment were available in the current Terraform version.

Experiments are subject to arbitrary changes in later releases and, depending on the outcome of the experiment, may change drastically before final release or may not be released in stable form at all. Such breaking changes may appear even in minor and patch releases. We do not recommend using experimental features in Terraform modules intended for production use.

In order to make that explicit and to avoid module callers inadvertently depending on an experimental feature, any module with experiments enabled will generate a warning on every `terraform plan` or `terraform apply`. If you want to try experimental features in a shared module, we recommend enabling the experiment only in alpha or beta releases of the module.

The introduction and completion of experiments is reported in Terraform's changelog (https://github.com/hashicorp/terraform/blob/main/CHANGELOG.md), so you can watch the release notes there to discover which experiment keywords, if any, are available in a particular Terraform release.

Passing Metadata to Providers

The `terraform` block can have a nested `provider_meta` block for each provider a module is using, if the provider defines a schema for it. This allows the provider to receive module-specific information, and is primarily intended for modules distributed by the same vendor as the associated provider.

For more information, see Provider Metadata (/docs/internals/provider-meta.html).