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

Provider Configuration

JUMP TO SECTION

Terraform relies on plugins called "providers" to interact with remote systems.

Terraform configurations must declare which providers they require, so that Terraform can install and use them. Additionally, some providers require configuration (like endpoint URLs or cloud regions) before they can be used.

- This page documents how to configure settings for providers.
- The Provider Requirements (/docs/language/providers/requirements.html) page documents how to declare providers so Terraform can install them.

Provider Configuration

Provider configurations belong in the root module of a Terraform configuration. (Child modules receive their provider configurations from the root module; for more information, see The Module `providers` Meta-Argument (/docs/language/meta-arguments/module-providers.html) and Module Development: Providers Within Modules (/docs/language/modules/develop/providers.html).)

A provider configuration is created using a `provider` block:

```
provider "google" {
  project = "acme-app"
  region  = "us-central1"
}
```

The name given in the block header (`"google"` in this example) is the local name (/docs/language/providers/requirements.html#local-names) of the provider to configure. This provider should already be included in a `required_providers` block.

The body of the block (between `{` and `}`) contains configuration arguments for the provider. Most arguments in this section are defined by the provider itself; in this example both `project` and `region` are specific to the `google` provider.

You can use expressions (/docs/language/expressions/index.html) in the values of these configuration arguments, but can only reference values that are known before the configuration is applied. This means you can safely reference input variables, but not attributes exported by resources (with an exception for resource arguments that are specified directly in the configuration).

A provider's documentation should list which configuration arguments it expects. For providers distributed on the Terraform Registry (https://registry.terraform.io), versioned documentation is available on each provider's page, via the "Documentation" link in the provider's header.

Some providers can use shell environment variables (or other alternate sources, like VM instance profiles) as values for some of their arguments; when available, we recommend using this as a way to keep credentials out of your version-controlled Terraform code.

There are also two "meta-arguments" that are defined by Terraform itself and available for all `provider` blocks:

- `alias`, for using the same provider with different configurations for different resources
- `version`, which we no longer recommend (use provider requirements (/docs/language/providers/requirements.html) instead)

Unlike many other objects in the Terraform language, a `provider` block may be omitted if its contents would otherwise be empty. Terraform assumes an empty default configuration for any provider that is not explicitly configured.

`alias`: Multiple Provider Configurations

You can optionally define multiple configurations for the same provider, and select which one to use on a per-resource or per-module basis. The primary reason for this is to support multiple regions for a cloud platform; other examples include targeting multiple

Docker hosts, multiple Consul hosts, etc.

To create multiple configurations for a given provider, include multiple `provider` blocks with the same provider name. For each additional non-default configuration, use the `alias` meta-argument to provide an extra name segment. For example:

```
# The default provider configuration; resources that begin with `aws_` will use
# it as the default, and it can be referenced as `aws`.
provider "aws" {
  region = "us-east-1"
}

# Additional provider configuration for west coast region; resources can
# reference this as `aws.west`.
provider "aws" {
  alias  = "west"
  region = "us-west-2"
}
```

To declare a configuration alias within a module in order to receive an alternate provider configuration from the parent module, add the `configuration_aliases` argument to that provider's `required_providers` entry. The following example declares both the `mycloud` and `mycloud.alternate` provider configuration names within the containing module:

```
terraform {
  required_providers {
    mycloud = {
      source  = "mycorp/mycloud"
      version = "~> 1.0"
      configuration_aliases = [ mycloud.alternate ]
    }
  }
}
```

Default Provider Configurations

A `provider` block without an `alias` argument is the *default* configuration for that provider. Resources that don't set the `provider` meta-argument will use the default provider configuration that matches the first word of the resource type name. (For

example, an `aws_instance` resource uses the default `aws` provider configuration unless otherwise stated.)

If every explicit configuration of a provider has an alias, Terraform uses the implied empty configuration as that provider's default configuration. (If the provider has any required configuration arguments, Terraform will raise an error when resources default to the empty configuration.)

Referring to Alternate Provider Configurations

When Terraform needs the name of a provider configuration, it expects a reference of the form `<PROVIDER NAME>.<ALIAS>`. In the example above, `aws.west` would refer to the provider with the `us-west-2` region.

These references are special expressions. Like references to other named entities (for example, `var.image_id`), they aren't strings and don't need to be quoted. But they are only valid in specific meta-arguments of `resource`, `data`, and `module` blocks, and can't be used in arbitrary expressions.

Selecting Alternate Provider Configurations

By default, resources use a default provider configuration (one without an `alias` argument) inferred from the first word of the resource type name.

To use an alternate provider configuration for a resource or data source, set its `provider` meta-argument to a `<PROVIDER NAME>.<ALIAS>` reference:

```
resource "aws_instance" "foo" {
  provider = aws.west

  # ...
}
```

To select alternate provider configurations for a child module, use its `providers` meta-argument to specify which provider configurations should be mapped to which local provider names inside the module:

```
module "aws_vpc" {
  source = "./aws_vpc"
  providers = {
    aws = aws.west
  }
}
```

Modules have some special requirements when passing in providers; see The Module `providers` Meta-Argument (/docs/language/meta-arguments/module-providers.html) for more details. In most cases, only *root modules* should define provider configurations, with all child modules obtaining their provider configurations from their parents.

`version` : An Older Way to Manage Provider Versions

The `version` meta-argument specifies a version constraint for a provider, and works the same way as the `version` argument in a `required_providers` block (/docs/language/providers/requirements.html). The version constraint in a provider configuration is only used if `required_providers` does not include one for that provider.

The `version` argument in provider configurations is deprecated. In Terraform 0.13 and later, version constraints should always be declared in the `required_providers` block (/docs/language/providers/requirements.html). The `version` argument will be removed in a future version of Terraform.

Note: The `version` meta-argument made sense before Terraform 0.13, since Terraform could only install providers that were distributed by HashiCorp. Now that Terraform can install providers from multiple sources, it makes more sense to keep version constraints and provider source addresses together.