

Lock and Upgrade Provider Versions

🕒 7 MIN

PRODUCTS USED  Terraform

This tutorial also appears in: [0.14 Release](#), [HashiConf Europe](#) and [Associate Tutorials](#).

Terraform providers manage resources by communicating between Terraform and target APIs. Whenever the target APIs change or add functionality, provider maintainers may update and version the provider.

When multiple users or automation tools run the same Terraform configuration, they should all use the same versions of their required providers. There are two ways for you to manage provider versions in your configuration.

- 1 Specify provider version constraints in your configuration's `terraform` block.
- 2 Use the [dependency lock file](#)

If you do not scope provider version appropriately, Terraform will download the latest provider version that fulfills the version constraint. This may lead to unexpected infrastructure changes. By specifying carefully scoped provider versions and using the dependency lock file, you can ensure Terraform is using the correct provider version so your configuration is applied consistently.

In this tutorial, you will create a S3 bucket from an initialized Terraform configuration. Then, you will ask the Terraform dependency lock file to use the latest version of the AWS provider, and edit the Terraform configuration to conform to the new provider version's requirements.

Prerequisites

This tutorial assumes you are familiar with the standard Terraform workflow. If you are unfamiliar with Terraform, complete the [Get Started tutorials](#) first.

For this tutorial, you will need:

- [Terraform 0.14+ installed locally](#)
- an [AWS account](#)

Explore the workspace

Clone the [Learn Terraform Provider Versioning](#) repository.

```
$ git clone https://github.com/hashicorp/learn-terraform
```

 Copy 

Navigate to the repository directory in your terminal.

```
$ cd learn-terraform-provider-versioning
```

 Copy 

This directory is a pre-initialized Terraform workspace with three files: `main.tf`, `versions.tf`, and `.terraform.lock.hcl`. HashiCorp has released a newer version of the AWS provider since this workspace was first initialized.

Explore `main.tf`

Open the `main.tf` file. This file uses the AWS and random providers to deploy a randomly named S3 bucket to the `us-west-2` region.

```
provider "aws" {  
  region = "us-west-2"  
}  
  
resource "random_pet" "petname" {  
  length      = 5  
  separator   = "-"  
}  
  
resource "aws_s3_bucket" "sample" {  
  bucket = random_pet.petname.id  
  acl    = "public-read"  
  
  region = "us-west-2"  
}
```

Explore `versions.tf`

Open the `versions.tf` file. Here you will find the `terraform` block which specifies the required provider version and required Terraform version for this configuration.

```
terraform {  
  required_providers {  
    random = {  
      source = "hashicorp/random"  
      version = "3.0.0"  
    }  
  
    aws = {  
      source = "hashicorp/aws"  
      version = ">= 2.0.0"  
    }  
  }  
}
```

```

    }

    required_version = "~> 0.14"
  }

```

The `terraform` block contains the `required_providers` block which specifies the provider local name, the [source address](#) and the version.

When you initialize this configuration, Terraform will download:

- 1 Version 3.0.0 of the `random` provider.
- 2 The latest version of the `AWS` provider that is at greater than 2.0. The `>=` [version constraint operator](#) specifies the minimum provider version that's compatible with the configuration.

In addition, the `Terraform` block specifies only Terraform binaries that are v0.14.x can run this configuration. The `~>` [operator](#) is a convenient shorthand for allowing only patch releases within a specific minor release.

Explore `terraform.lock.hcl`

When you initialize a Terraform configuration for the first time with Terraform 0.14 or later, Terraform will generate a new `.terraform.lock.hcl` file in the current working directory. You should include the lock file in your version control repository to ensure that Terraform uses the same provider versions across your team and in ephemeral remote execution environments.

Open the `.terraform.lock.hcl` file.

```

# This file is maintained automatically by "terraform in
# Manual edits may be lost in future updates.

```

```

provider "registry.terraform.io/hashicorp/aws" {
  version      = "2.50.0"
  constraints = ">= 2.0.0"
  hashes = [

```

```

    "h1:aKw4NLrMEAf1s110XCCz6Ewo4ay9dpgSpkNHujRXX08=",
    ## ...
    "zh:fdeaf059f86d0ab59cf68ece2e8cec522b506c47e2cfca7ba6125b1cd0f
  ]
}

provider "registry.terraform.io/hashicorp/random" {
  version      = "3.0.0"
  constraints = "3.0.0"
  hashes = [
    "h1:yHJpb4IfQQfuio7qjUXuUFTU/s+ensuEpm23A+VWz0=",
    ## ...
    "zh:fbdd0684e62563d3ac33425b0ac9439d543a3942465f4b26582bcfabcb1
  ]
}

```

Notice the two providers specified in your `versions.tf` file. The AWS provider version is v2.50.0. This fulfills the `>=2.0.0` constraint, but is no longer the latest version of the AWS provider. The random provider is set to v3.0.0 and fulfills its version constraints.

Initialize and apply the configuration

In your terminal, re-initialize your Terraform workspace.

```
$ terraform init
```

Copy 

```
Initializing the backend...
```

```
Initializing provider plugins...
```

- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Installing hashicorp/aws v2.50.0...
- Installed hashicorp/aws v2.50.0 (signed by HashiCorp)
- Installing hashicorp/random v3.0.0...
- Installed hashicorp/random v3.0.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform operations should now work.

...



Notice that instead of installing the latest version of the AWS provider that conforms with the configured version constraints, Terraform installed the version specified in the lock file. While initializing your workspace, Terraform read the dependency lock file and downloaded the specified versions of the AWS and random providers.

If Terraform did not find a lock file, it would download the latest versions of the providers that fulfill the version constraints you defined in the `required_providers` block. The following table shows which provider Terraform would download in this scenario, based on the version constraint and presence of a lock file.

Provider	Version Constraint	terraform init (no lock file)	terraform init (lock file)
aws	>= 2.0.0	Latest version (e.g. 3.24.1)	Lock file version (2.50.0)
random	3.0.0	3.0.0	Lock file version (3.0.0)

The lock file causes Terraform to always install the same provider version, ensuring that runs across your team or remote sessions will be consistent.

Apply your configuration. Remember to confirm your apply with a `yes`.

\$ terraform apply

Copy

...

Plan: 2 to add, 0 to change, 0 to destroy.

...

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Upgrade the AWS provider version

The `-upgrade` flag will upgrade all providers to the latest version consistent within the version constraints previously established in your configuration.

Upgrade the AWS provider.

Note: You should never directly modify the lock file.

```
$ terraform init -upgrade
```

Copy 

```
Initializing the backend...
```

```
Initializing provider plugins...
```

- Finding hashicorp/random versions matching "3.0.0"...
- Finding hashicorp/aws versions matching ">= 2.0.0"...
- Installing hashicorp/random v3.0.0...
- Installed hashicorp/random v3.0.0 (signed by HashiCorp)
- Installing hashicorp/aws v3.24.1...
- Installed hashicorp/aws v3.24.1 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selectio in the `.terraform.lock.hcl` file. Review those changes and commit th version control system if they represent changes you intended to ma

```
Terraform has been successfully initialized!
```

...

Notice that Terraform installs the latest version of the AWS provider.

Open the `.terraform.lock.hcl` file and notice that the AWS provider's version is now the latest version (3.18.0 or later).

```
provider "registry.terraform.io/hashicorp/aws" {  
  version      = "3.24.1"  
  constraints = ">= 2.0.0"  
  ## ...  
}
```

Copy 

Tip: You can also use the `-upgrade` flag to downgrade the provider versions if the version constraints are modified to specify a lower provider version.

Apply your configuration with the new provider version installed to see an example of why you would want to lock the provider version. The apply step will fail because the `aws_s3_bucket` resource's `region` attribute is [read only for AWS providers v3.0.0+](#).

```
$ terraform apply
```

Copy 

```
Error: Computed attribute cannot be set
```

```
on main.tf line 14, in resource "aws_s3_bucket" "sample":  
14:   region = "us-west-2"
```

Remove the `region` attribute from the `aws_s3_bucket.sample` resource.

```
resource "aws_s3_bucket" "sample" {  
  bucket = random_pet.petname.id
```



```
acl      = "public-read"  
-   region = "us-west-2"  
}
```

Apply your configuration. There will be no errors now.

```
$ terraform apply  
random_pet.petname: Refreshing state... [id=cheaply-jolly-apparentl  
aws_s3_bucket.sample: Refreshing state... [id=cheaply-jolly-apparer
```

Copy 

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



If the apply step completes successfully, it is safe to commit the configuration with the updated lock file to version control. If the plan or apply steps fail, do **not** commit the lock file to version control.

Clean up workspace

After verifying that the resources were deployed successfully, destroy them. Remember to respond to the confirmation prompt with `yes`.

```
$ terraform destroy
```

Copy 

Next steps

In this tutorial, you used the dependency lock file to manage provider versions, and upgraded the lock file.

To learn more about providers, visit the following resources.

- [Dependency lock file documentation](#)
- [Provider Version Constraint documentation](#)

- [Call APIs with Terraform Providers](#) tutorials walk you through how providers serve as a bridge between Terraform and target APIs, and show you how to build a custom Terraform provider.

Was this tutorial helpful?

[← BACK TO COLLECTION](#)[System Status](#)[Cookie Manager](#)[Terms of Use](#)[Security](#)[Privacy](#)

stdin: is not a tty