Lock and Upgrade Provider Versions

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 update 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

You can complete this tutorial using the same workflow with either Terraform OSS or Terraform Cloud. Terraform Cloud is a platform that you can use to manage and execute your Terraform projects. It includes features like remote state and execution, structured plan output, workspace resource summaries, and more.

Select the **Terraform Cloud** tab to complete this tutorial using Terraform Cloud.

This tutorial assumes that you are familiar with the Terraform workflow. If you are new to Terraform, complete the Get Started tutorials first.

In order to complete this tutorial, you will need the following:

- Terraform v1.1+ installed locally.
- An AWS account with local credentials configured for use with Terraform.

This tutorial assumes that you are familiar with the Terraform and Terraform Cloud workflows. If you are new to Terraform, complete the Get Started tutorials first. If you are new to Terraform Cloud, complete the Terraform Cloud Get Started tutorials first.

In order to complete this tutorial, you will need the following:

- Terraform v1.1+ installed locally.
- An AWS account.
- A Terraform Cloud account with Terraform Cloud locally authenticated.
- A Terraform Cloud variable set configured with your AWS credentials.

Clone example repository

Clone the Learn Terraform Provider Versioning repository.

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

Navigate to the repository directory in your terminal.

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

Review configuration

This directory is a pre-initialized Terraform project with three files: `main.tf`, `terraform.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.


main.tf

```
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"

  tags = {
    public_bucket = true
  }
}
```

Explore `terraform.tf`

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


terraform.tf

```
terraform {
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }

    aws = {
      source  = "hashicorp/aws"
      version = ">= 2.0.0"
    }
  }

  required_version = ">= 1.1"
}
```

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 3.1.0 of the random provider.
2. The latest version of the AWS provider that is at greater than 2.0.0. The >= version constraint operator specifies the minimum provider version that is compatible with the configuration.

The Terraform block also specifies that only Terraform binaries newer than v1.1.x can run this configuration by using the `>=` operator as well.

Explore `terraform.lock.hcl`

When you initialize a Terraform configuration for the first time with Terraform 1.1 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.


.terraform.lock.hcl

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

provider "registry.terraform.io/hashicorp/aws" {
  version     = "2.50.0"
  constraints = ">= 2.0.0"
  hashes = [
    "h1:aKw4NLrMEAflsl1OXCCz6Ewo4ay9dpgSpkNHujRXXO8=",
    ## ...
    "zh:fdeaf059f86d0ab59cf68ece2e8cec522b506c47e2cfca7ba6125b1cd06b8680",
  ]
}

provider "registry.terraform.io/hashicorp/random" {
  version     = "3.1.0"
  constraints = "3.1.0"
  hashes = [
    "h1:9cCiLO/Cqr6IUvMDSApCkQItooiYNatZpEXmcu0nnng=",
    ## ...
    "zh:f7605bd1437752114baf601bdf6931debe6dc6bfe3006eb7e9bb9080931dca8a",
  ]
}
```

Notice the two providers specified in your `terraform.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 v3.1.0 and fulfills its version constraints.

Initialize and apply the configuration

Note: If you are using an Apple M1 or M2 CPU, you can not initialize or apply the starting configuration because the AWS provider version is too old for those processors. Read this section and follow the others, and the final configuration will work as expected.

Initialize this configuration.

```
$ terraform init

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/random v3.1.0...
- Installed hashicorp/random v3.1.0 (signed by HashiCorp)
- Installing hashicorp/aws v2.50.0...
- Installed hashicorp/aws v2.50.0 (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 commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Open your `terraform.tf` file and uncomment the `cloud` block. Replace the `organization` name with your own Terraform Cloud organization.

terraform.tf

```
terraform {
  cloud {
    organization = "organization-name"
    workspaces {
      name = "learn-terraform-provider-versioning"
    }
  }
  required_providers {
    random = {
      source  = "hashicorp/random"
      version = "3.0.0"
    }

    aws = {
      source  = "hashicorp/aws"
      version = ">= 2.0.0"
    }
  }

  required_version = ">= 1.1"
}
```

Initialize your configuration. Terraform will automatically create the `learn-terraform-provider-versioning` workspace in your Terraform Cloud organization.

```
$ terraform init

Initializing Terraform Cloud...

Initializing provider plugins...
- Reusing previous version of hashicorp/random from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/random v3.1.0...
- Installed hashicorp/random v3.1.0 (signed by HashiCorp)
- Installing hashicorp/aws v2.50.0...
- Installed hashicorp/aws v2.50.0 (signed by HashiCorp)

Terraform Cloud has been successfully initialized!

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

If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.
```

Note: This tutorial assumes that you are using a tutorial-specific Terraform Cloud organization with a global variable set of your AWS credentials. Review the Create a Credential Variable Set for detailed guidance. If you are using a scoped variable set, assign it to your new workspace now.

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. 4.45.0)	Lock file version (2.50.0)
random	3.1.0	3.1.0	Lock file version (3.1.0)

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

Apply your configuration. Respond to the confirmation prompt with a `yes` to create the example infrastructure.

```
$ terraform apply

## ...

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 specified in your configuration.

Upgrade the AWS provider.

Note: You should never directly modify the lock file.

```
$ terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching ">= 2.0.0"...
- Finding hashicorp/random versions matching "3.1.0"...
- Installing hashicorp/aws v4.45.0...
- Installed hashicorp/aws v4.45.0 (signed by HashiCorp)
- Using previously-installed hashicorp/random v3.1.0

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

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.

.terraform.lock.hcl

```
provider "registry.terraform.io/hashicorp/aws" {
  version     = "4.45.0"
  constraints = ">= 2.0.0"
  ## ...
}
```

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 observe the potential side effects of not locking 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+. In addition, the `acl` attribute is deprecated for AWS providers version v4.0.0+.

```
$ terraform apply
╷
│ Warning: Argument is deprecated
│
│   with aws_s3_bucket.sample,
│   on main.tf line 12, in resource "aws_s3_bucket" "sample":
│   12:   acl    = "public-read"
│
│ Use the aws_s3_bucket_acl resource instead
╵

╷
│ Error: Value for unconfigurable attribute
│
│   with aws_s3_bucket.sample,
│   on main.tf line 14, in resource "aws_s3_bucket" "sample":
│   14:   region = "us-west-2"
│
│ Can't configure a value for "region": its value will be decided automatically
based on the result of applying this configuration.
```

Remove the `acl` and `region` attributes from the `aws_s3_bucket.sample` resource.

main.tf

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

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

  tags = {
    public_bucket = true
  }
}
```

Then, add the following resource to set ACLs for your bucket.

main.tf

```
resource "aws_s3_bucket_acl" "example" {
  bucket = aws_s3_bucket.sample.id
  acl    = "public-read"
}
```

Apply your configuration. Respond to the confirmation prompt with a `yes`.

```
$ terraform apply
random_pet.petname: Refreshing state... [id=cheaply-jolly-apparently-hopeful-dane]
aws_s3_bucket.sample: Refreshing state... [id=cheaply-jolly-apparently-hopeful-
dane]

## ...

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

## ...

aws_s3_bucket_acl.example: Creating...
aws_s3_bucket_acl.example: Creation complete after 1s [id=cheaply-jolly-
apparently-hopeful-dane,public-read]

Apply complete! Resources: 1 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 infrastructure

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

```
$ terraform destroy
## ...
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  - destroy

## ...

Plan: 0 to add, 0 to change, 3 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

## ...

Destroy complete! Resources: 3 destroyed.
```

If you used Terraform Cloud for this tutorial, after destroying your resources, delete the `learn-terraform-provider-versioning` workspace from your Terraform Cloud organization.

Next steps

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