Terraform Module Versions

Modules, like any piece of code, are never complete. There will always be new module requirements and changes.

Each distinct module address has associated with it a set of versions, each of which has an associated version number. Terraform assumes version numbers follow the Semantic Versioning 2.0 convention. Each module block may select a distinct version of a module, even if multiple blocks have the same source address.

- Task 1: Viewing Terraform Module Versions
- Task 2: Comparing Terraform Module Versions
- Task 3: Terraform Module Version Constraints

Task 1: Viewing Terraform Module Versions

Each module in the Terraform Public registry is versioned. These versions syntactically must follow semantic versioning. The version argument can be specified as part of the module block as is shown in our example for the AWS VPC Terraform module.

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.11.0"
}
```

When installing a module inside the working directory with a `terraform init` Terraform pulls down version of the module specified by the version argument.

```
terraform init

Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/vpc/aws 3.11.0 for
    vpc...
- vpc in .terraform/modules/vpc
```

Task 2: Comparing Terraform Module Versions

Update the vpc module block to specify a particular module version, by adding the `version` argument. You can find the latest version of the module by viewing the module information in the Terraform Public Module Registry

VPC Module

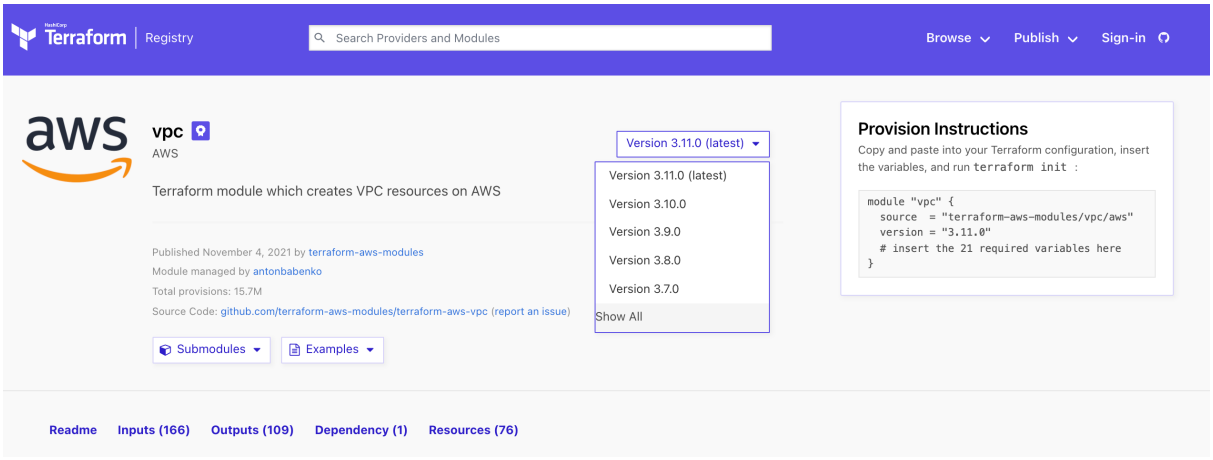

Figure 1: VPC Module Version

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = "3.11.0"

  name = "my-vpc-terraform"
  cidr = "10.0.0.0/16"

  azs             = ["us-east-1a", "us-east-1b", "us-east-1c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = true
  enable_vpn_gateway = true

  tags = {
    Name        = "VPC from Module"
    Terraform   = "true"
    Environment = "dev"
  }
}
```

Execute a `terraform init` followed by a `terraform plan` to validate that specifying the module version did not result in a change to the vpc.

Now we will change to an older version of the vpc module. Update the VPC module block to utilize version `1.73.0`

```
module "vpc" {
```


```
  source = "terraform-aws-modules/vpc/aws"
  version = "1.73.0"

  name = "my-vpc-terraform"
  cidr = "10.0.0.0/16"

  azs             = ["us-east-1a", "us-east-1b", "us-east-1c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = true
  enable_vpn_gateway = true

  tags = {
    Name        = "VPC from Module"
    Terraform   = "true"
    Environment = "dev"
  }
}
```

After changing the module version run a `terraform init` to install the `1.73.0` version of the module.

```
terraform init
Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/vpc/aws 1.73.0 for
    vpc...
- vpc in .terraform/modules/vpc

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/local from the dependency lock
   file
- Reusing previous version of hashicorp/tls from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/http from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock
   file
- Using previously-installed hashicorp/random v3.1.0
- Using previously-installed hashicorp/local v2.1.0
- Using previously-installed hashicorp/tls v3.1.0
- Using previously-installed hashicorp/aws v3.70.0
- Using previously-installed hashicorp/http v2.1.0

| Warning: Quoted references are deprecated
|
|   on .terraform/modules/vpc/main.tf line 110, in resource "
```

```
   aws_route_table" "private":
|  110:     ignore_changes = ["propagating_vgws"]
|
| In this context, references are expected literally rather than in quotes
   . Terraform
| 0.11 and earlier required quotes, but quoted references are now
   deprecated and will
| be removed in a future version of Terraform. Remove the quotes
   surrounding this
| reference to silence this warning.
|
| (and 2 more similar warnings elsewhere)
```

Notice that during the installation of this module version we recieved warnings that there are depcreated commands that this version of the module uses. Remember that Terraform modules are simply terraform configuration files and this version of the module is using terraform configuration that has been depcreated. This highlights why it is important to be sure we specify a version of a module that works within the Terraform core version we are using.

Run a `terraform validate` to showcase other errors surfaced with this older version of the module.

```
terraform validate

| Warning: Quoted references are deprecated
|
|   on .terraform/modules/vpc/main.tf line 110, in resource "
   aws_route_table" "private":
|  110:     ignore_changes = ["propagating_vgws"]
|
| In this context, references are expected literally rather than in quotes
   . Terraform 0.11 and
| earlier required quotes, but quoted references are now deprecated and
   will be removed in a future
| version of Terraform. Remove the quotes surrounding this reference to
   silence this warning.
|
| (and 2 more similar warnings elsewhere)

| Error: Error in function call
|
|   on .terraform/modules/vpc/main.tf line 26, in resource "aws_vpc" "this
   ":
|   26:   tags = "${merge(map("Name", format("%s", var.name)), var.tags,
   var.vpc_tags)}"
|     |----------------
|     | var.name will be known only after apply
```


```
|
| Call to function "map" failed: the "map" function was deprecated in
   Terraform v0.12 and is no
| longer available; use tomap({ ... }) syntax to write a literal map.

| Error: Error in function call
|
|   on .terraform/modules/vpc/main.tf line 49, in resource "
   aws_vpc_dhcp_options" "this":
|   49:   tags = "${merge(map("Name", format("%s", var.name)), var.tags,
   var.dhcp_options_tags)}"
|     |----------------
|     | var.name will be known only after apply
|
| Call to function "map" failed: the "map" function was deprecated in
   Terraform v0.12 and is no
| longer available; use tomap({ ... }) syntax to write a literal map.

| Error: Error in function call
|
|   on .terraform/modules/vpc/main.tf line 590, in resource "aws_eip" "nat
   ":
|  590:   tags = "${merge(map("Name", format("%s-%s", var.name, element(
   var.azs, (var.single_nat_gateway ? 0 : count.index)))), var.tags, var.
   nat_eip_tags)}"
|     |----------------
|     | count.index is a number, known only after apply
|     | var.azs will be known only after apply
|     | var.name will be known only after apply
|     | var.single_nat_gateway will be known only after apply
|
| Call to function "map" failed: the "map" function was deprecated in
   Terraform v0.12 and is no
| longer available; use tomap({ ... }) syntax to write a literal map.

| Error: Error in function call
|
|   on .terraform/modules/vpc/main.tf line 711, in resource "
   aws_default_vpc" "this":
|  711:   tags = "${merge(map("Name", format("%s", var.default_vpc_name)),
    var.tags, var.default_vpc_tags)}"
|     |----------------
|     | var.default_vpc_name will be known only after apply
|
| Call to function "map" failed: the "map" function was deprecated in
   Terraform v0.12 and is no
| longer available; use tomap({ ... }) syntax to write a literal map.
```

It should now be obvious that this version of the VPC module is not compatiable with our current

version of Terraform. Let's look at how we can incorporate module verison constraints to make sure our code is using a compatible version of our module.

Task 3: Terraform Module Version Constraints

When using modules installed from a module registry it is highly recommended to explicitly constrain the acceptable version numbers to avoid unexpected or unwanted changes. Terraform provides the ability to resolve any provided module version constraints and using them is highly recommended to avoid pulling in breaking changes. The version argument accepts a version constraint string. Terraform will use the newest installed version of the module that meets the constraint; if no acceptable versions are installed, it will download the newest version that meets the constraint.

Update the VPC module block to utilze any version greater then 3.0.0

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = ">3.0.0"

  name = "my-vpc-terraform"
  cidr = "10.0.0.0/16"

  azs             = ["us-east-1a", "us-east-1b", "us-east-1c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = true
  enable_vpn_gateway = true

  tags = {
    Name        = "VPC from Module"
    Terraform   = "true"
    Environment = "dev"
  }
}
```

Run a `terraform init` to validate that the installed module meets the version constraints.

```
terraform init

Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/vpc/aws 3.11.0 for
    vpc...
- vpc in .terraform/modules/vpc
```

Version constraints are supported only for modules installed from a module registry, such as the public

Terraform Registry or Terraform Cloud's private module registry. Other module sources can provide their own versioning mechanisms within the source string itself, or might not support versions at all. In particular, modules sourced from local file paths do not support version; since they're loaded from the same source repository, they always share the same version as their caller.

Now that we have shown how to use modules from the Public Module Registry and constrain to appropriate versions we can cleanup from this section of the course by issuing a `terraform destroy` and removing the module references from the within our `main.tf`