

Terraform is the infrastructure as code offering from HashiCorp. It is a tool for building, changing, and managing infrastructure in a safe, repeatable way. Operators and Infrastructure teams can use Terraform to manage environments with a configuration language called the HashiCorp Configuration Language (HCL) for human-readable, automated deployments.

Infrastructure as code is the process of managing infrastructure in a file or files rather than manually configuring resources in a user interface. A resource in this instance is any piece of infrastructure in a given environment, such as a virtual machine, security group, network interface, etc. At a high level, Terraform allows operators to use HCL to author files containing definitions of their desired resources on almost any provider (AWS, Google Cloud, GitHub, Docker, etc.) and automates the creation of those resources at the time of apply.

A simple workflow for deployment will follow closely to the steps below:

- **Scope** - Confirm what resources need to be created for a given project.
- **Author** - Create the configuration file in HCL based on the scoped parameters.
- **Initialize** - Run `terraform init` in the project directory with the configuration files. This will download the correct provider plug-ins for the project.
- **Plan & Apply** - Run `terraform plan` to verify creation process and then `terraform apply` to create real resources as well as the state file that compares future changes in your configuration files to what actually exists in your deployment environment.

In this lab, you will learn how to perform the following tasks:

- Build, change, and destroy infrastructure with Terraform
- Create Resource Dependencies with Terraform
- Provision infrastructure with Terraform

Task 1. Build infrastructure

Terraform comes pre-installed in Cloud Shell. With Terraform already installed, you can dive right in and create some infrastructure.

Start by creating your example configuration to a file named `main.tf`. Terraform recognizes files ending in `.tf` or `.tf.json` as configuration files and will load them when it runs.

1. Create the `main.tf` file:

```
touch main.tf
```

2. In the Editor, add the following content to the `main.tf` file. Make sure to replace `<PROJECT_ID>` with the lab's Project ID:

```
terraform {  
  required_providers {  
    google = {  
      source = "hashicorp/google"  
    }  
  }  
}  
  
provider "google" {  
  version = "3.5.0"  
  project = "<PROJECT_ID>"  
  region  = "us-central1"  
  zone    = "us-central1-c"  
}  
  
resource "google_compute_network" "vpc_network" {  
  name = "terraform-network"  
}
```

```
student_00_4003902989c4@cloudshell:~$ touch main.tf
student_00_4003902989c4@cloudshell:~$ ls
main.tf  README-cloudshell.txt
student_00_4003902989c4@cloudshell:~$ vi main.tf
student_00_4003902989c4@cloudshell:~$ cat main.tf
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
    }
  }
}
provider "google" {
  version = "3.5.0"
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"
  region  = "us-central1"
  zone    = "us-central1-c"
}
resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}
```

Terraform block

The `terraform {}` block is required so Terraform knows which provider to download from the [Terraform Registry](#). In the configuration above, the `google` provider's source is defined as `hashicorp/google` which is shorthand for `registry.terraform.io/hashicorp/google`.

You can also assign a version to each provider defined in the `required_providers` block. The `version` argument is optional, but recommended. It is used to constrain the provider to a specific version or a range of versions in order to prevent downloading a new provider that may possibly contain breaking changes. If the version isn't specified, Terraform will automatically download the most recent provider during initialization.

To learn more, on the HashiCorp Terraform website, see [Provider Requirements](#).

Providers

The `provider` block is used to configure the named provider, in this case `google`. A provider is responsible for creating and managing resources. Multiple provider blocks can exist if a Terraform configuration manages resources from different providers.

Initialization

The first command to run for a new configuration – or after checking out an existing configuration from version control – is `terraform init`, which initializes various local settings and data that will be used by subsequent commands.

- Initialize your new Terraform configuration by running the `terraform init` command in the same directory as your `main.tf` file:

`terraform init`

```
student_00_4003902989c4@cloudshell:~$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/google versions matching "3.5.0"...
- Installing hashicorp/google v3.5.0...
- Installed hashicorp/google v3.5.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Warning: Version constraints inside provider configuration blocks are deprecated
on main.tf line 9, in provider "google":
9:   version = "3.5.0"

Terraform 0.13 and earlier allowed provider version constraints inside the provider
configuration blocks. This warning was introduced in Terraform 0.12. To silence this warning, move the provider version constraints
to the required_providers block in the terraform block at the top of your configuration.

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.
student_00_4003902989c4@cloudshell:~$
```

Creating resources

1. Apply your configuration now by running the command `terraform apply`:

`terraform apply`

```

student_00_4003902989c4@cloudshell:~$ terraform apply

Terraform used the selected providers to generate the following execution plan:
+ create

Terraform will perform the following actions:

# google_compute_network.vpc_network will be created
+ resource "google_compute_network" "vpc_network" {
  + auto_create_subnetworks      = true
  + delete_default_routes_on_create = false
  + gateway_ipv4                 = (known after apply)
  + id                           = (known after apply)
  + ipv4_range                   = (known after apply)
  + name                         = "terraform-network"
  + project                      = (known after apply)
  + routing_mode                  = (known after apply)
  + self_link                     = (known after apply)
}

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

```

```

Enter a value: yes

google_compute_network.vpc_network: Creating...
google_compute_network.vpc_network: Still creating... [10s elapsed]
google_compute_network.vpc_network: Still creating... [20s elapsed]
google_compute_network.vpc_network: Still creating... [30s elapsed]
google_compute_network.vpc_network: Creation complete after 30s [id=project-4003902989c4]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
student_00_4003902989c4@cloudshell:~$

```

The output has a + next to resource "google_compute_network" "vpc_network", meaning that Terraform will create this resource. Beneath that, it shows the attributes that will be set. When the value displayed is (known after apply), it means that the value won't be known until the resource is created.

If the plan was created successfully, Terraform will now pause and wait for approval before proceeding. If anything in the plan seems incorrect or dangerous, it is safe to abort here with no changes made to your infrastructure.

If terraform apply failed with an error, read the error message and fix the error that occurred.

2. The plan looks acceptable here, so type yes at the confirmation prompt to proceed.

Executing the plan will take a few minutes since Terraform waits for the network to be created successfully:

After this, Terraform is all done! You can go to the Cloud Console to see the network you have provisioned.

3. In the Console, from the **Navigation menu**, navigate to **VPC network**. You will see the terraform-network has been provisioned.

VPC network

VPC networks [+ CREATE](#)

VPC networks

NETWORKS IN CURRENT PROJECT

Use Network Inte

- ✓ Visualize you
- ✓ Diagnose and
- ✓ View packet
- ✓ Keep your fir

[GO TO NETWORK](#)

SMTP port 25 disallowed in this project

VPC networks

Filter Enter property name or value

Name ↑	Subnets
default	6
terraform-network	6

4. In Cloud Shell run the terraform show command to inspect the current state:

terraform show

```
student_00_4003902989c4@cloudshell:~$ terraform show
# google_compute_network.vpc_network:
resource "google_compute_network" "vpc_network" {
  auto_create_subnetworks = true
  delete_default_routes_on_create = false
  id = "projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network"
  name = "terraform-network"
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"
  routing_mode = "REGIONAL"
  self_link = "https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network"
}
student_00_4003902989c4@cloudshell:~$
```

Task 2. Change infrastructure

In the previous section, you created basic infrastructure with Terraform: a VPC network. In this section, you're going to modify your configuration, and see how Terraform handles change.

Infrastructure is continuously evolving, and Terraform was built to help manage and enact that change. As you change Terraform configurations, Terraform builds an execution plan that only modifies what is necessary to reach your desired state.

By using Terraform to change infrastructure, you can version control not only your configurations but also your state so you can see how the infrastructure evolves over time.

Adding resources

You can add new resources by adding them to your Terraform configuration and running `terraform apply` to provision them.

1. In the Editor, add a compute instance resource to `main.tf`:

```
resource "google_compute_instance" "vm_instance" {  
  name      = "terraform-instance"  
  machine_type = "f1-micro"  
  boot_disk {  
    initialize_params {  
      image = "debian-cloud/debian-11"  
    }  
  }  
  network_interface {  
    network = google_compute_network.vpc_network.name  
    access_config {  
    }  
  }  
}
```

```

student_00_4003902989c4@cloudshell:~$ cat main.tf
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
    }
  }
}

provider "google" {
  version = "3.5.0"
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"
  region  = "us-central1"
  zone    = "us-central1-c"
}

resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}

resource "google_compute_instance" "vm_instance" {
  name         = "terraform-instance"
  machine_type = "f1-micro"
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }
  network_interface {
    network = google_compute_network.vpc_network.name
    access_config {
    }
  }
}

student_00_4003902989c4@cloudshell:~$

```

This resource includes a few more arguments. The name and machine type are simple strings, but `boot_disk` and `network_interface` are more complex blocks. You can see all of the available options in the [google_compute_instance documentation](#).

For this example, your compute instance will use a Debian operating system, and will be connected to the VPC Network you created earlier. Notice how this configuration refers to the network's name property with `google_compute_network.vpc_network.name` -- `google_compute_network.vpc_network` is the ID, matching the values in the block that defines the network, and `name` is a property of that resource.

The presence of the `access_config` block, even without any arguments, ensures that the instance will be accessible over the internet.


```

+ initialize_params {
+   + image = "debian-cloud/debian-11"
+   + labels = (known after apply)
+   + size   = (known after apply)
+   + type   = (known after apply)
+ }
}

+ network_interface {
+   + name                = (known after apply)
+   + network              = "terraform-network"
+   + network_ip           = (known after apply)
+   + subnetwork            = (known after apply)
+   + subnetwork_project   = (known after apply)

+   + access_config {
+     + nat_ip         = (known after apply)
+     + network_tier   = (known after apply)
+   }
}

+ scheduling {
+   + automatic_restart   = (known after apply)
+   + on_host_maintenance = (known after apply)
+   + preemptible         = (known after apply)

+   + node_affinities {
+     + key       = (known after apply)
+     + operator  = (known after apply)
+     + values    = (known after apply)
+   }
}
}

```

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

2. Now run `terraform apply` to create the compute instance:

`terraform apply`

```

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

Warning: Version constraints inside provider configuration blocks are deprecated
on main.tf line 9, in provider "google":
  9:   version = "3.5.0"

Terraform 0.13 and earlier allowed provider version constraints inside the
configuration. To silence this warning, move the provider version constraint to the
version of Terraform.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

google_compute_instance.vm_instance: Creating...
google_compute_instance.vm_instance: Still creating... [10s elapsed]
google_compute_instance.vm_instance: Creation complete after 16s [id=project-1234567890]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```

3. Once again, answer yes to the confirmation prompt.

This is a fairly straightforward change - you added a "google_compute_instance" resource named "vm_instance" to your configuration, and Terraform created the resource in Google Cloud.

Changing resources

In addition to creating resources, Terraform can also make changes to those resources.

1. Add a tags argument to your "vm_instance" so that it looks like this:

```

resource "google_compute_instance" "vm_instance" {
  name          = "terraform-instance"
  machine_type  = "f1-micro"
  tags          = ["web", "dev"]
  # ...
}

```

```
student_00_4003902989c4@cloudshell:~$ cat main.tf
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
    }
  }
}

provider "google" {
  version = "3.5.0"
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"
  region  = "us-central1"
  zone    = "us-central1-c"
}

resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}

resource "google_compute_instance" "vm_instance" {
  name          = "terraform-instance"
  machine_type  = "f1-micro"
  tags          = ["web", "dev"]
  network_interface {
    network = google_compute_network.vpc_network.name
    access_config {
    }
  }
}
}
```

We removed boot disk from main.tf and bcz of that we got error in terraform plan command .

```
student_00_4003902989c4@cloudshell:~$ cat main.tf
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
    }
  }
}
provider "google" {
  version = "3.5.0"
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"
  region  = "us-central1"
  zone    = "us-central1-c"
}
resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}
resource "google_compute_instance" "vm_instance" {
  name         = "terraform-instance"
  machine_type = "f1-micro"
  tags         = ["web", "dev"]
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }
  network_interface {
    network = google_compute_network.vpc_network.name
    access_config {
    }
  }
}
}
```

```
student_00_4003902989c4@cloudshell:~$ terraform plan
```

Warning: Version constraints inside provider configuration blocks are deprecated

on main.tf line 9, in provider "google":
9: version = "3.5.0"

Terraform 0.13 and earlier allowed provider version constraints inside the p
version of Terraform. To silence this warning, move the provider version con

Error: Insufficient boot_disk blocks

on main.tf line 17, in resource "google_compute_instance" "vm_instance":
17: resource "google_compute_instance" "vm_instance" {

At least 1 "boot_disk" blocks are required.

```

student_00_4003902989c4@cloudshell:~$ terraform plan
google_compute_network.vpc_network: Refreshing state... [id=
google_compute_instance.vm_instance: Refreshing state... [id=

Terraform used the selected providers to generate the following
~ update in-place

Terraform will perform the following actions:

# google_compute_instance.vm_instance will be updated in-place
~ resource "google_compute_instance" "vm_instance" {
  id          = "projects/qwiklabs-gcp-03-5b..."
  name        = "terraform-instance"
  ~ tags      = [
    + "dev",
    + "web",
  ]
  # (15 unchanged attributes hidden)

  # (4 unchanged blocks hidden)
}

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

```

In terraform plan we can see 1 change is going to added related to tags .

2. Run terraform apply again to update the instance:

terraform apply

```

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

google_compute_instance.vm_instance: Modifying... [id=projects/qwiklabs-gc
google_compute_instance.vm_instance: Still modifying... [id=projects/qwikl
google_compute_instance.vm_instance: Modifications complete after 15s [id=

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.

```

3. The prefix ~ means that Terraform will update the resource in-place. You can go and apply this change now by responding yes, and Terraform will add the tags to your instance.

Destructive changes

A destructive change is a change that requires the provider to replace the existing resource rather than updating it. This usually happens because the cloud provider doesn't support updating the resource in the way described by your configuration.

Changing the disk image of your instance is one example of a destructive change.

1. Edit the `boot_disk` block inside the `vm_instance` resource in your configuration file and change it to the following:

```
boot_disk {  
  initialize_params {  
    image = "cos-cloud/cos-stable"  
  }  
}
```

```
student_00_4003902989c4@cloudshell:~$ cat main.tf  
terraform {  
  required_providers {  
    google = {  
      source = "hashicorp/google"  
    }  
  }  
}  
provider "google" {  
  version = "3.5.0"  
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"  
  region  = "us-central1"  
  zone    = "us-central1-c"  
}  
resource "google_compute_network" "vpc_network" {  
  name = "terraform-network"  
}  
resource "google_compute_instance" "vm_instance" {  
  name         = "terraform-instance"  
  machine_type = "f1-micro"  
  tags         = ["web", "dev"]  
  boot_disk {  
    initialize_params {  
      image = "cos-cloud/cos-stable"  
    }  
  }  
  network_interface {  
    network = google_compute_network.vpc_network.name  
    access_config {  
    }  
  }  
}
```

```

~ network_interface {
  ~ name           = "nic0" -> (known after apply)
  ~ network        = "https://www.googleapis.com/compute/v1/"
  ~ network_ip     = "10.128.0.2" -> (known after apply)
  ~ subnetwork     = "https://www.googleapis.com/compute/v1/"
after apply)
  ~ subnetwork_project = "qwiklabs-gcp-03-5b76bf8e8e3c" -> (known after apply)

  ~ access_config {
    ~ nat_ip       = "34.134.9.61" -> (known after apply)
    ~ network_tier = "PREMIUM" -> (known after apply)
  }
}

~ scheduling {
  ~ automatic_restart = true -> (known after apply)
  ~ on_host_maintenance = "MIGRATE" -> (known after apply)
  ~ preemptible       = false -> (known after apply)

+ node_affinities {
  + key      = (known after apply)
  + operator = (known after apply)
  + values   = (known after apply)
}

- shielded_instance_config {
  - enable_integrity_monitoring = true -> null
  - enable_secure_boot         = false -> null
  - enable_vtpm                = true -> null
}

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

```

2. Now run `terraform apply` again to see how Terraform will apply this change to the existing resources:

`terraform apply`

```

Enter a value: yes
google_compute_instance.vm_instance: Destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance]
google_compute_instance.vm_instance: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance, 10s elapsed]
google_compute_instance.vm_instance: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance, 20s elapsed]
google_compute_instance.vm_instance: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance, 30s elapsed]
google_compute_instance.vm_instance: Destruction complete after 32s
google_compute_instance.vm_instance: Creating...
google_compute_instance.vm_instance: Still creating... [10s elapsed]
google_compute_instance.vm_instance: Creation complete after 16s [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.

```

The prefix `-/+` means that Terraform will destroy and recreate the resource, rather than updating it in-place. While some attributes can be updated in-place (which are shown with the `~` prefix), changing the boot disk image for an instance requires recreating it. Terraform and the Google Cloud provider handle these details for you, and the execution plan makes it clear what Terraform will do.

Additionally, the execution plan shows that the disk image change is what required your instance to be replaced. Using this information, you can adjust your changes to possibly avoid destroy/create updates if they are not acceptable in some situations.

3. Once again, Terraform prompts for approval of the execution plan before proceeding. Answer yes to execute the planned steps.

As indicated by the execution plan, Terraform first destroyed the existing instance and then created a new one in its place. You can use `terraform show` again to see the new values associated with this instance.

Destroy infrastructure

You have now seen how to build and change infrastructure. Before moving on to creating multiple resources and showing resource dependencies, you will see how to completely destroy your Terraform-managed infrastructure.

Destroying your infrastructure is a rare event in production environments. But if you're using Terraform to spin up multiple environments such as development, testing, and staging, then destroying is often a useful action.

Resources can be destroyed using the `terraform destroy` command, which is similar to `terraform apply` but it behaves as if all of the resources have been removed from the configuration.

- Try the `terraform destroy` command. Answer yes to execute this plan and destroy the infrastructure:

```
terraform destroy
```



```

- network_interface {
  - name           = "nic0" -> null
  - network        = "https://www.googleapis.com/compute/v1/p
  - network_ip     = "10.128.0.3" -> null
  - subnetwork     = "https://www.googleapis.com/compute/v1/p
  - subnetwork_project = "qwiklabs-gcp-03-5b76bf8e8e3c" -> null

  - access_config {
    - nat_ip       = "34.134.9.61" -> null
    - network_tier = "PREMIUM" -> null
  }
}

- scheduling {
  - automatic_restart = true -> null
  - on_host_maintenance = "MIGRATE" -> null
  - preemptible        = false -> null
}

- shielded_instance_config {
  - enable_integrity_monitoring = true -> null
  - enable_secure_boot         = false -> null
  - enable_vtpm                = true -> null
}
}

# google_compute_network.vpc_network will be destroyed
- resource "google_compute_network" "vpc_network" {
  - auto_create_subnetworks = true -> null
  - delete_default_routes_on_create = false -> null
  - id                             = "projects/qwiklabs-gcp-03-5b76b
  - name                           = "terraform-network" -> null
  - project                        = "qwiklabs-gcp-03-5b76bf8e8e3c"
  - routing_mode                 = "REGIONAL" -> null
  - self_link                     = "https://www.googleapis.com/com
}

```

```

google_compute_instance.vm_instance: Destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance]
google_compute_instance.vm_instance: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance, 10s elapsed]
google_compute_instance.vm_instance: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance, 20s elapsed]
google_compute_instance.vm_instance: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/zones/us-central1-c/instances/terraform-instance, 30s elapsed]
google_compute_instance.vm_instance: Destruction complete after 31s
google_compute_network.vpc_network: Destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network]
google_compute_network.vpc_network: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network, 10s elapsed]
google_compute_network.vpc_network: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network, 20s elapsed]
google_compute_network.vpc_network: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network, 30s elapsed]
google_compute_network.vpc_network: Still destroying... [id-projects/qwiklabs-gcp-03-5b76bf8e8e3c/global/networks/terraform-network, 40s elapsed]
google_compute_network.vpc_network: Destruction complete after 47s

Destroy complete! Resources: 2 destroyed.

```

The `-` prefix indicates that the instance and the network will be destroyed. As with `apply`, Terraform shows its execution plan and waits for approval before making any changes.

Just like with `terraform apply`, Terraform determines the order in which things must be destroyed. Google Cloud won't allow a VPC network to be deleted if there are resources still in it, so Terraform waits until the instance is destroyed before destroying the network. When performing operations, Terraform creates a dependency graph to

determine the correct order of operations. In more complicated cases with multiple resources, Terraform will perform operations in parallel when it's safe to do so.

Task 3. Create resource dependencies

In this section, you will learn more about resource dependencies and how to use resource parameters to share information about one resource with other resources.

Real-world infrastructure has a diverse set of resources and resource types. Terraform configurations can contain multiple resources, multiple resource types, and these types can even span multiple providers.

In this section, you will be shown a basic example of how to configure multiple resources and how to use resource attributes to configure other resources.

- Recreate your network and instance. After you respond to the prompt with yes, the resources will be created:

terraform apply

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

```
Warning: Version constraints inside provider configuration blocks a
```

```
on main.tf line 9, in provider "google":
  9:   version = "3.5.0"
```

```
Terraform 0.13 and earlier allowed provider version constraints ins
version of Terraform. To silence this warning, move the provider ve
```

```
Do you want to perform these actions?
```

```
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
```

```
Enter a value: yes
```

```
google_compute_network.vpc_network: Creating...
```

```
google_compute_network.vpc_network: Still creating... [10s elapsed]
```

```
google_compute_network.vpc_network: Still creating... [20s elapsed]
```

```
google_compute_network.vpc_network: Still creating... [30s elapsed]
```

```
google_compute_network.vpc_network: Creation complete after 38s [id=p
```

```
google_compute_instance.vm_instance: Creating...
```

```
google_compute_instance.vm_instance: Still creating... [10s elapsed]
```

```
google_compute_instance.vm_instance: Still creating... [20s elapsed]
```

```
google_compute_instance.vm_instance: Creation complete after 25s [id=
```

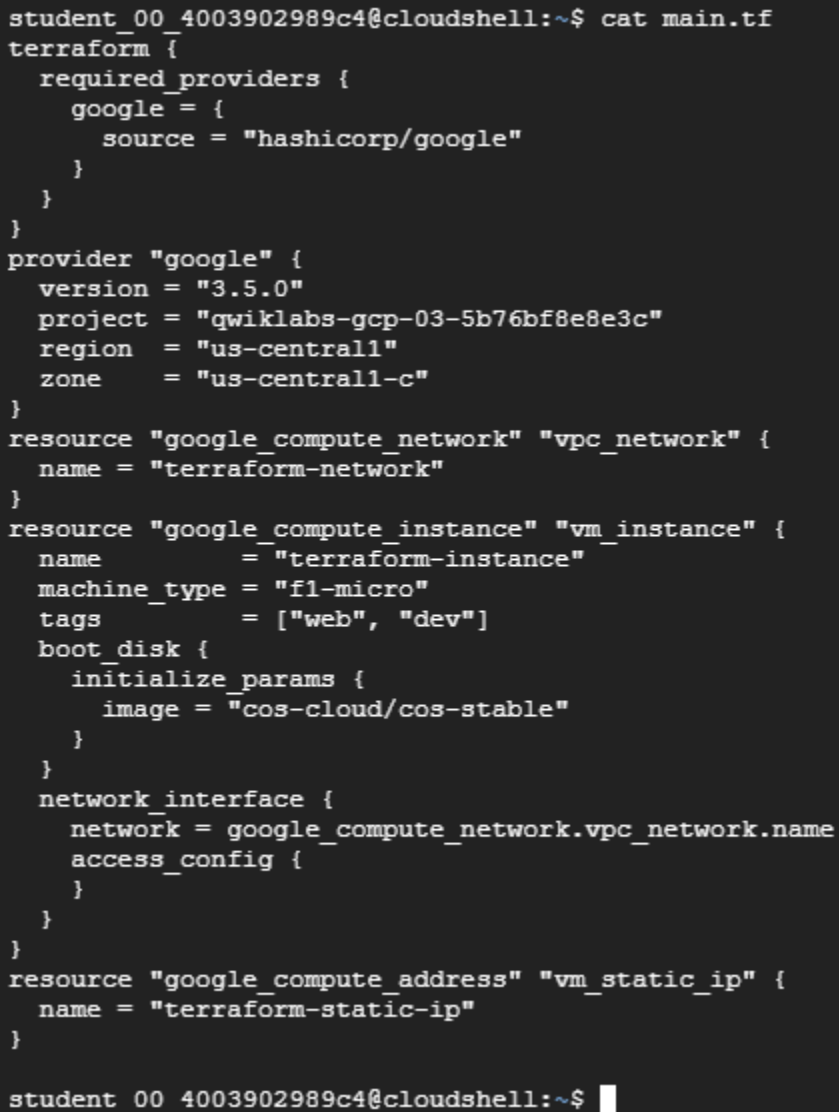
```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

```
student 00 4003902989c4@cloudshell:~$
```

Assigning a static IP address

1. Now add to your configuration by assigning a static IP to the VM instance in `main.tf`:

```
resource "google_compute_address" "vm_static_ip" {  
  name = "terraform-static-ip"  
}
```

A screenshot of a terminal window with a dark background. The prompt is 'student_00_4003902989c4@cloudshell:~\$'. The user has entered 'cat main.tf'. The output shows a Terraform configuration file. It starts with a 'terraform' block containing 'required_providers' for 'google' with source 'hashicorp/google'. Then a 'provider' block for 'google' with version '3.5.0', project 'qwiklabs-gcp-03-5b76bf8e8e3c', region 'us-central1', and zone 'us-central1-c'. Next is a 'resource' block for 'google_compute_network' named 'vpc_network' with name 'terraform-network'. Then a 'resource' block for 'google_compute_instance' named 'vm_instance' with machine type 'f1-micro', tags ['web', 'dev'], and a boot disk with image 'cos-cloud/cos-stable'. It also has a 'network_interface' block that references the 'vpc_network' and has an empty 'access_config' block. Finally, there is a 'resource' block for 'google_compute_address' named 'vm_static_ip' with name 'terraform-static-ip'. The prompt returns at the bottom.

```
student_00_4003902989c4@cloudshell:~$ cat main.tf  
terraform {  
  required_providers {  
    google = {  
      source = "hashicorp/google"  
    }  
  }  
}  
  
provider "google" {  
  version = "3.5.0"  
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"  
  region  = "us-central1"  
  zone    = "us-central1-c"  
}  
  
resource "google_compute_network" "vpc_network" {  
  name = "terraform-network"  
}  
  
resource "google_compute_instance" "vm_instance" {  
  name         = "terraform-instance"  
  machine_type = "f1-micro"  
  tags         = ["web", "dev"]  
  boot_disk {  
    initialize_params {  
      image = "cos-cloud/cos-stable"  
    }  
  }  
  network_interface {  
    network = google_compute_network.vpc_network.name  
    access_config {  
    }  
  }  
}  
  
resource "google_compute_address" "vm_static_ip" {  
  name = "terraform-static-ip"  
}  
  
student_00_4003902989c4@cloudshell:~$
```

This should look familiar from the earlier example of adding a VM instance resource, except this time you're creating a "google_compute_address" resource type. This resource type allocates a reserved IP address to your project.

2. Next, run `terraform plan`:

terraform plan

```
student_00_4003902989c4@cloudshell:~$ terraform plan
google_compute_network.vpc_network: Refreshing state... [id=projects/qw
google_compute_instance.vm_instance: Refreshing state... [id=projects/q

Terraform used the selected providers to generate the following execution
+ create

Terraform will perform the following actions:

# google_compute_address.vm_static_ip will be created
+ resource "google_compute_address" "vm_static_ip" {
  + address           = (known after apply)
  + address_type      = "EXTERNAL"
  + creation_timestamp = (known after apply)
  + id                = (known after apply)
  + name              = "terraform-static-ip"
  + network_tier       = (known after apply)
  + project            = (known after apply)
  + purpose            = (known after apply)
  + region             = (known after apply)
  + self_link          = (known after apply)
  + subnetwork         = (known after apply)
  + users              = (known after apply)
}

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

You can see what will be created with terraform plan:

Unlike terraform apply, the plan command will only show what would be changed, and never actually apply the changes directly. Notice that the only change you have made so far is to add a static IP. Next, you need to attach the IP address to your instance.

3. Update the network_interface configuration for your instance like so:

```
network_interface {
  network = google_compute_network.vpc_network.self_link
  access_config {
    nat_ip = google_compute_address.vm_static_ip.address
  }
}
```

```

required_providers {
  google = {
    source = "hashicorp/google"
  }
}
provider "google" {
  version = "3.5.0"
  project = "qwiklabs-gcp-03-5b76bf8e8e3c"
  region  = "us-centrall"
  zone    = "us-centrall-c"
}
resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}
resource "google_compute_instance" "vm_instance" {
  name         = "terraform-instance"
  machine_type = "f1-micro"
  tags         = ["web", "dev"]
  boot_disk {
    initialize_params {
      image = "cos-cloud/cos-stable"
    }
  }
  network_interface {
    network = google_compute_network.vpc_network.name
    access_config {
      nat_ip = google_compute_address.vm_static_ip.address
    }
  }
}
resource "google_compute_address" "vm_static_ip" {
  name = "terraform-static-ip"
}

```

The access_config block has several optional arguments, and in this case you'll set nat_ip to be the static IP address. When Terraform reads this configuration, it will:

- Ensure that vm_static_ip is created before vm_instance
- Save the properties of vm_static_ip in the state
- Set nat_ip to the value of the vm_static_ip.address property

4. Run terraform plan again, but this time, save the plan:

```
terraform plan -out static_ip
```

```
student_00_4003902989c4@cloudshell:~$ terraform plan -out static_ip
google_compute_network.vpc_network: Refreshing state... [id=projects/qwiklabs-gcp-03-5b76bf8e8e3c/zon
google_compute_instance.vm_instance: Refreshing state... [id=projects/qwiklabs-gcp-03-5b76bf8e8e3c/zon
```

Terraform used the selected providers to generate the following execution plan: Resource actions are indicated with the following symbols:

```
+ create
~ update in-place
```

Terraform will perform the following actions:

```
# google_compute_address.vm_static_ip will be created
+ resource "google_compute_address" "vm_static_ip" {
  + address           = (known after apply)
  + address_type      = "EXTERNAL"
  + creation_timestamp = (known after apply)
  + id                = (known after apply)
  + name              = "terraform-static-ip"
  + network_tier       = (known after apply)
  + project            = (known after apply)
  + purpose            = (known after apply)
  + region             = (known after apply)
  + self_link          = (known after apply)
  + subnetwork         = (known after apply)
  + users              = (known after apply)
}
```

```
# google_compute_instance.vm_instance will be updated in-place
~ resource "google_compute_instance" "vm_instance" {
  id          = "projects/qwiklabs-gcp-03-5b76bf8e8e3c/zon
  name        = "terraform-instance"
  tags        = [
    "dev",
    "web",
  ]
  # (15 unchanged attributes hidden)

  ~ network_interface {
    name      = "nic0"
    # (4 unchanged attributes hidden)

    ~ access_config {
      ~ nat_ip = "146.148.85.187" -> (known after apply)
      # (1 unchanged attribute hidden)
    }
  }
  # (3 unchanged blocks hidden)
}
```

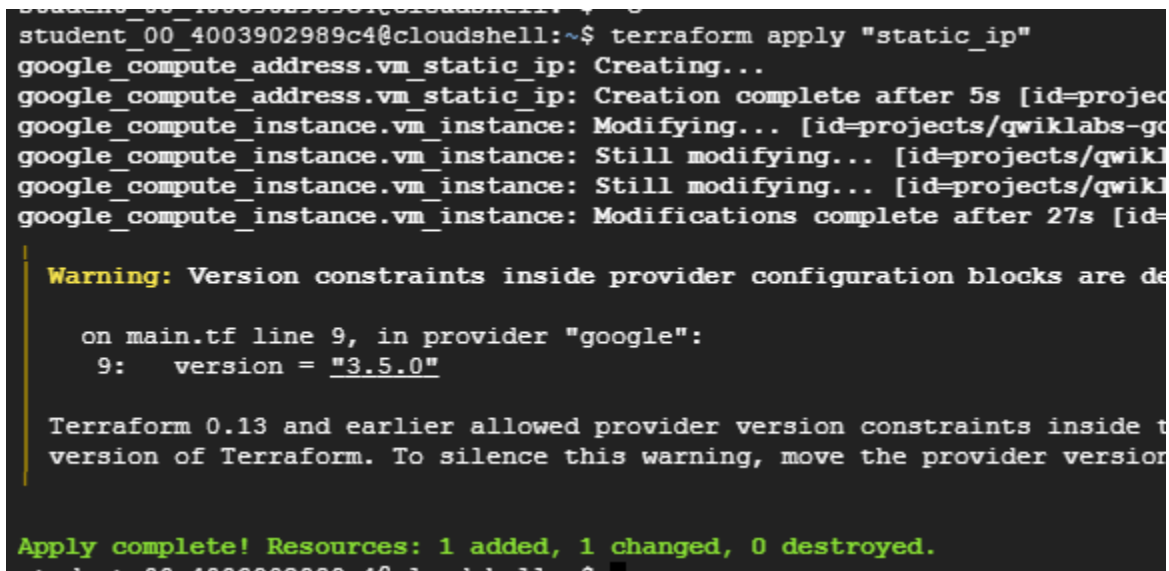
Plan: 1 to add, 1 to change, 0 to destroy.

Saving the plan this way ensures that you can apply exactly the same plan in the future. If you try to apply the file created by the plan, Terraform will first check to make sure the exact same set of changes will be made before applying the plan.

In this case, you can see that Terraform will create a new `google_compute_address` and update the existing VM to use it.

5. Run `terraform apply "static_ip"` to see how Terraform plans to apply this change:

```
terraform apply "static_ip"
```



```
student_00_4003902989c4@cloudshell:~$ terraform apply "static_ip"
google_compute_address.vm_static_ip: Creating...
google_compute_address.vm_static_ip: Creation complete after 5s [id=projects/qwiklabs-gcp-01-4003902989c4/global/addresses/vm-static-ip]
google_compute_instance.vm_instance: Modifying... [id=projects/qwiklabs-gcp-01-4003902989c4/zones/us-central1-a/instances/vm-instance]
google_compute_instance.vm_instance: Still modifying... [id=projects/qwiklabs-gcp-01-4003902989c4/zones/us-central1-a/instances/vm-instance]
google_compute_instance.vm_instance: Still modifying... [id=projects/qwiklabs-gcp-01-4003902989c4/zones/us-central1-a/instances/vm-instance]
google_compute_instance.vm_instance: Modifications complete after 27s [id=projects/qwiklabs-gcp-01-4003902989c4/zones/us-central1-a/instances/vm-instance]

Warning: Version constraints inside provider configuration blocks are deprecated
on main.tf line 9, in provider "google":
  9:   version = "3.5.0"

Terraform 0.13 and earlier allowed provider version constraints inside the
configuration. This warning has been added to help you identify configurations
that will break with a future version of Terraform. To silence this warning, move the provider version
constraint to the provider block.

Apply complete! Resources: 1 added, 1 changed, 0 destroyed.
```

As shown above, Terraform created the static IP before modifying the VM instance. Due to the interpolation expression that passes the IP address to the instance's network interface configuration, Terraform is able to infer a dependency, and knows it must create the static IP before updating the instance.

Implicit and explicit dependencies

By studying the resource attributes used in interpolation expressions, Terraform can automatically infer when one resource depends on another. In the example above, the reference to `google_compute_address.vm_static_ip.address` creates an *implicit dependency* on the `google_compute_address` named `vm_static_ip`.

Terraform uses this dependency information to determine the correct order in which to create and update different resources. In the example above, Terraform knows that the `vm_static_ip` must be created before the `vm_instance` is updated to use it.

Implicit dependencies via interpolation expressions are the primary way to inform Terraform about these relationships, and should be used whenever possible.

Sometimes there are dependencies between resources that are *not* visible to Terraform. The `depends_on` argument can be added to any resource and accepts a list of resources to create explicit dependencies for.

For example, perhaps an application you will run on your instance expects to use a specific Cloud Storage bucket, but that dependency is configured inside the application code and thus not visible to Terraform. In that case, you can use `depends_on` to explicitly declare the dependency.

1. Add a Cloud Storage bucket and an instance with an explicit dependency on the bucket by adding the following to `main.tf`:

```
# New resource for the storage bucket our application will use.
```

```
resource "google_storage_bucket" "example_bucket" {  
  name      = "<UNIQUE-BUCKET-NAME>"  
  location = "US"  
  website {  
    main_page_suffix = "index.html"  
    not_found_page   = "404.html"  
  }  
}
```

```
# Create a new instance that uses the bucket
```

```
resource "google_compute_instance" "another_instance" {  
  # Tells Terraform that this VM instance must be created only after the  
  # storage bucket has been created.  
  depends_on = [google_storage_bucket.example_bucket]  
  name       = "terraform-instance-2"  
  machine_type = "f1-micro"  
  boot_disk {  
    initialize_params {  
      image = "cos-cloud/cos-stable"  
    }  
  }  
}
```



```

}

network_interface {

  network = google_compute_network.vpc_network.self_link

  access_config {

  }

}

}

```

Note: Storage buckets must be globally unique. Because of this, you will need to replace UNIQUE-BUCKET-NAME with a unique, valid name for a bucket. Using your name and the date is usually a good way to guess a unique bucket name.

You may wonder where in your configuration these resources should go. The order that resources are defined in a terraform configuration file has no effect on how Terraform applies your changes. Organize your configuration files in a way that makes the most sense for you and your team.

2. Now run terraform plan and terraform apply to see these changes in action:

terraform plan

terraform apply

```

# google_storage_bucket.new-bucket will be created
+ resource "google_storage_bucket" "new-bucket" {
  + bucket_policy_only = (known after apply)
  + force_destroy      = false
  + id                 = (known after apply)
  + location           = "US"
  + name               = "new-bucket"
  + project            = (known after apply)
  + self_link          = (known after apply)
  + storage_class      = "STANDARD"
  + url                = (known after apply)

  + website {
    + main_page_suffix = "index.html"
    + not_found_page   = "404.html"
  }
}

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

```

```
google_storage_bucket.new-bucket-today: Creating...
google_storage_bucket.new-bucket-today: Creation complete after 2s [id=new-bucket-today]
google_compute_instance.another_instance: Creating...
google_compute_instance.another_instance: Still creating... [10s elapsed]
google_compute_instance.another_instance: Creation complete after 16s [id=projects/qwiklabs

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

3. Before moving on, remove these new resources from your configuration and run `terraform apply` once again to destroy them. You won't use the bucket or the second instance any further in the getting started guide.

Task 4. Provision infrastructure

The compute instance you launched at this point is based on the Google image given, but has no additional software installed or configuration applied.

Google Cloud allows customers to manage their own [custom operating system images](#). This can be a great way to ensure the instances you provision with Terraform are pre-configured based on your needs. [Packer](#) is the perfect tool for this and includes a [builder for Google Cloud](#).

Terraform uses provisioners to upload files, run shell scripts, or install and trigger other software like configuration management tools.

Defining a provisioner

1. To define a provisioner, modify the resource block defining the first `vm_instance` in your configuration to look like the following:

```
resource "google_compute_instance" "vm_instance" {

  name          = "terraform-instance"

  machine_type  = "f1-micro"

  tags          = ["web", "dev"]

  provisioner "local-exec" {

    command = "echo ${google_compute_instance.vm_instance.name}:
${google_compute_instance.vm_instance.network_interface[0].access_config[0]
.nat_ip} >> ip_address.txt"

  }
```

```
# ...  
}
```

This adds a provisioner block within the resource block. Multiple provisioner blocks can be added to define multiple provisioning steps. Terraform supports [many provisioners](#), but for this example you are using the `local-exec` provisioner.

The `local-exec` provisioner executes a command locally on the machine running Terraform, not the VM instance itself. You're using this provisioner versus the others so we don't have to worry about specifying any [connection info](#) right now.

This also shows a more complex example of string interpolation than you've seen before. Each VM instance can have multiple network interfaces, so refer to the first one with `network_interface[0]`, count starting from 0, as most programming languages do. Each network interface can have multiple `access_config` blocks as well, so once again you specify the first one.

2. Run `terraform apply`:

```
terraform apply
```

At this point, the output may be confusing at first.

Terraform found nothing to do - and if you check, you'll find that there's no `ip_address.txt` file on your local machine.

Terraform treats provisioners differently from other arguments. Provisioners only run when a resource is created, but adding a provisioner does not force that resource to be destroyed and recreated.

3. Use `terraform taint` to tell Terraform to recreate the instance:

```
terraform taint google_compute_instance.vm_instance
```

A tainted resource will be destroyed and recreated during the next `apply`.

```
student_00_4003902989c4@cloudshell:~$ terraform taint google_compute_instance.vm_instance  
Resource instance google_compute_instance.vm_instance has been marked as tainted.  
student_00_4003902989c4@cloudshell:~$ terraform apply
```

4. Run `terraform apply` now:

```
terraform apply
```

5. Verify everything worked by looking at the contents of the `ip_address.txt` file.

It contains the IP address, just as you asked.

Failed provisioners and tainted resources

If a resource is successfully created but fails a provisioning step, Terraform will error and mark the resource as *tainted*. A resource that is tainted still exists, but shouldn't be considered safe to use, since provisioning failed.

When you generate your next execution plan, Terraform will remove any tainted resources and create new resources, attempting to provision them again after creation.

Destroy provisioners

Provisioners can also be defined that run only during a destroy operation. These are useful for performing system cleanup, extracting data, etc.

For many resources, using built-in cleanup mechanisms is recommended if possible (such as init scripts), but provisioners can be used if necessary.

This lab won't show any destroyed provisioner examples. If you need to use destroy provisioners, please see the [Provisioners documentation](#).

In this lab, you learned how to build, change, and destroy infrastructure with Terraform. You then created resource dependencies, and provisioned basic infrastructure with Terraform configuration files.