Change infrastructure

4min |

Reference this often? Create an account to bookmark tutorials.

In the last tutorial, you created your first infrastructure with Terraform: a single Docker container. In this tutorial, you will modify that resource, and learn how to apply changes to your Terraform projects.

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

When using Terraform in production, we recommend that you use a version control system to manage your configuration files, and store your state in a remote backend such as HCP Terraform or Terraform Enterprise.

Prerequisites

This tutorial assumes that you are continuing from the previous tutorials. If not, follow the steps below before continuing.

Install the Terraform CLI (0.15+), and the Docker CLI (configured with a default profile), as described in the install tutorial.

Create a directory named `learn-terraform-docker-container` and paste the following configuration into a file named `main.tf`.

Mac or Linux | Windows

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}

provider "docker" {}

resource "docker_image" "nginx" {
  name         = "nginx:latest"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"
  ports {
    internal = 80
    external = 8000
  }
}
```

Initialize the configuration.

```
$ terraform init
```

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

```
$ terraform apply
```

Once you have successfully applied the configuration, you can continue with the rest of this tutorial.

Update configuration

Now update the external port number of your container. Change the `docker_container.nginx` resource under the provider block in `main.tf` by replacing the `ports.external` value of `8000` with `8080`

> **Tip**
> The below snippet is formatted as a diff to give you context about which parts of your configuration you need to change. Replace the content displayed in red with the content displayed in green, leaving out the leading `+` and `-` signs.]

```
resource "docker_container" "nginx" {
  image = docker_image.nginx.latest
  name  = "tutorial"
  hostname = "learn-terraform-docker"
  ports {
    internal = 80
-   external = 8000
+   external = 8080
  }
}
```

This update changes the port number your container uses to serve your nginx server. The Docker provider knows that it cannot change the port of a container after it has been created, so Terraform will destroy the old container and create a new one.

Apply Changes

After changing the configuration, run `terraform apply` again to see how Terraform will apply this change to the existing resources.

```
$ terraform apply
docker_image.nginx: Refreshing state... [id=sha256:d1a364dc548d5357f0da3268c888e1971b
docker_container.nginx: Refreshing state... [id=5896c6cb7ae654503c36562472b573da8f490

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # docker_container.nginx must be replaced
-/+ resource "docker_container" "nginx" {
##...
      ~ ports {
          ~ external = 8000 -> 8080 # forces replacement
            # (3 unchanged attributes hidden)
        }
    }

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

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

The prefix `-/+` means that Terraform will destroy and recreate the resource, rather than updating it in-place. Terraform can update some attributes in-place (indicated with the `~` prefix), but changing the port for a Docker container requires recreating it. Terraform handles these details for you, and the execution plan displays what Terraform will do.

Additionally, the execution plan shows that the port change is what forces Terraform to replace the container. Using this information, you can adjust your changes to to avoid destructive updates if necessary.

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

```
docker_container.nginx: Destroying... [id=5896c6cb7ae654503c36562472b573da8f4905[redacted]46
docker_container.nginx: Destruction complete after 1s
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 1s [id=b2140f8c6aa79f62c8ac3c3d792f20

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

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

Previous

Build

Next

Destroy