Query data with outputs

- 5min
- Terraform

Reference this often? Create an account to bookmark tutorials.

In the previous tutorial, you used an input variable to parameterize your Terraform configuration. In this tutorial, you will use output values to organize data to be easily queried and displayed to the Terraform user.

If you have not yet completed the Define Input Variables tutorial, do so before following this one.

Initial configuration

After following the previous tutorials, you will have a directory named `learn-terraform-docker-container` with the following configuration.

Mac or LinuxWindows

```
# main.tf

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  = var.container_name
  ports {
    internal = 80
    external = 8080
  }
}

# variables.tf
```

```
variable "container_name" {
  description = "Value of the name for the Docker container"
  type        = string
  default     = "ExampleNginxContainer"
}
```

Ensure that your configuration matches this, and that you have initialized your configuration in the `learn-terraform-docker-container` directory.

```
$ terraform init
```

Apply the configuration before continuing this tutorial. Respond to the confirmation prompt with a `yes`.

```
$ terraform apply
```

Output Docker container configuration

Create a file called `outputs.tf` in your `learn-terraform-docker-container` directory. Add the configuration below to `outputs.tf` to define outputs for your container's ID and the image ID.

```
output "container_id" {
  description = "ID of the Docker container"
  value       = docker_container.nginx.id
}

output "image_id" {
  description = "ID of the Docker image"
  value       = docker_image.nginx.id
}
```

Inspect output values

You must apply this configuration before you can use these output values. Apply your configuration now. Respond to the confirmation prompt with `yes`.

```
$ terraform apply
docker_image.nginx: Refreshing state...
[id=sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest]
docker_container.nginx: Refreshing state...
[id=a7a6d308b8d4f77cda08ce20729f899530264d487cc1fd8ca92f9eb591ebcc6d]

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" {
##...
Plan: 1 to add, 0 to change, 1 to destroy.

Changes to Outputs:
  + container_id = (known after apply)
  + image_id     =
"sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest"

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

docker_container.nginx: Destroying...
[id=a7a6d308b8d4f77cda08ce20729f899530264d487cc1fd8ca92f9eb591ebcc6d]
docker_container.nginx: Destruction complete after 1s
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 2s
[id=e5fff27c62e26dc9504d21980543f21161225ab483a1e534a98311a677b9453a]

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

Outputs:

container_id = "e5fff27c62e26dc9504d21980543f21161225ab483a1e534a98311a677b9453a"
image_id =
"sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest"
```

Terraform prints output values to the screen when you apply your configuration. Query the outputs with the `terraform output` command.

```
$ terraform output
container_id = "e5fff27c62e26dc9504d21980543f21161225ab483a1e534a98311a677b9453a"
image_id =
"sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest"
```

You can use Terraform outputs to connect your Terraform projects with other parts of your infrastructure, or with other Terraform projects. To learn more, follow our in-depth tutorial, Output Data from Terraform.

Destroy infrastructure

Destroy your infrastructure. Respond to the confirmation prompt with `yes`.

```
$ terraform destroy
docker_image.nginx: Refreshing state...
[id=sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest]
docker_container.nginx: Refreshing state...
[id=3a81c009ce50689663c80b8af5f45453543e192d0c23c24e7e883f9391489b1f]

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

Terraform will perform the following actions:

  # docker_container.nginx will be destroyed
  - resource "docker_container" "nginx" {
##...
  # docker_image.nginx will be destroyed
  - resource "docker_image" "nginx" {
##...

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

Changes to Outputs:
  - container_id    = "3a81c009ce50689663c80b8af5f45453543e192d0c23c24e7e883f9391489b1f" ->
null
```

```
  - image_id =
"sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest" ->
null

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

docker_container.nginx: Destroying...
[id=3a81c009ce50689663c80b8af5f45453543e192d0c23c24e7e883f9391489b1f]
docker_container.nginx: Destruction complete after 1s
docker_image.nginx: Destroying...
[id=sha256:d1a364dc548d5357f0da3268c888e1971bbdb957ee3f028fe7194f1d61c6fdeenginx:latest]
docker_image.nginx: Destruction complete after 1s

Destroy complete! Resources: 2 destroyed.
```

Next steps

That concludes the getting started tutorials for Terraform. Hopefully you're now able to not only see what Terraform is useful for, but you're also able to put this knowledge to use to improve building your own infrastructure.

For more hands-on experience with the Terraform configuration language, or to learn more of the building blocks of Terraform, review the tutorials below.

- Configuration Language - Get more familiar with variables, outputs, dependencies, meta-arguments, and other language features to write more sophisticated Terraform configurations. These tutorials use the AWS provider. To configure your AWS credentials, use the AWS Get Started Build tutorial.
- Modules - Organize and re-use Terraform configuration with modules.
- Provision - Use Packer or Cloud-init to automatically provision SSH keys and a web server onto a Linux VM created by Terraform in AWS.
- Import - Import existing infrastructure into Terraform.

To read more about available configuration options, explore the Terraform documentation.

Learn more about HCP Terraform

Although HCP Terraform can act as a standard remote backend to support Terraform runs on local machines, it works even better as a remote run environment. It supports two main workflows for performing Terraform runs:

- A VCS-driven workflow, in which it automatically queues plans whenever changes are committed to your configuration's VCS repo.

- An API-driven workflow, in which a CI pipeline or other automated tool can upload configurations directly.

For a hands-on introduction to the HCP Terraform VCS-driven workflow, follow the HCP Terraform getting started tutorials. HCP Terraform also offers commercial solutions which include team permission management, policy enforcement, agents, and more.