BOSCH
Invented for life

Terraform infrastructure setup

inside.Docupedia Export

Author: Shanmugapriya Manickaraj (RBEI/BSF6)
Date: 17-Jul-2019 12:25

Table of Contents

1 Install Terraform:

To install Terraform on any supported system:

1. Find the appropriate Terraform distribution package for your system and download it. Terraform is distributed as a single `.zip` file.
2. After downloading Terraform, unzip the package to a directory of your choosing. Terraform runs as a single binary named `terraform`. Any other files in the package can be safely removed and Terraform will still function.
3. Optional but recommended: modify the path to include the directory that contains the Terraform binary.

After installing Terraform, verify the installation by opening a new terminal session and checking that Terraform is available. Execute `terraform` at the prompt, and you should see output similar to this:

```
meier@Azure:~/tftest$ terraform
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a Terraform working directory
    output             Read an output from a state file
    plan               Generate and show an execution plan
    providers          Prints a tree of the providers used in the configuration
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
```

2 Initialization

The first command to run for a new configuration is `terraform init`, which initializes various local settings and data that will be used by subsequent commands.

Terraform uses a plugin-based architecture to support the numerous infrastructure and service providers available. Each provider is its own encapsulated binary distributed separately from Terraform itself. The `terraform init` command will automatically download and install any provider binary for the providers in use within the configuration.

```
meier@Azure:~/tftest$ terraform init

Initializing provider plugins...

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.
meier@Azure:~/tftest$
```

3 Apply changes:

In the same directory as the `main.tf` file you created, run `terraform apply`.

You should see output similar to this:

```
meier@Azure:~/tftest$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + azurerm_resource_group.rg
      id:                   <computed>
      location:             "westus2"
      name:                 "myTERRAFORMResourceGroup"
      tags.%:               <computed>

  + azurerm_subnet.subnet
      id:                   <computed>
      address_prefix:       "10.0.1.0/24"
      ip_configurations.#:  <computed>
      name:                 "myTERRAFORMSubnet"
      resource_group_name:  "myTERRAFORMResourceGroup"
      virtual_network_name: "myTERRAFORMVnet"

  + azurerm_virtual_network.vnet
      id:                   <computed>
      address_space.#:      "1"
      address_space.0:      "10.0.0.0/16"
      location:             "westus2"
      name:                 "myTERRAFORMVnet"
      resource_group_name:  "myTERRAFORMResourceGroup"
      subnet.#:             <computed>
      tags.%:               <computed>


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

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
```

```
azurerm_resource_group.rg: Creating...
  location: "" => "westus2"
  name:     "" => "myTERRAFORMResourceGroup"
  tags.%:   "" => "<computed>"
azurerm_resource_group.rg: Creation complete after 3s (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup)
azurerm_virtual_network.vnet: Creating...
  address_space.#:     "" => "1"
  address_space.0:     "" => "10.0.0.0/16"
  location:            "" => "westus2"
  name:                "" => "myTERRAFORMVnet"
  resource_group_name: "" => "myTERRAFORMResourceGroup"
  subnet.#:            "" => "<computed>"
  tags.%:              "" => "<computed>"
azurerm_virtual_network.vnet: Still creating... (10s elapsed)
azurerm_virtual_network.vnet: Creation complete after 16s (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...etwork/virtualNetworks/myTERRAFORMVnet)
azurerm_subnet.subnet: Creating...
  address_prefix:       "" => "10.0.1.0/24"
  ip_configurations.#:  "" => "<computed>"
  name:                 "" => "myTERRAFORMSubnet"
  resource_group_name:  "" => "myTERRAFORMResourceGroup"
  virtual_network_name: "" => "myTERRAFORMVnet"
azurerm_subnet.subnet: Creation complete after 3s (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...ERRAFORMVnet/subnets/myTERRAFORMSubnet)

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
meier@Azure:~/tftest$
```

You can inspect the current state using `terraform state show` or `terraform state list`:

```
meier@Azure:~/tftest$ terraform state list
azurerm_resource_group.rg
azurerm_subnet.subnet
azurerm_virtual_network.vnet
meier@Azure:~/tftest$
```

4 Destroy

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.

```
meier@Azure:~/tftest$ terraform destroy
azurerm_resource_group.rg: Refreshing state... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup)
azurerm_virtual_network.vnet: Refreshing state... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...etwork/virtualNetworks/myTERRAFORMVnet)
azurerm_subnet.subnet: Refreshing state... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...ERRAFORMVnet/subnets/myTERRAFORMSubnet)

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - azurerm_resource_group.rg

  - azurerm_subnet.subnet

  - azurerm_virtual_network.vnet


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

azurerm_subnet.subnet: Destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...ERRAFORMVnet/subnets/myTERRAFORMSubnet)
azurerm_subnet.subnet: Still destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...ERRAFORMVnet/subnets/myTERRAFORMSubnet, 10s elapsed)
azurerm_subnet.subnet: Destruction complete after 12s
azurerm_virtual_network.vnet: Destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...etwork/virtualNetworks/myTERRAFORMVnet)
azurerm_virtual_network.vnet: Destruction complete after 1s
azurerm_resource_group.rg: Destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup)
azurerm_resource_group.rg: Still destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup, 10s elapsed)
azurerm_resource_group.rg: Still destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup, 20s elapsed)
azurerm_resource_group.rg: Still destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup, 30s elapsed)
azurerm_resource_group.rg: Still destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup, 40s elapsed)
azurerm_resource_group.rg: Still destroying... (ID: /subscriptions/8bef34ae-2480-4638-a7e9-...esourceGroups/myTERRAFORMResourceGroup, 50s elapsed)
azurerm_resource_group.rg: Destruction complete after 50s

Destroy complete! Resources: 3 destroyed.
meier@Azure:~/tftest$
```

5 Example Configuration (main.tf) :

main.tf

```
# Configure the Microsoft Azure Provider.
provider "azurerm" {
    version = "=1.20.0"
}

# Create a resource group
resource "azurerm_resource_group" "rg" {
    name     = "TerraFormResourceGroup"
    location = "westus2"
}
# Create virtual network
resource "azurerm_virtual_network" "vnet" {
    name                = "TerraFormVnet"
    address_space       = ["10.0.0.0/16"]
    location            = "westus2"
    resource_group_name = "${azurerm_resource_group.rg.name}"
}

# Create subnet
resource "azurerm_subnet" "subnet" {
    name                 = "TerraFormSubnet"
    resource_group_name  = "${azurerm_resource_group.rg.name}"
    virtual_network_name = "${azurerm_virtual_network.vnet.name}"
    address_prefix       = "10.0.1.0/24"
}
```

6 Provision:

Terraform provisioners help you do additional setup and configuration when a resource is created or destroyed. You can move files, run shell scripts, and install software.

Provisioners are not intended to maintain desired state and configuration for existing resources.

For that purpose, you should use one of the many tools for configuration management, such as Chef, Ansible, and PowerShell Desired State Configuration. (Terraform includes a chef provisioner.)

An imaged-based infrastructure can eliminate much of the need to configure resources when they are created.

In this common scenario, Terraform is used to provision infrastructure based on a custom image.

The image is managed as code.

7 Defining a provisioner

Provisioners are defined on resources, most commonly a new instance of a virtual machine or container.

Provision.tf

```
# Create a Linux virtual machine
resource "azurerm_virtual_machine" "vm" {

    <...snip...>

    provisioner "file" {
        connection {
            type     = "ssh"
            user     = "${var.admin_username}"
            password = "${var.admin_password}"
        }

        source      = "newfile.txt"
        destination = "newfile.txt"
    }


}
```

8 Running provisioners

Provisioners run when a resource is created, or a resource is destroyed.

Provisioners do not run during update operations. The example configuration for this section define one provisioner that run only when a new virtual machine instance is created.

If the virtual machine instance is later modified or destroyed, the provisioner will not run.

To run the example configuration with provisioners:

1. Copy the configuration to a file named `main.tf`, and then upload it to your Terraform working directory. It should be the only `.tf` file in the folder.
2. In Cloud Shell, type `code newfile.txt`. In the editor, add the following text: "Testing the file and remote-exec provisioners." Save the file and close the editor.
3. Run `terraform init`
4. Run `terraform plan -out=tfprov`
5. If you would like to examine the Terraform execution plan before applying it, run `terraform show tfprov`
6. Run `terraform apply tfprov`. When prompted to continue, answer `yes`.

Continue the procedure from above by doing the following:

1. Run `terraform show` to examine the current state. Notice some differences between state and the plan; for example, values shown as `<computed>` in the plan will be replaced with actual values in the state. You can still access the plan for comparison by running `terraform show tfprov`.
2. Run `terraform destroy` to destroy the infrastructure

9 Input variables:

To become truly shareable and version-controlled, we need to parameterize the configurations.

Up to now we have embedded all necessary values as literals. We're going to add a few simple variables to our configuration:

- Prefix - the string to use as a prefix on all the resources created by this configuration
- Location - the Azure region where the resources will be created

Create another file `variables.tf` with the following contents

variables.tf

```
variable "location" {}

variable "prefix" {
    type = "string"
    default = "my"
}

variable "tags" {
    type = "map"
    default = {
        Environment = "Terraform GS"
        Dept = "Engineering"
  }
}

variable "sku" {
    default = {
        westus = "16.04-LTS"
        eastus = "18.04-LTS"
    }
}
```

This defines four variables within your Terraform configuration. The first has empty brackets, which tells you that the variable is required and the type of the variable will be determined by the input value.

The second variable defines the type and sets a default. We'll discuss the other two variables a little later.

If you run `terraform apply` you will be prompted to enter a value for location.

To persist variable values, create a file named `terraform.tfvars` and assign variables within this file.

terraform.tfvars

```
location = "westus"
prefix = "tf"
```

For all files which match `terraform.tfvars` or `*.auto.tfvars` present in the current directory, Terraform automatically loads them to populate variables.

If the file is named something else, you can use the `-var-file` flag directly to specify a file.

These files are the same syntax as Terraform configuration files.

And like Terraform configuration files, these files can also be JSON.