ORACLE
Cloud Infrastructure

Terraform
Level 200

Flavio Pereira

November 2018

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Objectives

- Quick Introduction of Terraform
- Terraform State File – Local and Remote
- Terraform Target Resources
- Terraform Modules
- Terraform provisioners
- Terraform and Instance Principal Configuration

Quick Introduction of Terraform

- Terraform is written by the team at Hashicorp.
- “Infrastructure as Code” tool for building and managing infrastructure efficiently and elegantly.
- Terraform - Create, combine and manage infrastructure across multiple providers
- Terraform also integrates with configuration management and provisioning tools like Chef, Puppet and Ansible.

Terraform State file

- Terraform stores the state of your managed infrastructure from the last time Terraform was run.
- Terraform uses this state to create plans and make changes to your infrastructure.
- It is critical that this state is maintained appropriately so future runs operate as expected.

```
terraform.tfstate
```

Terraform Local State File

- State is stored locally on local machine in JSON format
- Because it must exist, it is a frequent source of merge conflicts
- It is generally acceptable for individuals and small teams
- Tends not to scale for large teams
- Requires a more "mono repo" pattern

Terraform Remote State File

- Writes the state data to a remote data store
- Allows your infrastructure to be manage by multiple teams

Configuring and using remote backends is easy and you can get it configured with Object Storage:

```
terraform {
backend "http" {
update_method = "PUT"
address = "https://objectstorage.<region>.oraclecloud.com/<my-access-uri>"
 }
}
```

Terraform – Targeting resources

- You can use the `-target` flag on both the `terraform plan` and `terraform apply` commands.
- It allows you to target a resource or more if you specify multiple `-target` flags

```
[opc@terraform-server tftest]$ terraform plan -target=oci_identity_user.user02
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

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:

  + oci_identity_user.user02
      id:             <computed>
      compartment_id: <computed>
      description:    "A user managed with Terraform"
      inactive_state: <computed>
      name:           "user02-TF"
      state:          <computed>
      time_created:   <computed>
      time_modified:  <computed>


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

Terraform Modules

- Portable Terraform configurations (packages)
- Allow separation of concerns and responsibilities among teams
- Modules are just Terraform configurations inside a folder

```
module "vcn" {
source = "./vcn"
compartment_ocid = "${var.compartment_ocid}"
tenancy_ocid = "${var.tenancy_ocid}"
vcn_dns_name = "${var.vcn_dns_name}"
label_prefix = "${var.label_prefix}"
vcn_name = "${var.vcn_name}"
vcn_cidr = "${var.vcn_cidr}"
subnet_cidr = "${var.subnet_cidr}"
availability_domains = "${var.availability_domains}"
}
```

Terraform Provisioners

- Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction.
- Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.
- Terraform can also integrate with provisioners like Chef, puppet, Ansible, shells scripts.

```
provisioner "local-exec" {
  command = "ansible-playbook -i '${self.public_ip},' --private-key
${var.ssh_private_key} setup.yml" }
```

Terraform Provisioners remote and local

```
provisioner "remote-exec" {
inline = [
"echo 'This instance was provisioned by Terraform.' | sudo tee
/etc/motd",
]
}
```

```
provisioner "local-exec" {
command = "echo ${oci_core_instance.web.private_ip} >> private_ips.txt"
}
```

Terraform Provisioners null_resource

An example below is using a provisioner to remote-exec a command to touch a file.

```
resource "null_resource" "remote-exec" {
  depends_on = ["oci_core_instance.TFInstance"]

provisioner "remote-exec" {
    connection {
      agent       = false
      timeout     = "10m"
      host        = "${data.oci_core_vnic.InstanceVnic.public_ip_address}"
      user        = "opc"
      private_key = "${var.ssh_private_key}"
    }

    inline = [
      "touch ~/IMadeAFile.Right.Here",
    ]
  }
}
```

Terraform and Instance Principal Configuration

- Instance principal authorization allows your provider to make API calls from an Oracle Cloud Infrastructure compute instance without needing the `tenancy_ocid`, `user_ocid`, `fingerprint`, and `private_key_path` attributes in your provider definition.

- To enable Instance Principal authentication, set the `auth` attribute to `"InstancePrincipal"` in the provider definition as below:

```
provider "oci" {
auth = "InstancePrincipal"
region = "${var.region}"
}
```

Summary

- Understand how to work with State Files locally and remotely
- Use Instance Principal with Terraform
- Understand Target resources
- Describe and use Terraform Modules
- Understand how to use Terraform Provisioners

ORACLE®
Cloud Infrastructure

cloud.oracle.com/iaas

cloud.oracle.com/tryit