15 Days of Terraform – Curated by Shaik Hari Sadia Anjum

Terraform Provisioners

Terraform provisioners are used to execute scripts or commands on a local machine or on a remote resource after it is created or before it's destroyed. While provisioners can be powerful, they are considered a *last resort* in Terraform best practices, which recommend using cloud-init, configuration management tools (like Ansible), or image baking instead.

Types of Provisioners in Terraform

Provisioners are executed in two phases:

- **Creation-time Provisioners**: Run after a resource is created.
- **Destroy-time Provisioners**: Run before a resource is destroyed.

There are **two main categories** of provisioners:

1. **local-exec Provisioner**

- Executes a command **on the machine running Terraform**.
- Useful for operations like sending a Slack notification, calling an API, or running a local script.

```
resource "aws_instance" "example" {

 ami           = "ami-0c55b159cbfafe1f0"

 instance_type = "t2.micro"


 provisioner "local-exec" {

  command = "echo ${self.private_ip} >> private_ips.txt"

 }

}
```

2. remote-exec Provisioner

- **Connects to the provisioned resource (like EC2) via SSH or WinRM and runs commands.**
- **Requires credentials or connection block.**

```
resource "aws_instance" "example" {
```

```
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  key_name      = "my-key"

  connection {
    type        = "ssh"
    user        = "ec2-user"
    private_key = file("~/.ssh/my-key.pem")
    host        = self.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo yum update -y",
      "sudo yum install -y nginx",
      "sudo systemctl start nginx"
    ]
  }
}
```

Common Settings for Provisioners

- **inline**: Run multiple commands as an array.
- **script**: Reference to a local script file.
- **scripts**: Array of script file paths (deprecated in newer versions).
- **environment**: Define environment variables to use with remote-exec.

Destroy-Time Provisioners

You can specify a when = "destroy" block to run commands **before** resource destruction:

```
provisioner "remote-exec" {
```

```
  when    = "destroy"
  inline  = ["echo 'Resource is being destroyed'"]
}
```

Connection Block

Used by remote-exec or file provisioners to connect to the resource:

```
connection {
  type        = "ssh"
  user        = "ubuntu"
  private_key = file("~/.ssh/id_rsa")
  host        = self.public_ip
}
```

File Provisioner

Used to **upload files** from the local system to the remote resource.

```
provisioner "file" {
  source      = "app.conf"
  destination = "/etc/myapp/app.conf"
}
```

Provisioner Failure Behavior

By default, if a provisioner fails, Terraform marks the resource as **tainted**, and it will be destroyed and recreated in the next apply.

You can control this with:

- on_failure = "continue" → Terraform will not stop even if provisioner fails.

```
provisioner "remote-exec" {
  on_failure = "continue"
  inline     = ["some command"]
}
```

Best Practices and Warnings

- **Avoid provisioners when possible**: Use baked AMIs or cloud-init for initialization.
- **Provisioners can introduce non-determinism**: Commands may fail if the remote host isn’t ready yet.
- **State Management Issues**: Terraform doesn't track what provisioners do, so re-running them requires destroying and recreating resources.
- **Debug with TF_LOG=DEBUG**: Helps when provisioners fail unexpectedly.

⚠ Note: Use provisioners only as a last resort. They are not idempotent, can cause unpredictable behavior, and are not recommended for regular infrastructure automation.