Infrastructure as code testing, Terraform Vs Terratest

Ruchira Anil More
Department of Computing, TU Dublin, Tallaght, Ireland
X00193212@myTUDublin.ie

Introduction

Cloud adoption has become vital for organizations to deliver new services. Early time to market with stable code delivery has introduced use of automation in DevOps to provision infrastructure. Such automated process of infrastructure provisioning is termed as Infrastructure-as-code. Infrastructure-as-code is one of the practices that many IT industries have embraced to automate their infrastructure provisioning and maintaining process. Though, this has been favourable in administering and provisioning infrastructure using machine readable scripts, there have been issues with the functionality since they might also have defects. The manifestation of defects in the scripts is unavoidable since they are like any other software code, lines of code in a file. Thus, it is essential to have a qualitative analysis on these files to achieve defect free Infrastructure as code. This poster represents a brief overview of the research, where a study was performed to compare the two testing tools/frameworks for PaaS IaC, Terraform test framework Vs. Terratest. Below research question were identified in order to perform further experiments:

RQ1: How does Terratest (Gruntwork) and Terraform-test (Hashicorp) compares in terms of use cases, developer experience, flexibility, exception handling and reporting.

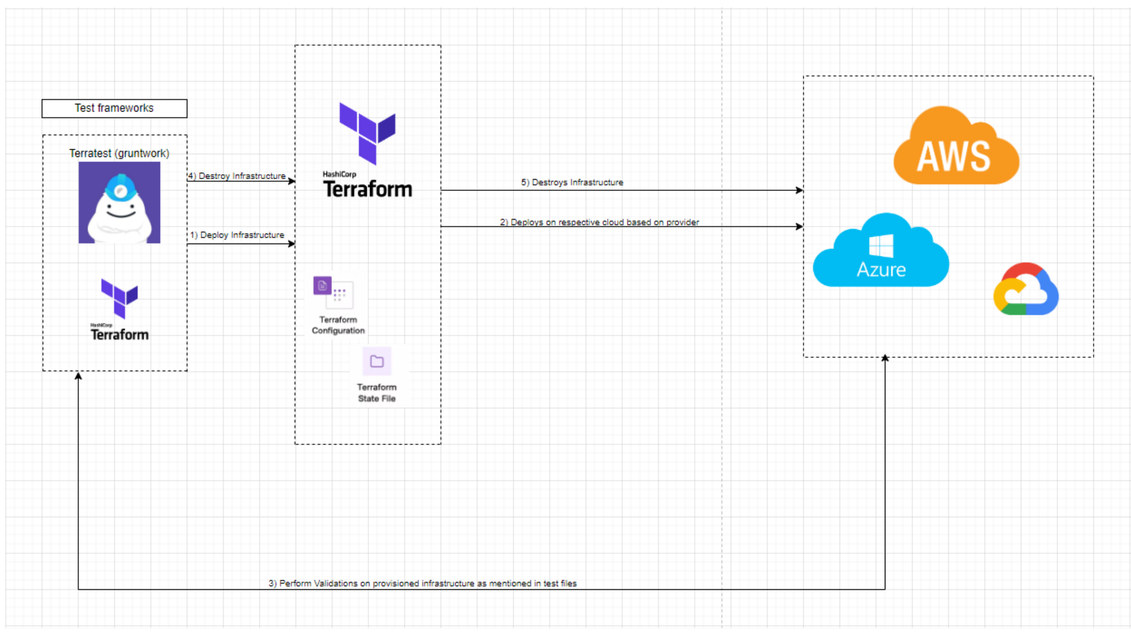
RQ2: Limitations and challenges to adopt Terratest and Terraform test.

Comparison

Metrics	Terratest (Gruntwork)	Terraform's test command (Hashicorp)
Release Date	April 2018	October 2023
Language	Go	Hashicorp's HCL
Syntax	Imperative	Declarative
Exception Handling	Error handling and timeouts implemented to allow scripts to terminate in case of exception.Sometimes infrastructure is not destroyed.	Exceptions are handled and the test terminates and destroys infrastructure as clean-up process.
Built in Helper methods	Good support	Minimal support
Flexibility	Allows to control flow of execution	No control from test framework, only possible from within terraform
Learning Curve	New language, may take some time to kick start	HCL, easy to learn
Execution time	Same as terraform code execution	Same as terraform code execution
Assertion	Importing 3rd party go library for assertions	Built in assertions on conditions
Testing stages	Unit, Integration, end2end	Unit, and Integration. end2end is not supported
Tool support	Terraform, packer, docker , K8s	Terraform, k8s provider within terraform
Cloud support	Tested for AWS, GCP and Azure	Tested for AWS, GCP and Azure
Execution logs	Difficult to read the logs, but with Terratest hepler utility it makes it readable and can be integrated with CI tools.	Easy to read,neat logging on console. Do not create xml report that can be integrated with CI
Reporting	With the helper utility, UI reports can be generated.	No reporting framework
Test file extension	_xx_test.go	.tftest.hcl

Test Architecture, Introduction to Terratest and Terraform test framework

1. Test Architecture



2. Introduction to Terratest and Terraform test framework

Terratest is a testing library developed by Gruntwork. It has libraries and functions that help with testing terraform, docker, packer, Kubernetes code on AWS, GCP and Azure. Hashicorp announced its test framework to test terraform code on 4th of October 2023.The idea behind doing so was to enable terraform practitioners to know that their configurations will function as expected.

Sample test files and output

```
package test
import (
    "github.com/gruntwork-io/terratest/modules/terraform"
)
func TestTerraformAwsExample(t *testing.T) {
    //Allows parallel execution of tests against same terraform module
    t.Parallel()
    //Terraform source code against which tests needs to be executed
    exampleFolder := test_structure.CopyTerraformFolderToTemp(t, "../", "terraform_ec2_aws_example")
    // configure the options with default retryable errors to handle the most common retryable errors encountered in
    // terraform testing.
    terraformOptions := terraform.WithDefaultRetryableErrors(t, &terraform.Options{
        // The path to where our Terraform code is located
        TerraformDir: exampleFolder,
    })
    //At the end of the test, run `terraform destroy` to clean up any resources that were created
    defer terraform.Destroy(t, terraformOptions)
    //Run `terraform init` and `terraform apply` and fail the test if there are any errors
    terraform.InitAndApply(t, terraformOptions)
    //custom test validations
}
```

```
$ go test -v -timeout 1m
=== RUN   TestTerraformHelloWorldExample
TestTerraformHelloWorldExample 2023-11-29T04:33:46Z retry.go:91: terraform [init -upgrade=false]
TestTerraformHelloWorldExample 2023-11-29T04:33:46Z logger.go:66: Running command terraform with args [init -upgrade=false]
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: [0m[1mInitializing the backend...[0m
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: [0m[1mInitializing provider plugins...[0m
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: [0m[1m[32mTerraform has been successfully initialized![0m[32m[0m
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: [0m[32m
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: You may now begin working with Terraform. Try running "terraform plan" to see
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: any changes that are required for your infrastructure. All Terraform commands
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: should now work.
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: If you ever set or change modules or backend configuration for Terraform,
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: commands will detect it and remind you to do so if necessary.+[0m forget, other
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z retry.go:91: terraform [apply -input=false -auto-approve -lock=false]
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: Running command terraform with args [apply -input=false -auto-approve -lock=false]
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: Changes to Outputs:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:   +[32m+[0m[0m hello_world = "Hello, World!"
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: You can apply this plan to save these new output values to the Terraform
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: state, without changing any real infrastructure.
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66:
TestTerraformHelloWorldExample 2023-11-29T04:33:41Z logger.go:66: [0m[1m[32mApply complete! Resources: 0 added, 0 changed, 0 destroyed.
TestTerraformHelloWorldExample 2023-11-29T04:33:42Z logger.go:66: [0m[1m[32m
TestTerraformHelloWorldExample 2023-11-29T04:33:42Z logger.go:66: Destroy complete! Resources: 0 destroyed.
```

```
variables {
  postfix = "3897"
  location = "West US2"
  sku = "Premium"
}
run "container_registry_sku" {
  command = apply #deploys the infrastructure
  #validations to be performed on the main.tf file
  assert {
    condition     = azurerm_container_registry.acr.sku == "${var.sku}"
    error_message = azurerm_container_registry.acr.sku
  }
}
run "container_registry_name" {
  command = plan #does not deploy the infrastructure
  assert {
    condition     = azurerm_container_registry.acr.name == "acr${var.postfix}"
    error_message = "acr name did not match expected"
  }
}
```

```
$ terraform test -verbose
ec2_example.tftest.hcl... in progress
  run "verifyTag"... pass

ec2_example.tftest.hcl... tearing down
ec2_example.tftest.hcl... pass

Success! 1 passed, 0 failed.
```

Conclusions and Future Work

The Terraform test framework is favored by developers due to its ease of adoption and familiar HCL syntax. Terratest offers more flexibility in handling complex tasks, particularly in complex integration tests and end2end tests. Despite Terratest's ability to handle error and exceptions, it sometimes fails the test and cleans up the spun infrastructure, causing manual intervention and potential cost concerns. For users seeking more control, Terratest is a better option. Reporting with Terratest can be improved by using an additional Terratest utility, which allows users to identify errors and generate UI reports. However, Terraform's reporting is limited to console output and cannot be integrated into a UI reporting tool. Overall, Terraform is better for unit test validations, but Terratest is recommended for complex integration tests and end2end tests. As for future work, Using the same set of experiments, the CI/CD support can be tested. Also, as part of Terratest a deeper analysis can be made as it offers other services like testing Kubernetes, it was tested briefly in the above experiments, yet an extensive study can be performed. Along with this, Terratest also offers testing Docker and Packer that was not evaluated in these experiments. One of the other interesting topics can be studying if these test frameworks detect configuration drift and does it provide any feature of analysing the same.

QR Code for Recording