Samuel Kihahu

DevOps Engineer, Tala

HashiCorp

Testing

Infrastructure As Code On 127.0.0.1

Agenda

- Why test on localhost?
- What tests to write?
- What tools to use.
- How to use the tools.
- What lessons did we learn?

Why

Test on localhost?

- Speed - quick iteration while doing development.
- Cost - no fees involved.
- Convenience - you are familiar with your environment setup.
- Failsafe - validate code before running on actual cloud platform (break with no fear)

What

Tools?

- Terratest
  - Previous experience using terragrunt.
  - Support for other tools e.g kubernetes, helm, docker, packer e.t.c
- Localstack
  - Our application run on AWS cloud.

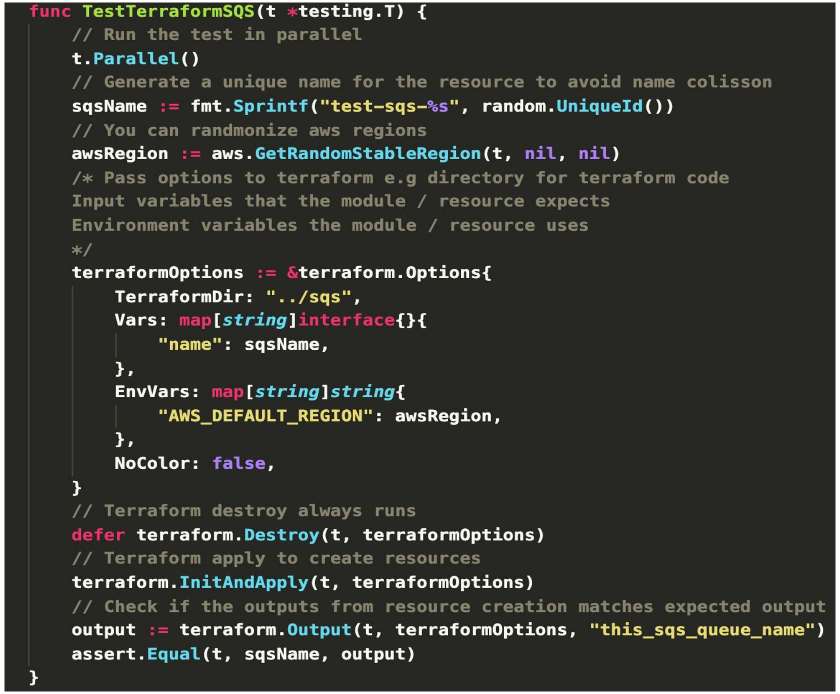
What

Tests?

- Unit tests
  - Individual resources in isolation.
- Integration tests
  - More than one resources that have dependencies.

How

to write terratest tests.



```go
func TestTerraformSQS(t *testing.T) {
    // Run the test in parallel
    t.Parallel()
    // Generate a unique name for the resource to avoid name colisson
    sqsName := fmt.Sprintf("test-sqs-%s", random.UniqueId())
    // You can randmonize aws regions
    awsRegion := aws.GetRandomStableRegion(t, nil, nil)
    /* Pass options to terraform e.g directory for terraform code
    Input variables that the module / resource expects
    Environment variables the module / resource uses
    */
    terraformOptions := &terraform.Options{
        TerraformDir: "../sqs",
        Vars: map[string]interface{}{
            "name": sqsName,
        },
        EnvVars: map[string]string{
            "AWS_DEFAULT_REGION": awsRegion,
        },
        NoColor: false,
    }
    // Terraform destroy always runs
    defer terraform.Destroy(t, terraformOptions)
    // Terraform apply to create resources
    terraform.InitAndApply(t, terraformOptions)
    // Check if the outputs from resource creation matches expected output
    output := terraform.Output(t, terraformOptions, "this_sqs_queue_name")
    assert.Equal(t, sqsName, output)
}
```

Components of a test.

Tests can run in parallel.

Function name must start as ***Test***Xxx, X is capitalised.

Test filename should end with _***test.go.***

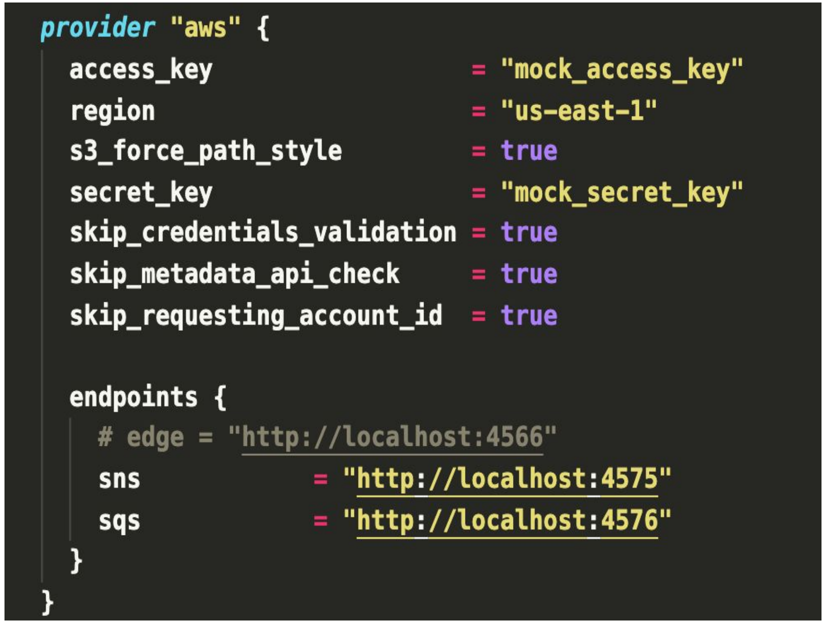
Tags can be used to manage tests e.g avoid load issues.

Randomize resource naming.

go test -run TestTerraformSNS

How

to configure localstack.



```
provider "aws" {
  access_key                  = "mock_access_key"
  region                      = "us-east-1"
  s3_force_path_style         = true
  secret_key                  = "mock_secret_key"
  skip_credentials_validation = true
  skip_metadata_api_check     = true
  skip_requesting_account_id  = true

  endpoints {
    # edge = "http://localhost:4566"
    sns             = "http://localhost:4575"
    sqs             = "http://localhost:4576"
  }
}
```

Terraform provider configuration.

- Configure alternate endpoint for localstack It’s possible to configure multiple providers using aliases.
- Edge service support open issue.

What

Lessons?

- Limited scope
  - localstack is currently limited to mocking AWS cloud
- Terraform provider provides “best effort” support
  - some features might not be immediately available e.g recent addition(0.11.0) of edge service
- Limited functionality
  - Available resources on localstack
  - localstack available resources are limited and you might need to eventually test on an actual cloud or consider upgrading to localstack pro
- CI integration
  - As you write more tests, consider using tags, short option, makefile to trigger different tests.

What

Lessons?

- Combining tests on localstack and actual cloud
  - Use provider aliasing to determine where tests run.
- Long running tests
  - Default timeout for tests is 10m, if your tests take longer consider extending time -timeout 30m
- Error handling in terratest
  - All functions have an error returning variant, if you use them, you need to handle error cases
- Unique naming for resources
  - This is to avoid name collision for resources
- Caching
  - A default for go>= 1.10, consider using *-count=1* to disable caching.

Recap

- Testing locally is cost effective, fast, convenient and builds confidence.
- There are limitation related to cloud provider and functionality support.
- Provider aliasing can be used to determine whether to run tests locally or in the cloud.

References

- gruntwork.io terratest talk
- Terraform alternate provider config
- Terraform provider instances
- github.com/kihahu/terraform-modules