Samuel Kihahu

DevOps Engineer, Tala

Tala Locations

Mexico

Philippines

India

Kenya

Testing

Infrastructure As Code

Agenda

- Why test Infrastructure?
- What tests to write?
- Where to run the tests?
- What tools to use?
- What lessons have we learnt?

Why

Test Infrastructure?

- Confidence
  - Similar environment is created every time.
- Speed
  - Move fast as tests protect you from known and potential issues.
- Feedback
  - If code is difficult tests, question the implementation.

Why

Test Infrastructure?

- Documentation
  - Well written tests help understanding of the code.
- Experiment with new features / architecture configurations.
  - E.g multi-cloud setup

- Documentation
  - Well written tests help understanding of the code.

What

Tests?

- Static analysis
  - linters or style checkers
- Unit tests
  - Individual resources in isolation.
- Integration tests
  - More than one resources that have dependencies.

What

Tests?

- End to end tests
  - All the infrastructure together with applications
- Blue / Green deployments
- Monitoring in production
  - Using performance metrics to gain visibility into your infrastructure

Where

To run tests?

- Static analysis
- Unit tests
- *Integration tests*

Localhost

Where

To run tests?

- *Integration tests*
- End to end tests
- Blue / Green deployments
- Monitoring in production

Production-*like*
(dev/qa/stage)

Where

To run tests?

- Production
  - Blue / Green deployments
  - Monitoring in production

Production

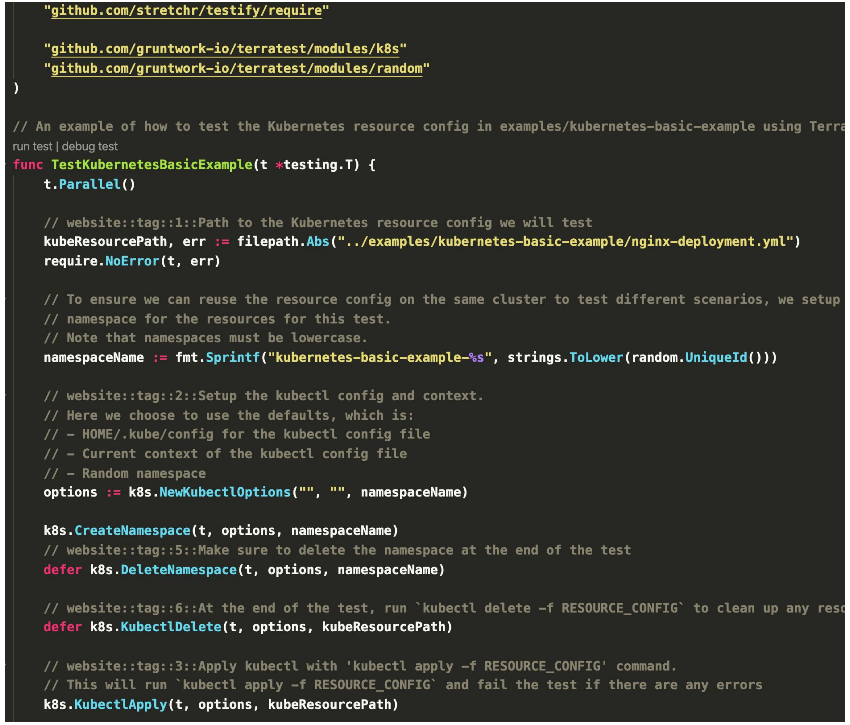
What

Tools?

- Terratest
  - Opensource
  - Previous experience using terragrunt.
  - Support for other tools e.g kubernetes, helm, docker, packer e.t.c
- KIND / Minikube / k3s
  - Local kubernetes instance
- Localhost cloud emulators
  - Localstack, gcloud emulators, Azure abstractions
- Actual cloud account

How

to write terratest tests.



```
    "github.com/stretchr/testify/require"

    "github.com/gruntwork-io/terratest/modules/k8s"
    "github.com/gruntwork-io/terratest/modules/random"
)

// An example of how to test the Kubernetes resource config in examples/kubernetes-basic-example using Terra
run test | debug test
func TestKubernetesBasicExample(t *testing.T) {
    t.Parallel()

    // website::tag::1::Path to the Kubernetes resource config we will test
    kubeResourcePath, err := filepath.Abs("../examples/kubernetes-basic-example/nginx-deployment.yml")
    require.NoError(t, err)

    // To ensure we can reuse the resource config on the same cluster to test different scenarios, we setup
    // namespace for the resources for this test.
    // Note that namespaces must be lowercase.
    namespaceName := fmt.Sprintf("kubernetes-basic-example-%s", strings.ToLower(random.UniqueId()))

    // website::tag::2::Setup the kubectl config and context.
    // Here we choose to use the defaults, which is:
    // - HOME/.kube/config for the kubectl config file
    // - Current context of the kubectl config file
    // - Random namespace
    options := k8s.NewKubectlOptions("", "", namespaceName)

    k8s.CreateNamespace(t, options, namespaceName)
    // website::tag::5::Make sure to delete the namespace at the end of the test
    defer k8s.DeleteNamespace(t, options, namespaceName)

    // website::tag::6::At the end of the test, run `kubectl delete -f RESOURCE_CONFIG` to clean up any reso
    defer k8s.KubectlDelete(t, options, kubeResourcePath)

    // website::tag::3::Apply kubectl with 'kubectl apply -f RESOURCE_CONFIG' command.
    // This will run `kubectl apply -f RESOURCE_CONFIG` and fail the test if there are any errors
    k8s.KubectlApply(t, options, kubeResourcePath)
```

Components of a test.

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

Tests can run in parallel.

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

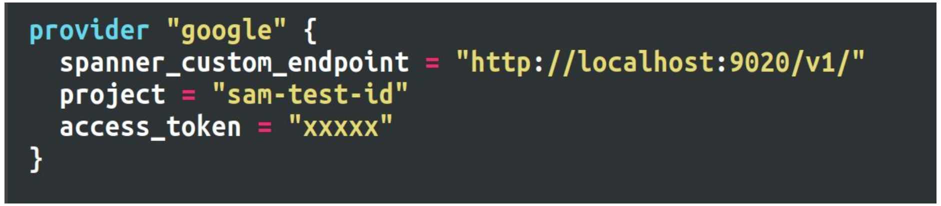
Tags can be used to manage tests e.g avoid load issues.

Randomize resource naming to avoid conflicts.

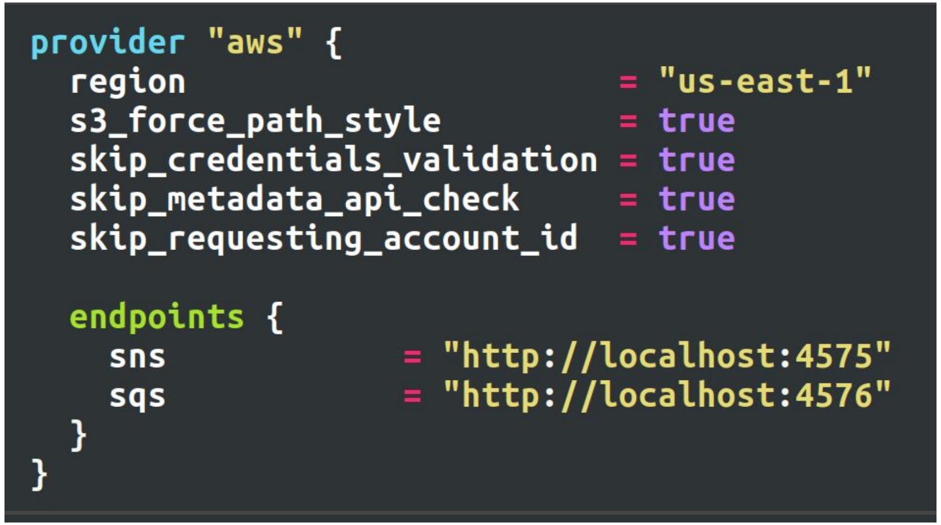
go test -v --tags kubernetes -run TestKubernetesBasicExample

How

to write terratest tests.



```
provider "google" {
  spanner_custom_endpoint = "http://localhost:9020/v1/"
  project = "sam-test-id"
  access_token = "xxxxx"
}
```



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

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

Environment Configuration.

Custom endpoints for particular service

Best effort support for custom endpoints.

Provider alias is possible opening possibility of doing hybrid testing.

What

Lessons?

- Choose testing environment based on the function of the infrastructure
  - Consider complexity, resource availability e.t.c
- Cost is a factor
  - Consider your budget
- CI integration
  - As you write more tests, consider using tags, short option, makefile to trigger different tests.
  - On busy mono repositories consider various test strategies e.g run tests via git hooks before commits.

What

Lessons?

- **Localhost Cloud Emulators**
  - Limited scope
    - e.g localstack is currently limited to mocking AWS cloud, gcloud and Azure emulators also don’t support all the provider functionality.

What

Lessons?

- **Terratest Library**
  - 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 for local testing.
- A hybrid approach to testing is possible by using provider aliasing.
- Production and production-like testing in the cloud is ultimately important to validate your infrastructure.

References

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