

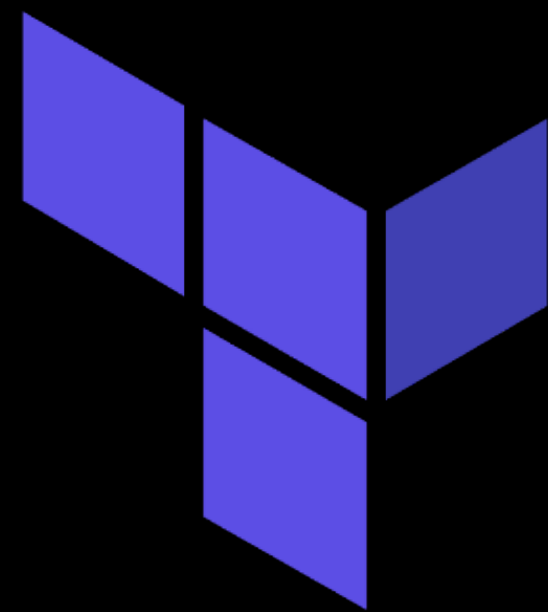
laC Testautomatisierung

Am Beispiel Terratest

Simon Bein
meshcloud GmbH

Weiße Folie = Konzept

Schwarze Folie = Praxisbeispiel



Abgrenzung IaC Tests für diesen Talk

IaC Tests

- Infrakomponente(n) angelegt?
- Konfiguration der Infrakomponente(n) korrekt?
- Kommunikation/Isolation der Infrakomponente(n) korrekt?

Out of Scope

- Applikation deployed?
- Konfiguration der Application korrekt?
- Kommunikation zwischen Applikation(en), Datenbank, ...

**Braucht man überhaupt
Tests bei IaC?**

It depends..

Wann IaC Tests?

Komponenten x Umgebungen x Rollouts*

* mit Einfluss auf Infrastruktur

Der Dev-Effekt

Anzahl Rollouts um 100-fache höher während Einführung neuer Komponenten

⇒ Allein hierfür kann sich das Schreiben der Tests lohnen

Welche IaC Tests gibt es?

Arten von IaC Tests

1. Statische & Lokale Checks
2. Apply & Destroy Tests
3. Imitations Tests

1. Statische & Lokale Checks

Idee: Konfigurationen und Dependencies vor Ausführung analysieren

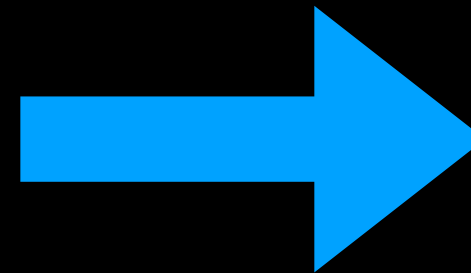
Ziel: Schnelles Aufdecken von statischen Fehlern

Beispiele: Vollständigkeitsprüfung, Feldvalidierung, Referenzanalyse

Vollständigkeit & Feld Validierung

```
resource google_compute_instance test {  
  name          = "hello-terratest"  
  
  boot_disk {  
    initialize_params {  
      image = "ubuntu-1804-lts"  
    }  
  }  
  
  network_interface {  
    network = "default"  
    access_config {}  
  }  
}
```

simple_vm.tf



```
> terraform validate
```

Error: Missing required argument

```
on simple_vm.tf line 8, in resource  
"google_compute_instance" "test":  
  8: resource google_compute_instance  
test {
```

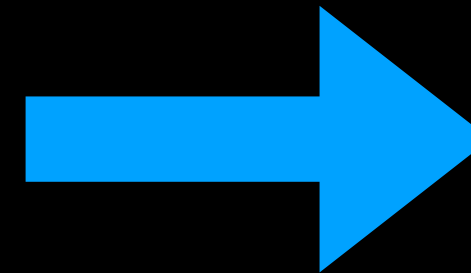
The argument "machine_type" is
required, but no definition was found.

Autovervollständigung mittels Language Server

```
resource google_compute_instance test {  
  name = "hello-terratest"  
  machine  
    machine_type (Required) string  
  boot_disk {  
    initialize_params {  
      image = "ubuntu-1804-lts"  
    }  
  }  
  
  network_interface {  
    network = "default"  
    access_config {}  
  }  
}
```

Referenz Validierung

```
resource google_compute_forwarding_rule test {  
  name    = "hello-terratest"  
  target = google_compute_target_pool.test.self_link  
}
```



```
> terraform validate
```

Error: Reference to undeclared resource

```
    on targetpool.tf line 4, in resource  
    "google_compute_forwarding_rule"  
    "test":  
      4:   target =  
    google_compute_target_pool.test.self_li  
nk
```

A managed resource
"google_compute_target_pool" "test" has
not been declared
in the root module.

targetpool.tf

2. Apply & Destroy Tests

Idee: Infrastruktur kurz ausrollen und direkt im Anschluss zerstören

Ziel: Prüfen von dynamischen Feldern und dependencies

Einfacher Apply & Destroy Test

```
package apply_test

import (
    "testing"

    "github.com/gruntwork-io/terratest/modules/terraform"
)

func TestGCP(t *testing.T) {
    t.Parallel()

    tfOptions := &terraform.Options{
        TerraformDir: "./terraform/apply_test",
    }

    terraform.InitAndApply(t, tfOptions)
    terraform.Destroy(t, tfOptions)
}
```

apply_test.go

2. Apply & Destroy Tests

- Dauer des Tests im Verhältnis zum durchschnittlichen Nutzen lang
- + Hilft bei komplexen Ressourcen mit vielen Abhängigkeiten
- + Testszenario einfach übertragbar auf andere Ressourcen

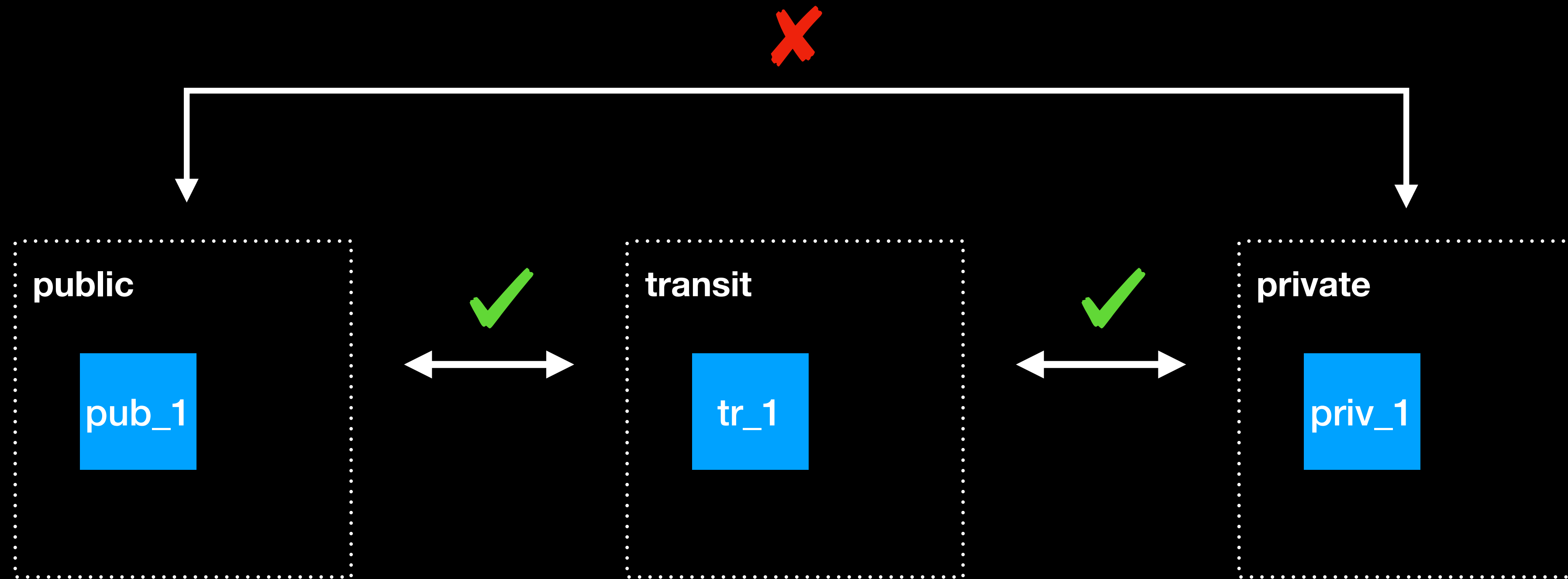
3. Imitationstests

Idee: Benötigte Operationen der Applikation imitieren

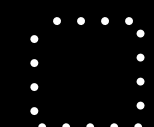
Ziel: Prüfen ob Infrastruktur korrekt konfiguriert ist

Beispiele: Netzwerktest, I/O Tests

Netzwerktest - Beispiel



 = Virtual Machine

 = Subnet

Netzwerktest - Beispiel

...

```
terraform.InitAndApply(t, tfOptions)
```

```
pubVM := gcp.FetchInstance(t, gcpProject, instanceName)
```

```
keyPair := ssh.GenerateRSAKeyPair(t, 2048)
```

```
pubVM.AddSshKey(t, "terratest", keyPair.PublicKey)
```

```
host := ssh.Host{
```

```
    Hostname:    terraform.Output(t, tfOptions, "public_ip"),
```

```
    SshUserName: "terratest",
```

```
    SshKeyPair:  keyPair,
```

```
}
```

```
// we have to wait for AddSshkey to finish. Don't do this in production!
```

```
time.Sleep(3 * time.Second)
```

```
// Check that we can access transit
```

```
_, err := ssh.CheckSshCommandE(t, host, "ping -c 3 10.0.1.2")
```

```
if err != nil {
```

```
    t.Fatalf("Could not connect to transit from public: '%v'", err)
```

```
}
```

```
// Check that we cannot access private
```

```
_, err = ssh.CheckSshCommandE(t, host, "ping -c 3 10.0.2.2")
```

```
if err == nil {
```

```
    t.Fatalf("Could connect to private from public, which is not allowed")
```

```
}
```

...

```
import (
```

```
    ...
```

```
    "github.com/gruntwork-io/terratest/modules/gcp"
```

```
    "github.com/gruntwork-io/terratest/modules/ssh"
```

```
    "github.com/gruntwork-io/terratest/modules/terraform"
```

```
)
```

3. Imitationstests

- Höhere Komplexität, teilweise “hacky”
- + Simulation von realen Szenarien auf Infrastrukturebene
- + Einfach Integration in Apply & Destroy Tests

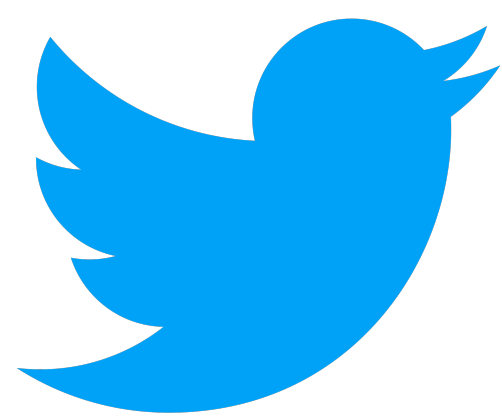
Gut geeignete Cases für IaC Tests

- Netzwerkstrukturen
- VM Configs
- Cloudspezifika
- Kubernetes Rollouts

Wichtig bei der Auswahl des IaC Test toolings

- Gute Integration in den IaC provider selbst
- Integration mit low level tooling (http client, ssh, openssl, ...) möglich
- Re-try, Repeat Intervalle einstellbar

Kontakt Daten



@SimonTheLeg



sbein@meshcloud.io

Frage-Runde