InfraCoding with Terraform: Writing Tests for Infrastructure-as-Code

DevOpsPro 2019

Peter Souter
Sr. Technical Account Manager - HashiCorp

Introductions

Who is this bloke?

Sr. Technical Account Manager
Peter Souter

Team
Customer Success

Based in
London, UK

Started at HashiCorp...
February 2018

What are we here to talk about?

TestCon Europe 2019

TestCon Europe 2019

Infrastructure-as-code

Treating the tooling that manages your infrastructure with the same way you would treat any other code.

Qui bono?

Who benefits?

1. Easy to regenerate from scratch
2. Code review and collaboration
3. Easier to conceptualise as a code model
4. Auditing and policies
5. Iteratively improve over time
6. Modularise, reuse and hide complexity
7. Testing

Qui bono?

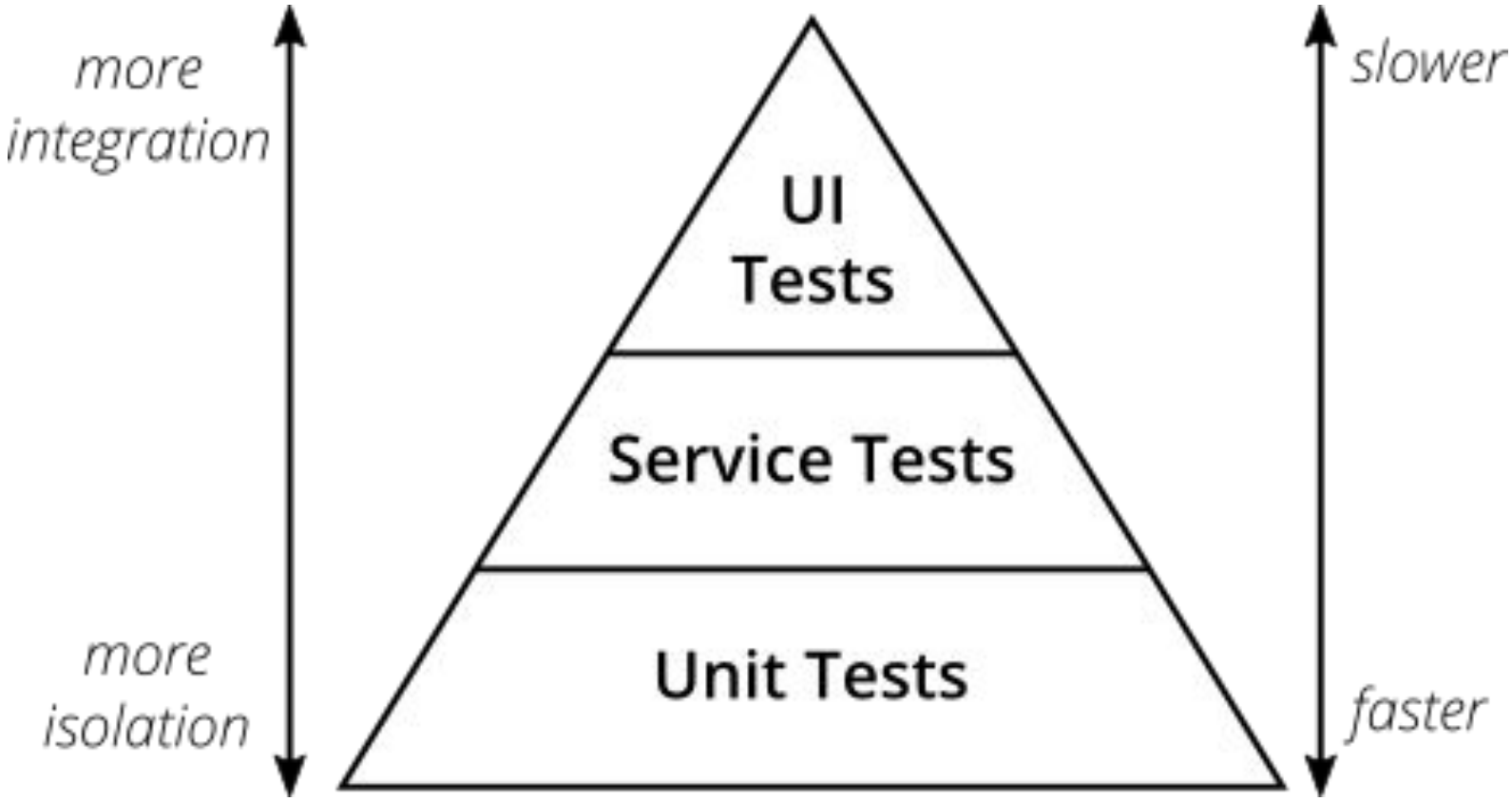
Who benefits?

1. Easy to regenerate from scratch
2. Code review and collaboration
3. Easier to conceptualise as a code model
4. Auditing and policies
5. Iteratively improve over time
6. Modularise, reuse and hide complexity
7. Testing

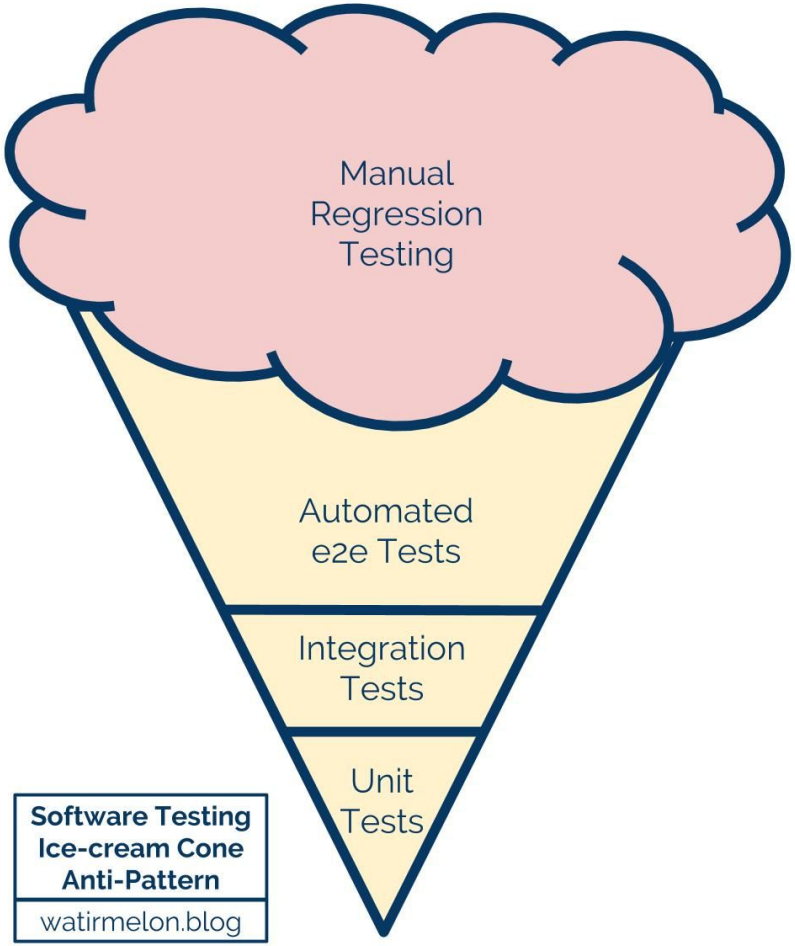
“Why should we test our infrastructure as code anyways?”

“The concern of velocity loss of added testing is very much worthwhile in my experience. It’s why we have a known, stable configuration and deployment codebase and can move on to more relevant business problems. In their absence, I’ve without exception had to firefight tooling, silent regressions, and human error burning out teams and reducing confidence in automation from across teams. **Infrastructure as code without testing to me is self-sabotage”**

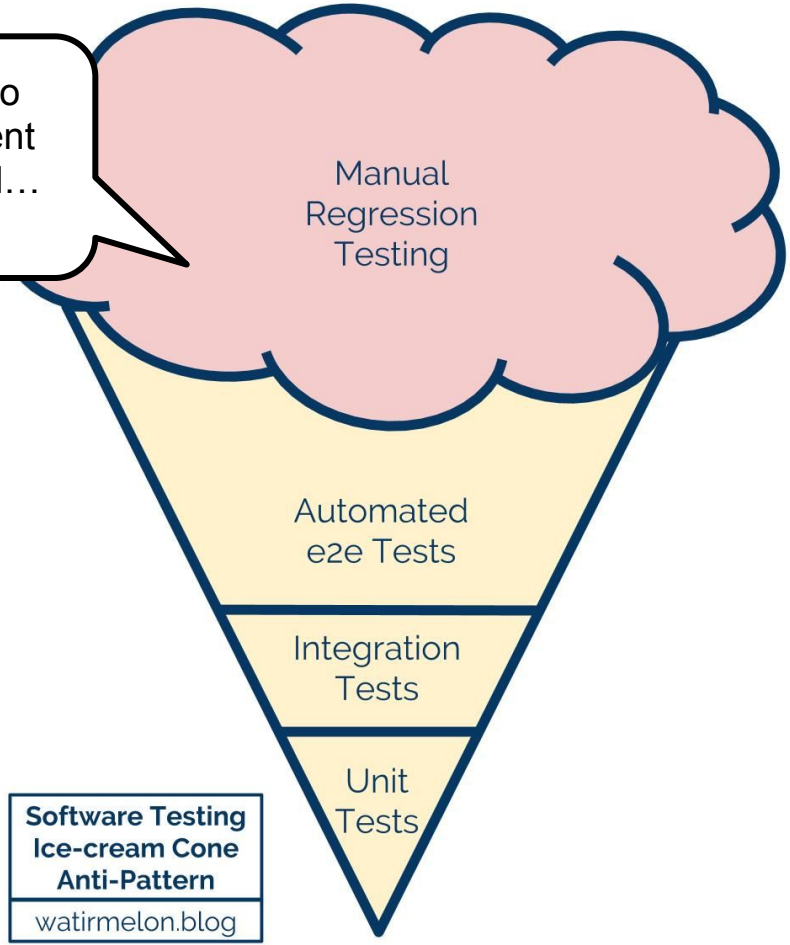
- **Devon Kim, @djk29a, HashiConf 2019 Slack**

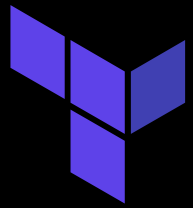


https://martinfowler.com/articles/practical-test-pyramid.html


Manual
Regression
Testing
Automated
e2e Tests
Integration
Tests
Unit
Tests
Software Testing
Ice-cream Cone
Anti-Pattern
watirmelon.blog

The terraform seemed to generate the environment correctly last time I tried… meh, ship it!



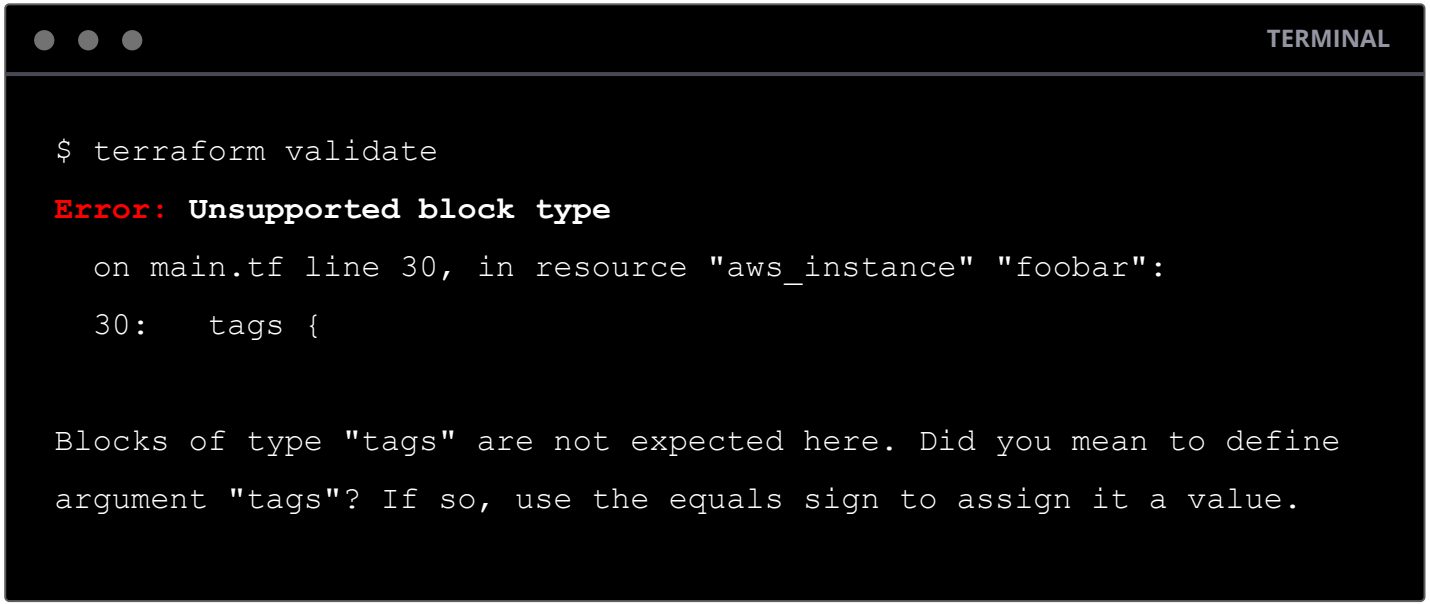


HashiCorp

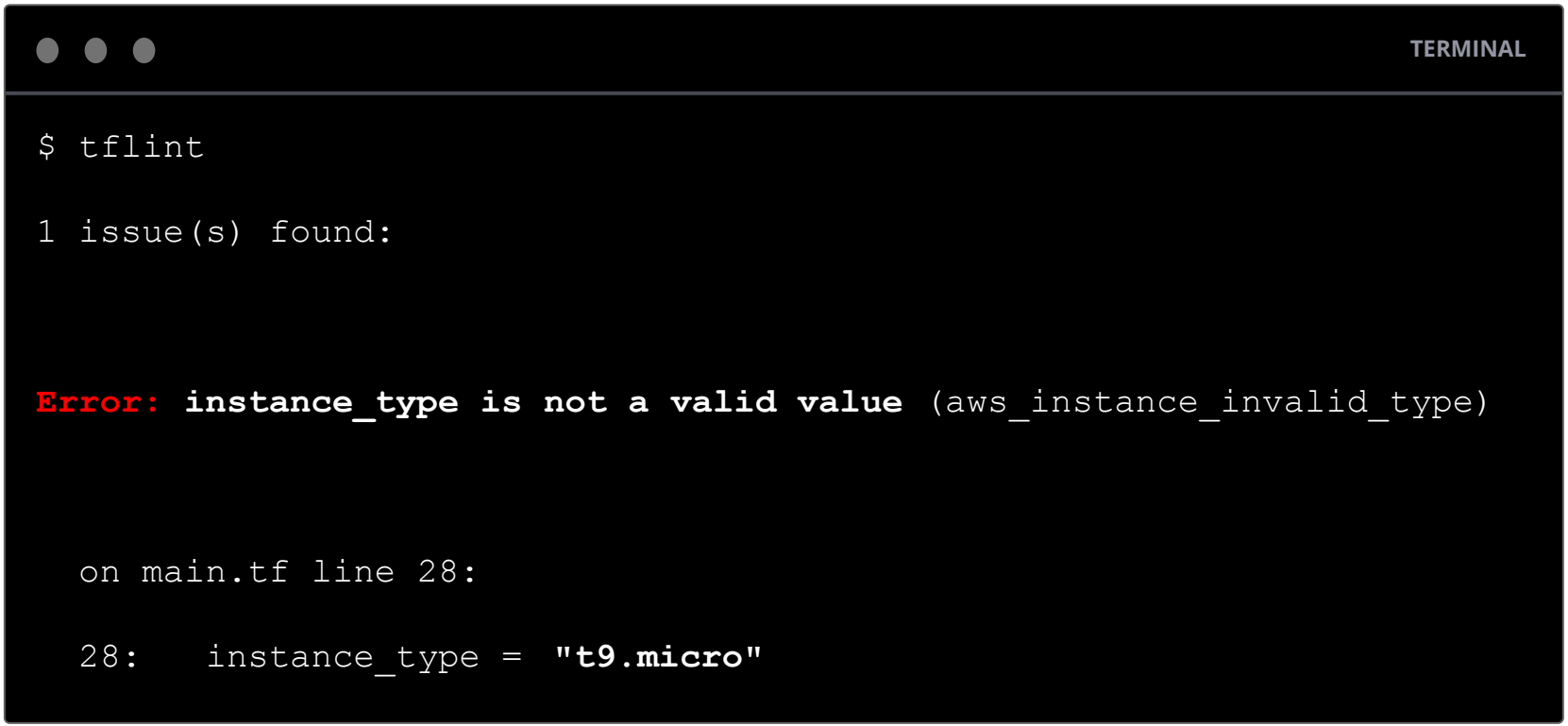
Terraform

The most simple “tests”: linting and compilation checks

terraform validate



tflint



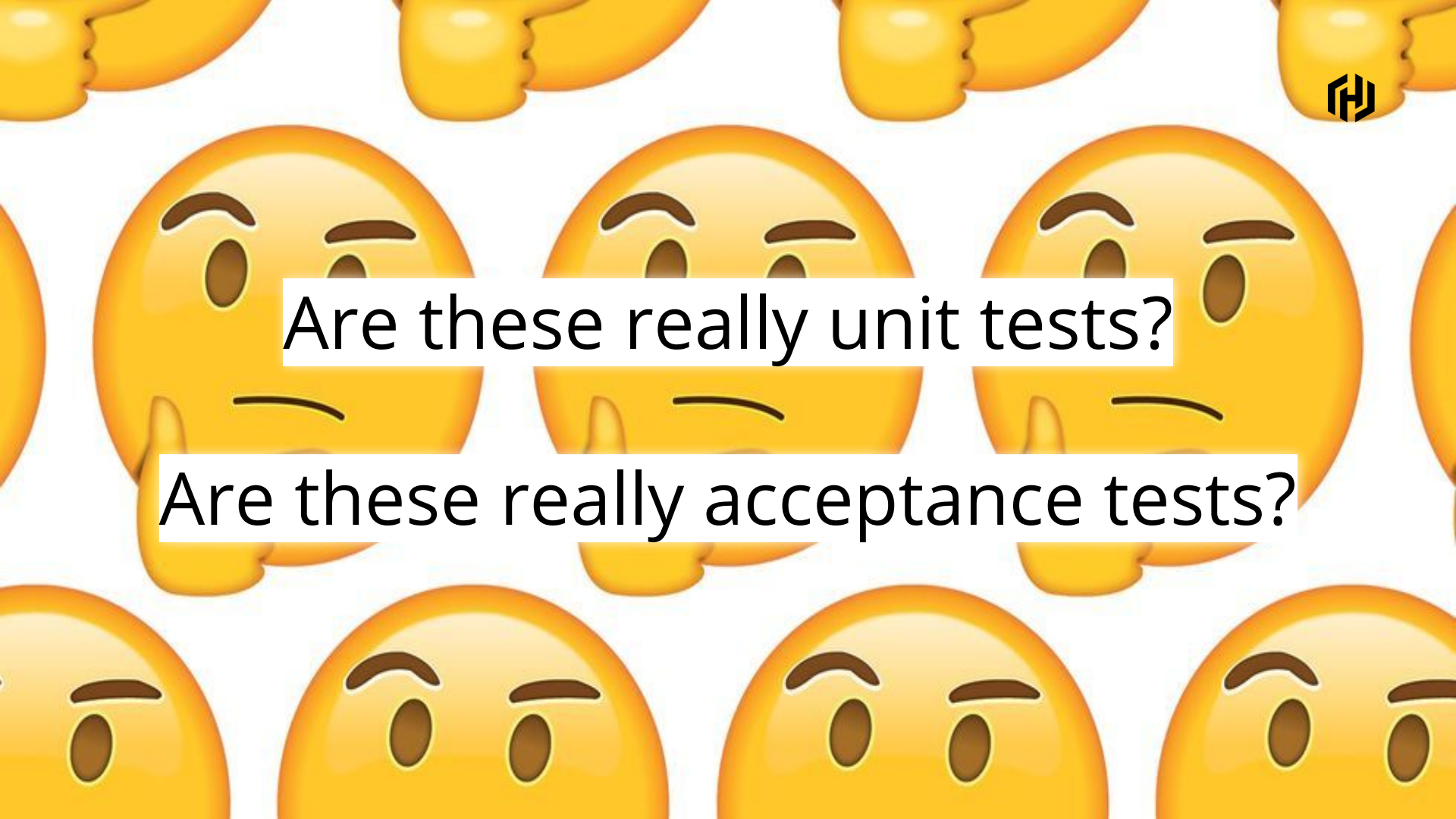
https://github.com/wata727/tflint

Ok, so let's start with unit tests for our Terraform code

Acceptance tests make sure that you're building the **right thing**

Unit tests make sure that you're building the **thing right**

https://stackoverflow.com/questions/4139095/unit-tests-vs-acceptance-tests


Are these really unit tests?
Are these really acceptance tests?

Some disagreement if these are “real” unit tests…

The only “real” Terraform unit test...


NOGA
LIU
BAJ

So let's use some different terms to be clearer

CODE EDITOR

```
resource "aws_instance" "foobar" {
  ami           = "ami-0fab23d0250b9a47e"
  instance_type = "t1.micro"

  tags = {
    Name = "foobar"
    DemoDate = "14-October-2019"
  }
}
```

Terraform Code

CODE EDITOR


```
{
  "version": 4,
  "terraform_version": "0.12.6",
  "serial": 6,
  "lineage": "4f59e0b7-698d-a0b9-50f6-3c31b5090200",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "foobar",
      "provider": "provider.aws"
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "ami": "ami-03ef731cc103c9f09",
            "associate_public_ip_address": true,
```


Terraform State

CODE EDITOR

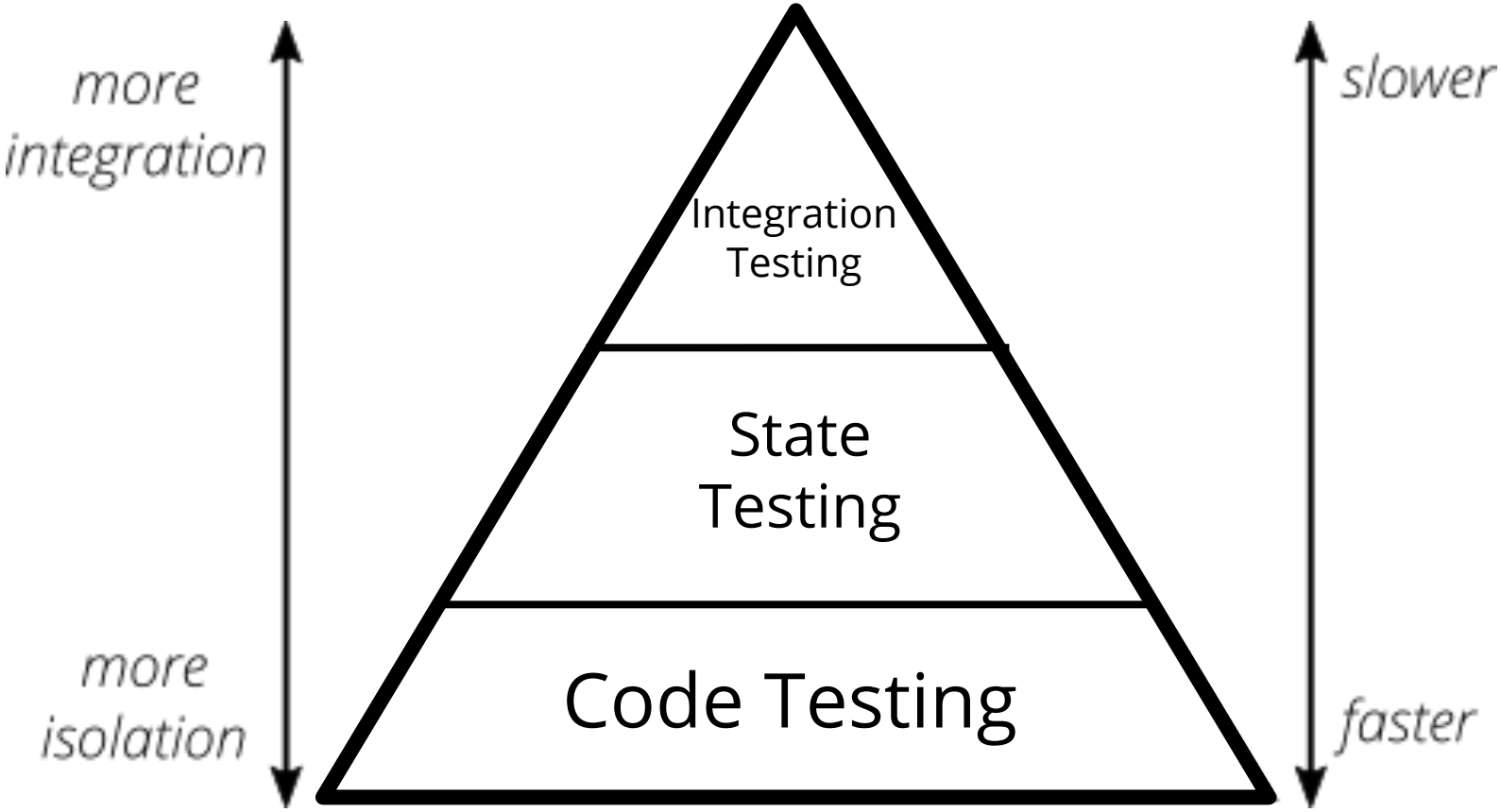
```
resource "aws_instance" "foobar" {
  ami           = "ami-0fab23d0250b9a47e"
  instance_type = "t1.micro"

  tags = {
    Name = "foobar"
    DemoDate = "14-October-2019"
  }
}
```

Code testing

CODE EDITOR


```
{
  "version": 4,
  "terraform_version": "0.12.6",
  "serial": 6,
  "lineage": "4f59e0b7-698d-a0b9-50f6-3c31b5090200",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "foobar",
      "provider": "provider.aws"
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "ami": "ami-03ef731cc103c9f09",
            "associate_public_ip_address": true,
```


State testing

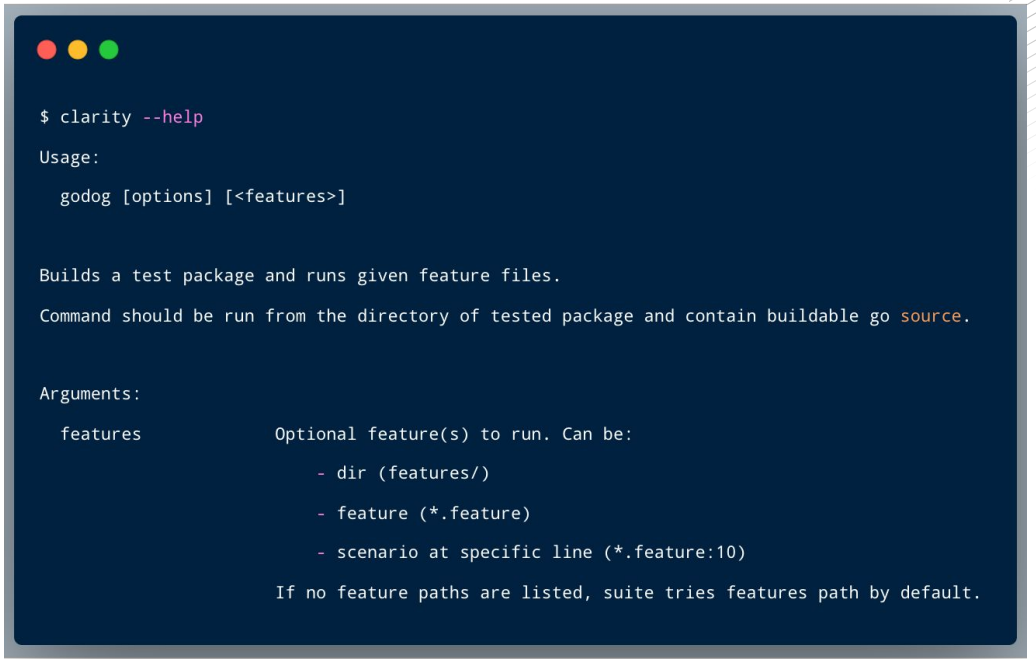

more
integration
more
isolation
Integration
Testing
State
Testing
Code Testing
slower
faster

Code Testing

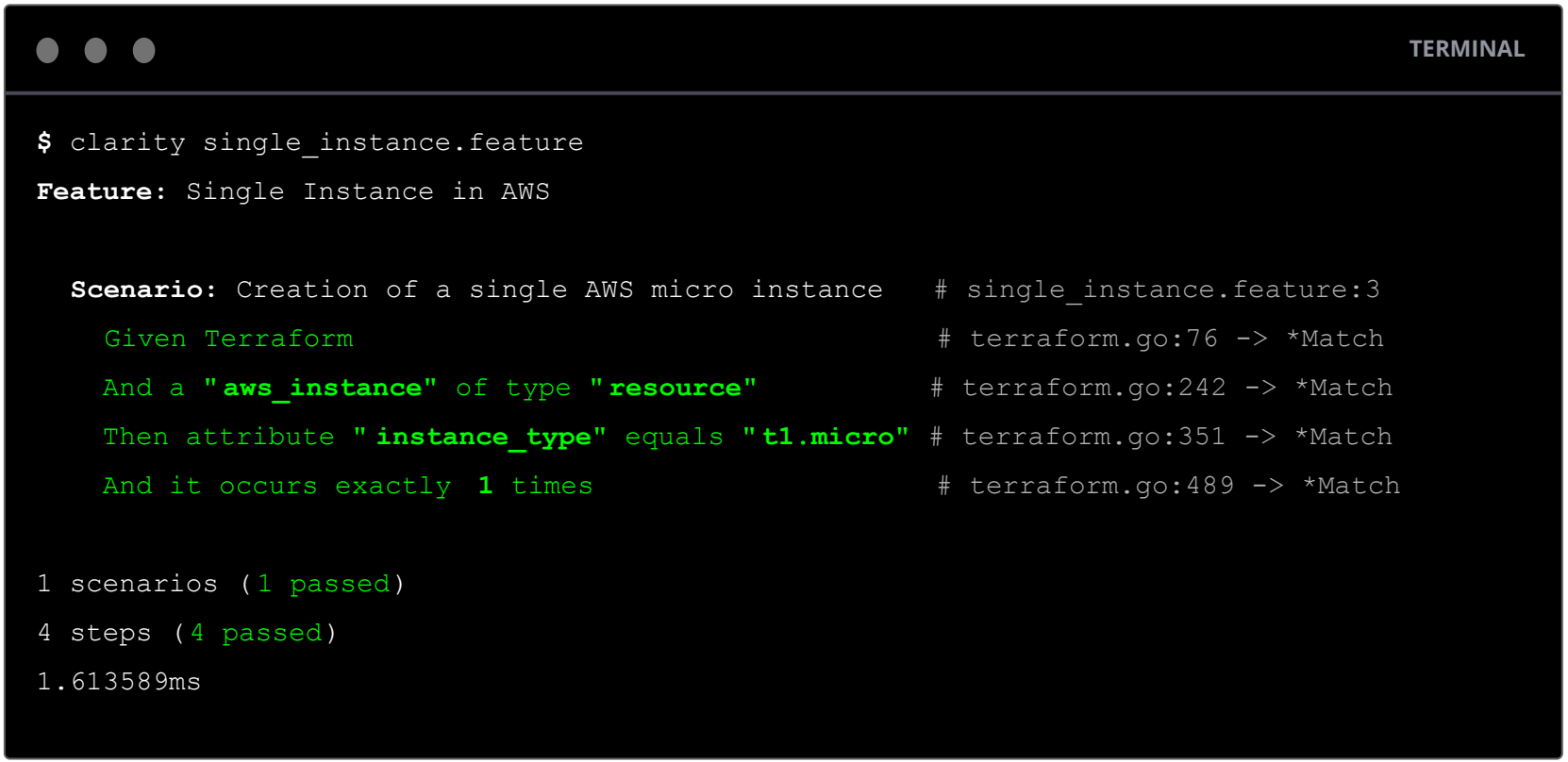
clarity

https://github.com/xchapter7x/clarity/tree

- Self-contained binary
- Gherkin style features
- Behaviour-Driven Development
- Parses HCL for running its checks
- Provides its own Terraform specific matchers



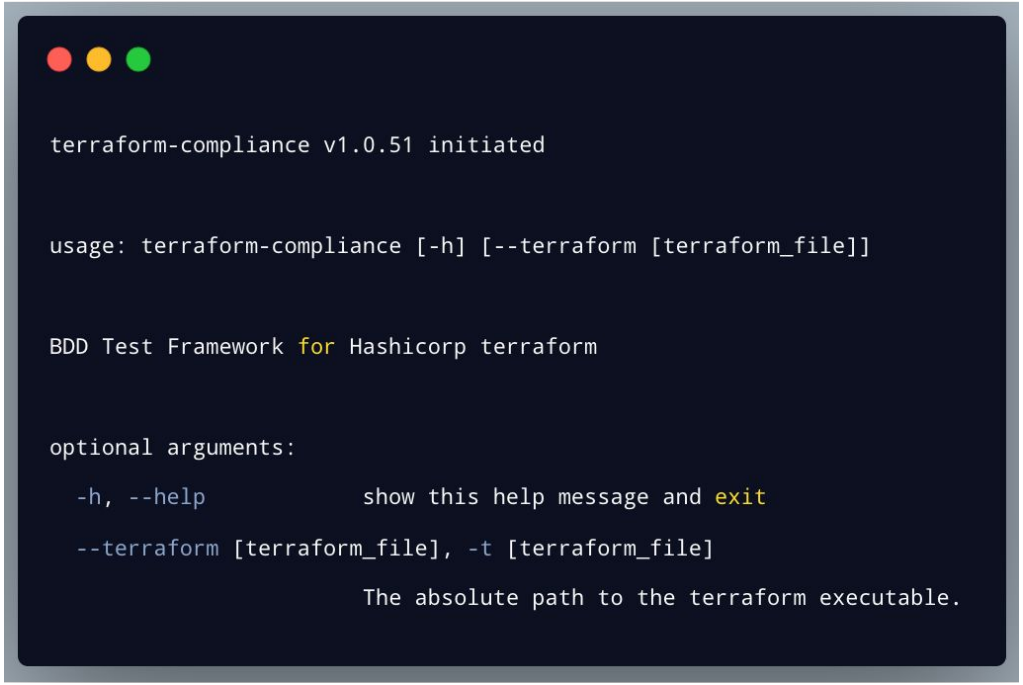
clarity example



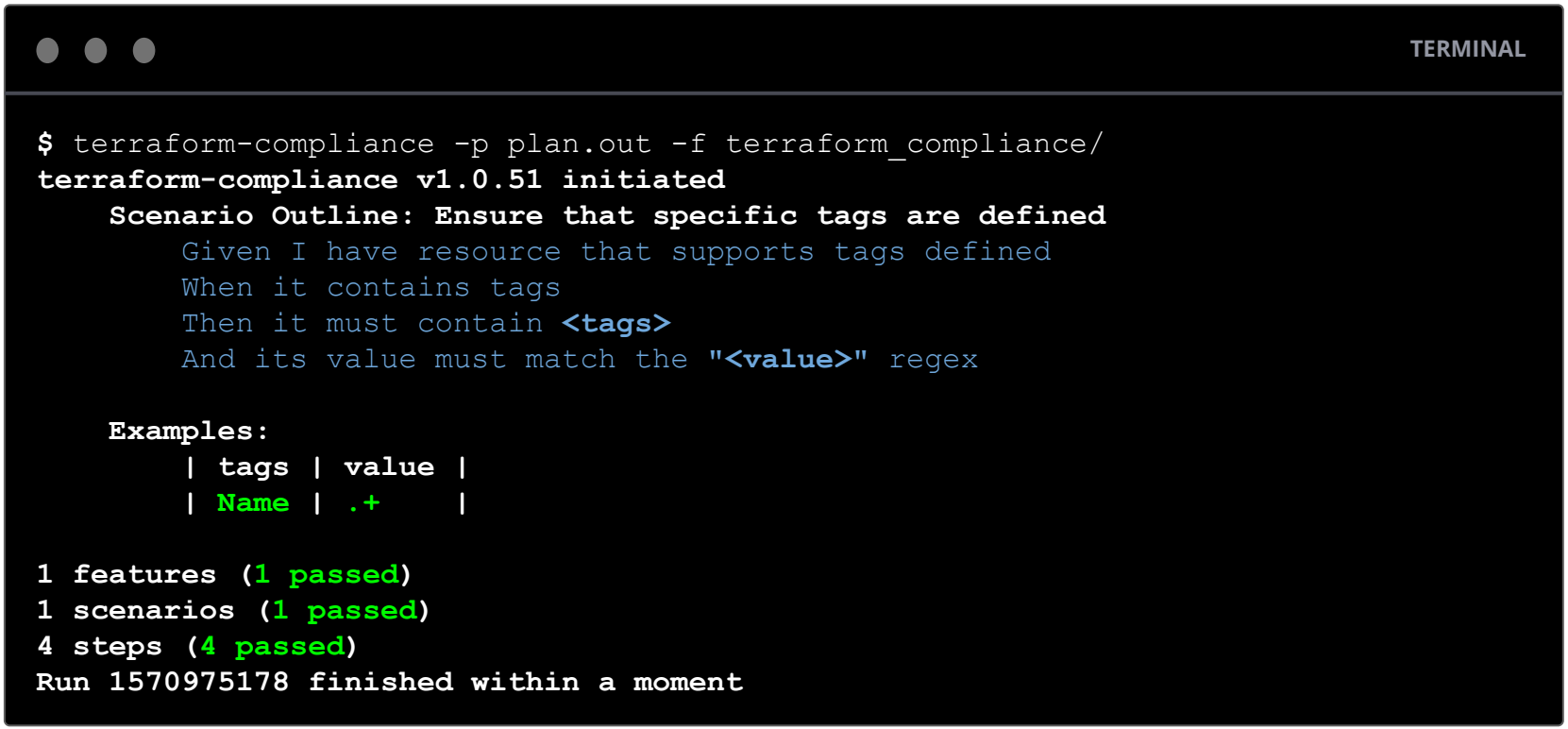
terraform-compliance

https://github.com/eerkunt/terraform-compliance

- Python CLI application
- Gherkin style features
- Behaviour-Driven Development
- Parses **plan outputs** for it's checks
- Provides its own Terraform specific matchers



terraform-compliance example



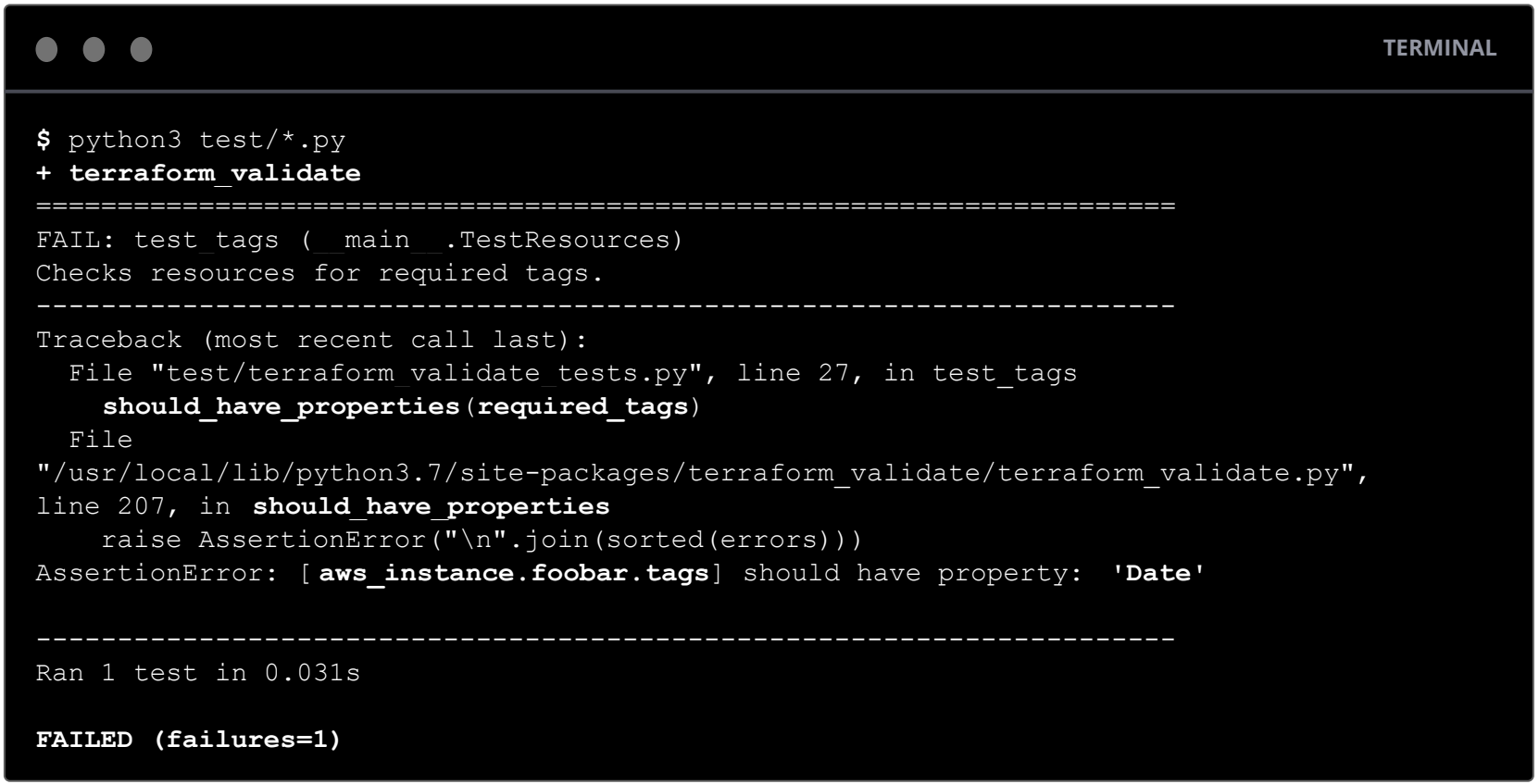
terraform_validate

https://github.com/elmundio87/terraform_validate

- Written as Python
- Standard unit testing
- Parses **HCL** for it's checks
- Provides it's own Terraform specific matchers

```
def test_tags(self):
    """Checks resources for required tags."""
    tagged_resources = [
        'aws_instance'
    ]
    required_tags = ['Name']
    self.validator.error_if_property_missing()
    self.validator.resources(tagged_resources).property('tags'). \
        should_have_properties(required_tags)
```

terraform_validate example



Don’t just write tautological unit tests, test the edge cases

A unit tests is for the logic you have written, not to test the language

Red

Green

Refactor

Live Demo Warning!

Code Testing

Pros

Fast to run

Easy to write

No environment or credentials required

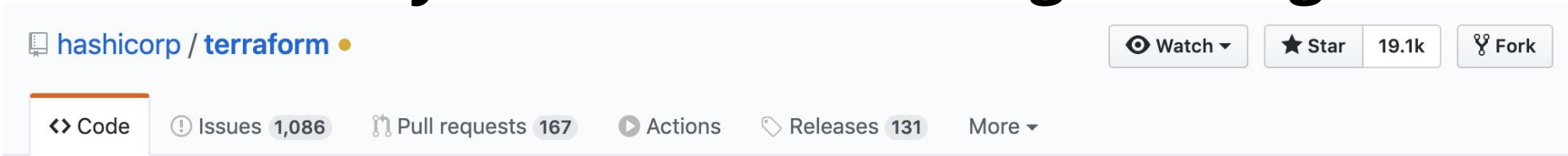
Cons

Doesn’t test actual outcome

The Future: Core Unit Testing?

https://github.com/hashicorp/terraform/issues/21628

Possibly... but we have a big backlog!



State Testing

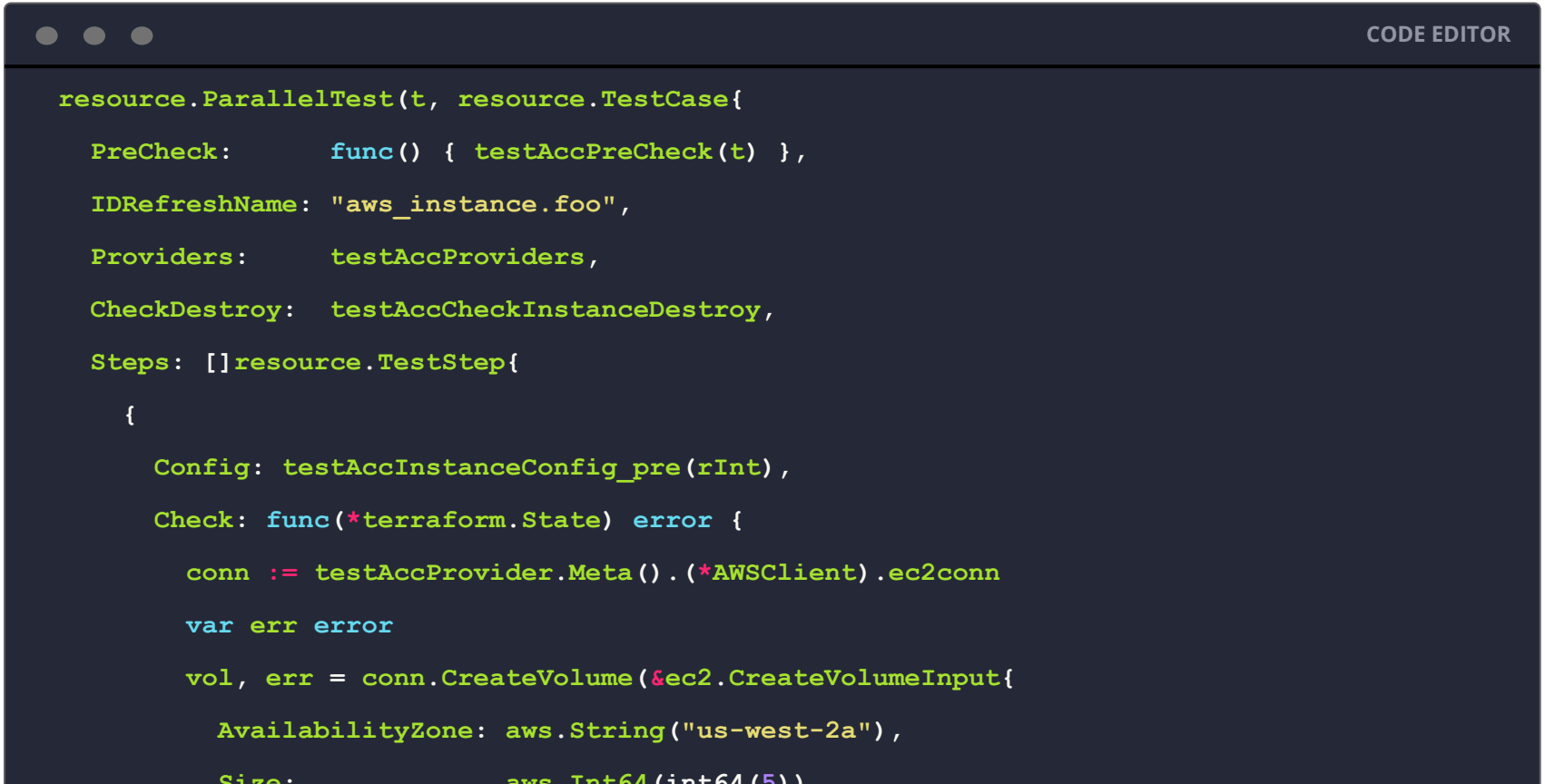
Terraform Testing SDK

http://github.com/hashicorp/terraform-plugin-sdk/helper/acctest

- Used for acceptance tests for Terraform providers
- Written in Golang
- Helper methods for validating state
- Dev focused

```go
func TestNamespace_basic(t *testing.T) {
  namespacePath := acctest.RandomWithPrefix("test-namespace")
  childPath := acctest.RandomWithPrefix("child-namespace")
  resource.Test(t, resource.TestCase{
    PreCheck:     func() { testAccPreCheck(t) },
    Providers:    testProviders,
    CheckDestroy: testNamespaceDestroy(namespacePath),
    Steps: []resource.TestStep{
      {
        Config: testNamespaceConfig(namespacePath),
        Check:  testNamespaceCheckAttrs(),
      }
```

Terraform Testing SDK example

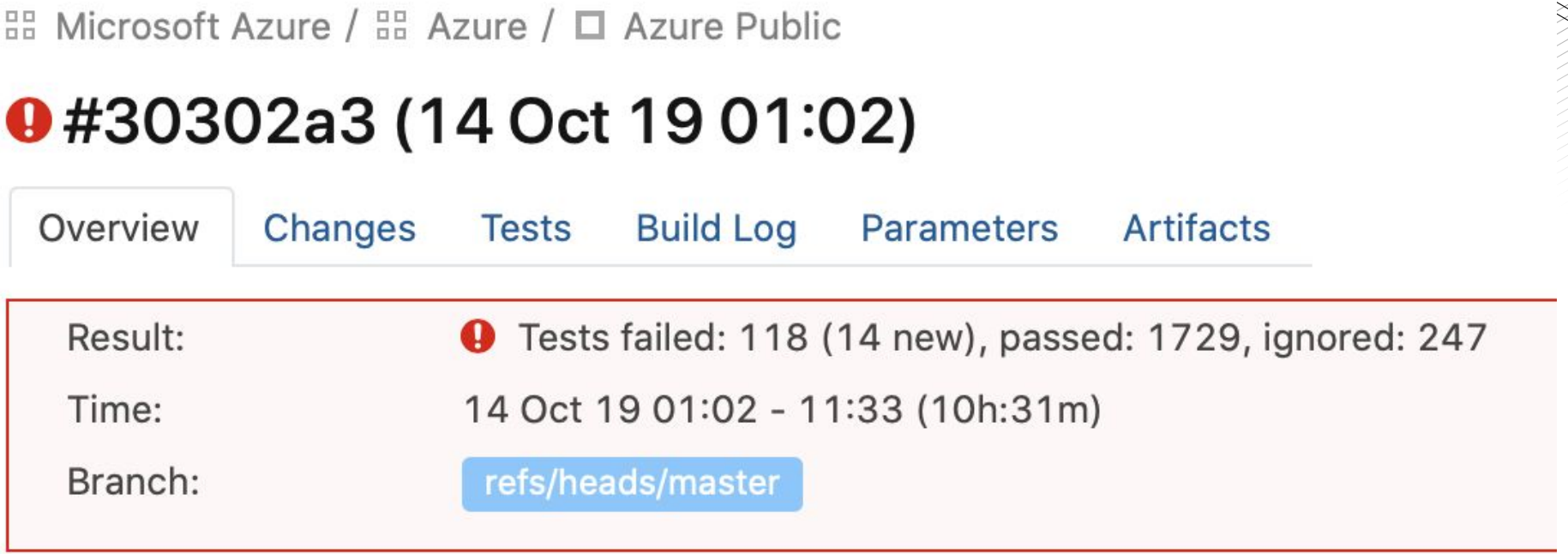


Pull-request acceptance tests for Azure Provider

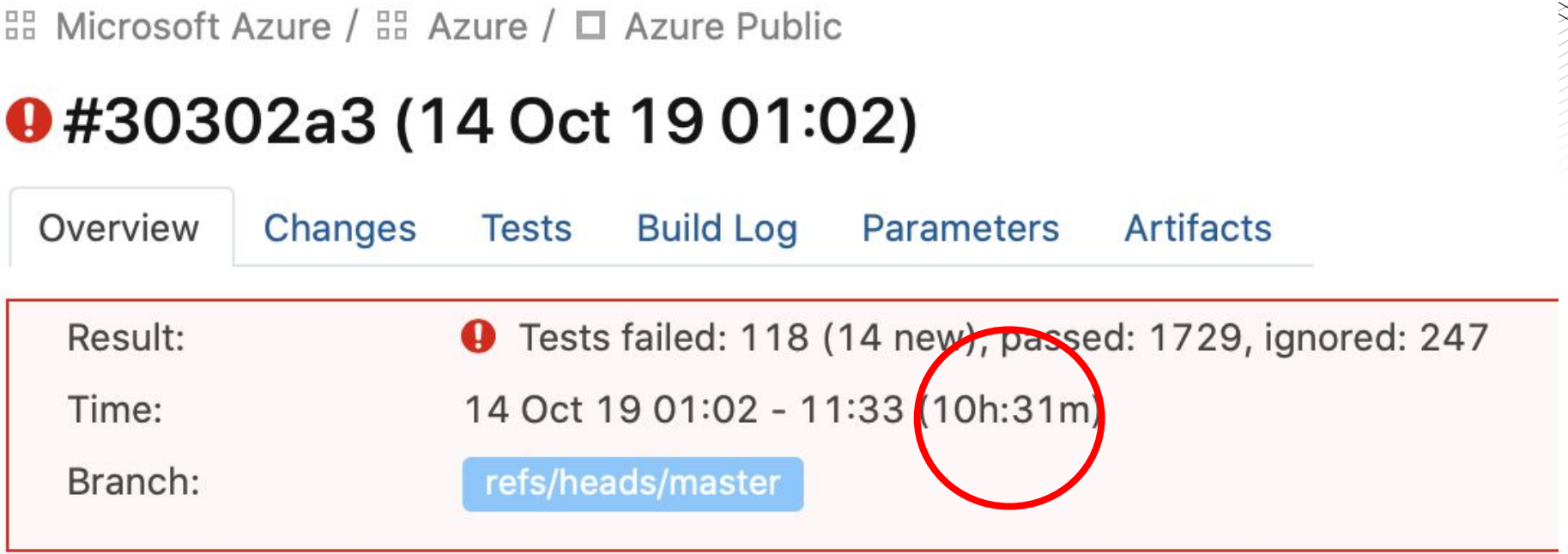
Azure Public

refs/heads/master	#17f20e0	Tests failed: 56 (3 new), passed: 782, ignored: 96; S...
/pull/4260/merge	#8d5c569	Tests failed: 8, passed: 9, ignored: 1
/pull/4615/merge	#d2d5b55	Tests failed: 1 (1 new), passed: 12, ignored: 2
/pull/4483/merge	#b6f7284	Tests failed: 6, passed: 41, ignored: 8
/pull/4618/merge	#47f4c5b	Tests passed: 7, ignored: 2
/pull/4609/merge	#f83b0bb	Tests passed: 4, ignored: 1
/pull/4605/merge	#b3303df	Tests passed: 1
/pull/3529/merge	#eca03cc	Tests passed: 4
refs/heads/master	#30302a3	Tests failed: 118 (14 new), passed: 1729, ignored: ...

Full Suite example for Azure:



Full Suite example for Azure:



terratest

https://github.com/gruntwork-io/terratest

- Full acceptance framework for Terraform
- Developed by Gruntworks
- Used to test large module deployments
- Written in Golang
- Helper methods for validating state

```go
func TestTerraformBasicExample(t *testing.T) {
    t.Parallel()

    terraformOptions := &terraform.Options{
        // The path to where our Terraform code is located
        TerraformDir: "../examples/terraform-basic-example",
    }

  // At the end of the test, run `terraform destroy` to clean up any resources that were created
    defer terraform.Destroy(t, terraformOptions)

    // This will run `terraform init` and `terraform apply` and fail the test if there are any errors
    terraform.InitAndApply(t, terraformOptions)

    instanceType := terraform.Output(t, terraformOptions, "instance_type")

    assert.Equal(t, instanceType, "t1.micro")

}
```

At Gruntwork we test dozens of modules with terratest. We have hooks to CircleCI to run tests on each commit. **For us, the ROI is huge - we've caught many regressions & problems thanks to the tests.**

It can definitely result in long test times. In some of our larger repos, test can take 45-60 minutes. What I normally do is add/update a test, run just that test locally to make sure it's working, then let the full suite run in CircleCI.

- **Ben Whaley, Gruntworks, HashiConf 2019 Slack**

terratest example

TERMINAL

```
$  go test -v test/terraform_basic_example_test.go
=== RUN   TestTerraformAWSInstanceType
=== PAUSE TestTerraformAWSInstanceType
=== CONT  TestTerraformAWSInstanceType
TestTerraformAWSInstanceType 2019-10-13T16:09:10+01:00 retry.go:72: terraform [init -upgrade=false]
TestTerraformAWSInstanceType 2019-10-13T16:09:10+01:00 command.go:87: Running command terraform with args
[init -upgrade=false]
TestTerraformAWSInstanceType 2019-10-13T16:09:21+01:00 command.go:158: aws_instance.foobar: Creating...
TestTerraformAWSInstanceType 2019-10-13T16:09:31+01:00 command.go:158: aws_instance.foobar: Still
creating... [10s elapsed]
TestTerraformAWSInstanceType 2019-10-13T16:09:41+01:00 command.go:158: aws_instance.foobar: Still
creating... [20s elapsed]
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158: aws_instance.foobar: Creation
complete after 21s [id=i-08d65b4371c14fc4e]
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158:
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158: Apply complete! Resources: 1
added, 0 changed, 0 destroyed.
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158:
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158: Outputs:
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158:
TestTerraformAWSInstanceType 2019-10-13T16:09:43+01:00 command.go:158: instance_type = t1.micro
ok    command-line-arguments  84.607s
```

Live Demo Warning!

State Testing

Pros

End-to-end testing

Closest to real testing

Interacts with real APIs

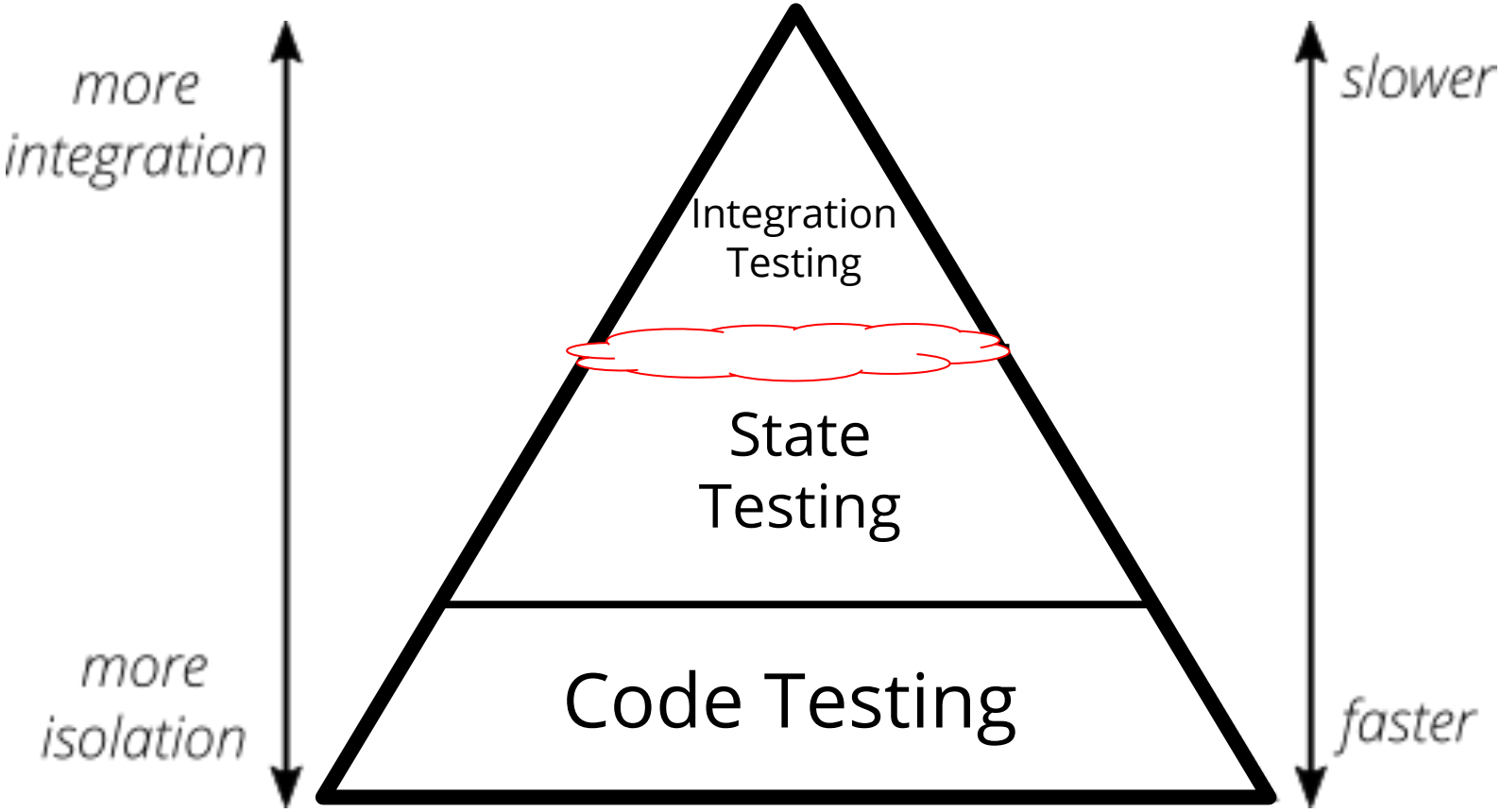
Cons

Slow

Can be Costly

Can interfere with real infrastructure

Integration Testing


more
integration
more
isolation
Integration
Testing
State
Testing
Code Testing
slower
faster

For me, the integration step is for testing the steps outside of Terraform’s direct control: Provisioning

Provisioning is very hard to model as state.

The tools that do it best are config management tools

Configuration management tools install and manage software on a machine that already exists. **Terraform is not a configuration management tool**, and it allows existing tooling to focus on their strengths: bootstrapping and initializing resources.

- https://www.terraform.io/intro/vs/chef-puppet.html

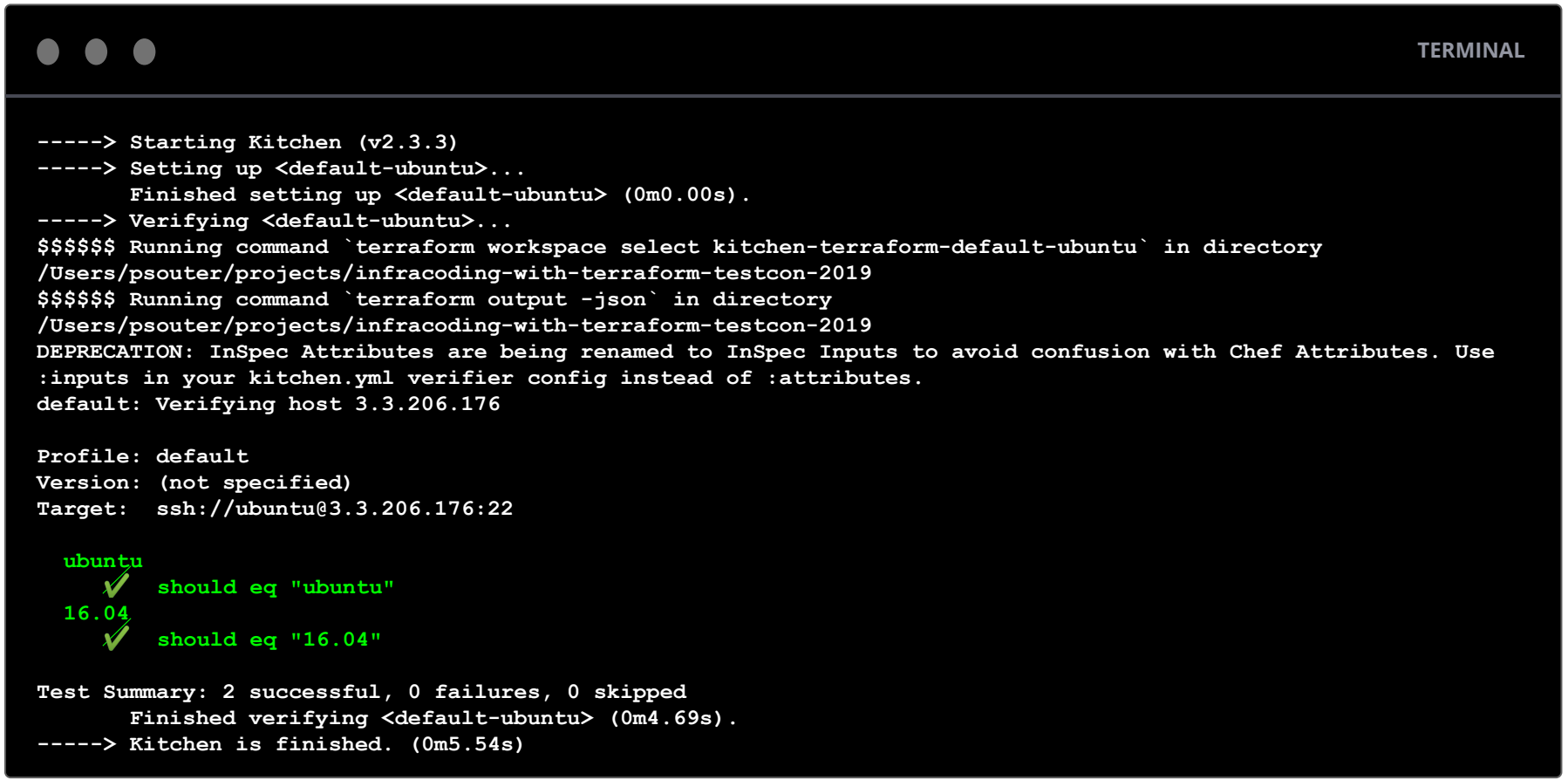
kitchen-terraform

https://newcontext-oss.github.io/kitchen-terraform/

- Fork of the test-kitchen tool
- Written in Ruby
- Various drivers and plugins
- Framework for creating infrastructure, then testing it against spec tests

```
---
driver:
  name: terraform

provisioner:
  name: terraform

transport:
  name: ssh
  ssh_key: ./keys/aws-testcon-demo-keypair.pem

verifier:
  name: terraform
  systems:
    - name: default
      backend: ssh
      user: ubuntu
      hosts_output: public_ip
      key_files:
        - ./keys/aws-testcon-demo-keypair.pem
suites:
  - name: default
```

kitchen-terraform example



Live Demo Warning!

Policy as Code and Policy Testing

Every organisation has policies

Naming conventions, tagging, price guides, legal bound constraints etc.

If we can reflect these as code, we get the same benefits we get from infrastructure as code

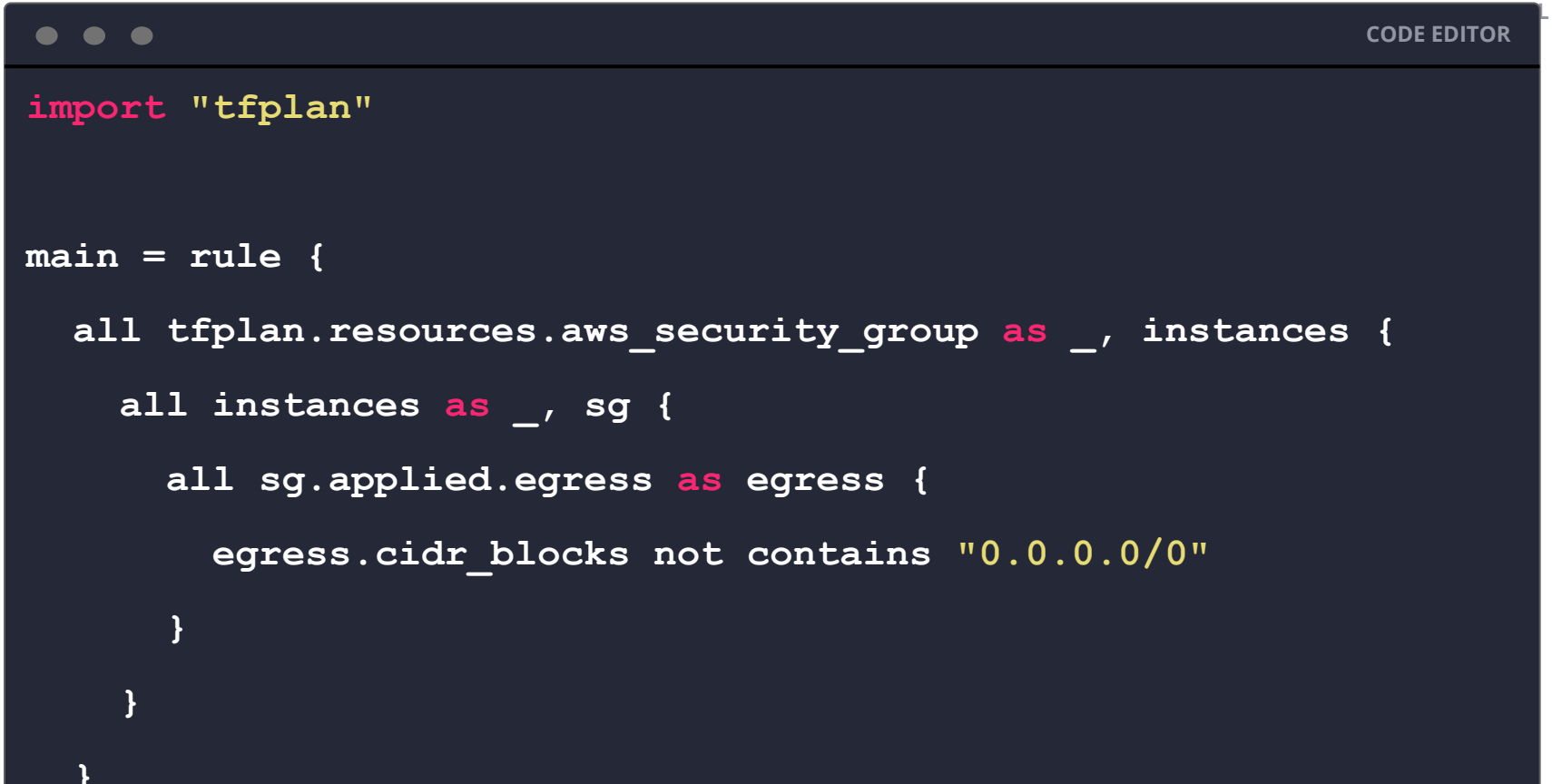
HashiCorp Sentinel

https://www.hashicorp.com/sentinel/

“An embeddable policy as code framework to enable fine-grained, logic-based policy decisions that can be extended to source external information to make decisions.”

```
import "tfplan"


main = rule {
  all tfplan.resources.aws_instance as _, instances {
    all instances as _, r {
      (length(r.applied.tags) else 0) > 0
    }
  }
}
```

Sentinel example for Terraform



Live Demo Warning

What have we learnt?

The benefits of testing IaC

Regardless of what tool you’re using, Infrastructure-as-code without testing is “self-sabotage”

How to do code/unit testing for Terraform

Allowing quick feedback loops and TDD

How to do state testing to test the final state

Allowing you assurance that your Terraform final state reflects what you expect

How to do integration testing to test the next steps after terraform

Getting deeper testing for things outside of direct Terraform control

How to model organisational policy as code

Would you like to know more?

- **InfraCoding with Terraform - Code Repo**

  https://github.com/petems/infracoding-with-terraform-testcon-2019
- **Open sourcing Terratest: a swiss army knife for testing infrastructure code**

  https://blog.gruntwork.io/open-sourcing-terratest-a-swiss-army-knife-for-testing-infrastructure-code-5d883336fcd5
- **About Kitchen-Terraform**

  https://newcontext-oss.github.io/kitchen-terraform/about.html
- **Test Terraform modules in Azure by using Terratest**

  https://docs.microsoft.com/en-us/azure/terraform/terratest-in-terraform-modules

Thank You

psouter@hashicorp.com
@petersouter
www.hashicorp.com