Elastic Cloud Engineering

How to keep your teams agile while managing many projects

PRESENTED BY: DENNIS VINK

Dennis Vink

AWS Cloud Consultant

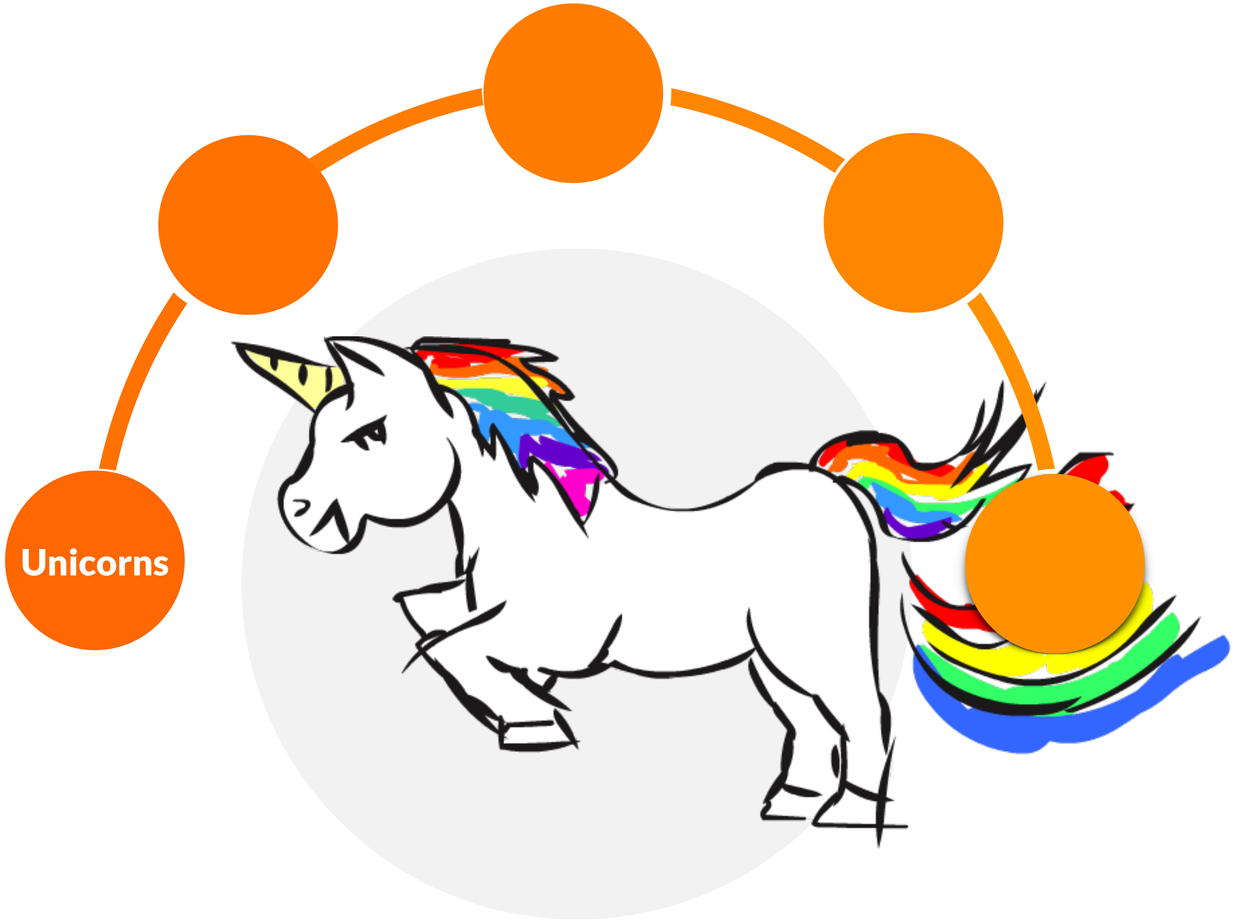
https://linkedin.com/in/drvink/

dennisvink@binx.io

https://github.com/dennisvink/

Unicorns!

I will talk about Unicorns, and why they are bad.



Happiness!

I will illustrate the ideal world, one with as few Unicorns as possible.

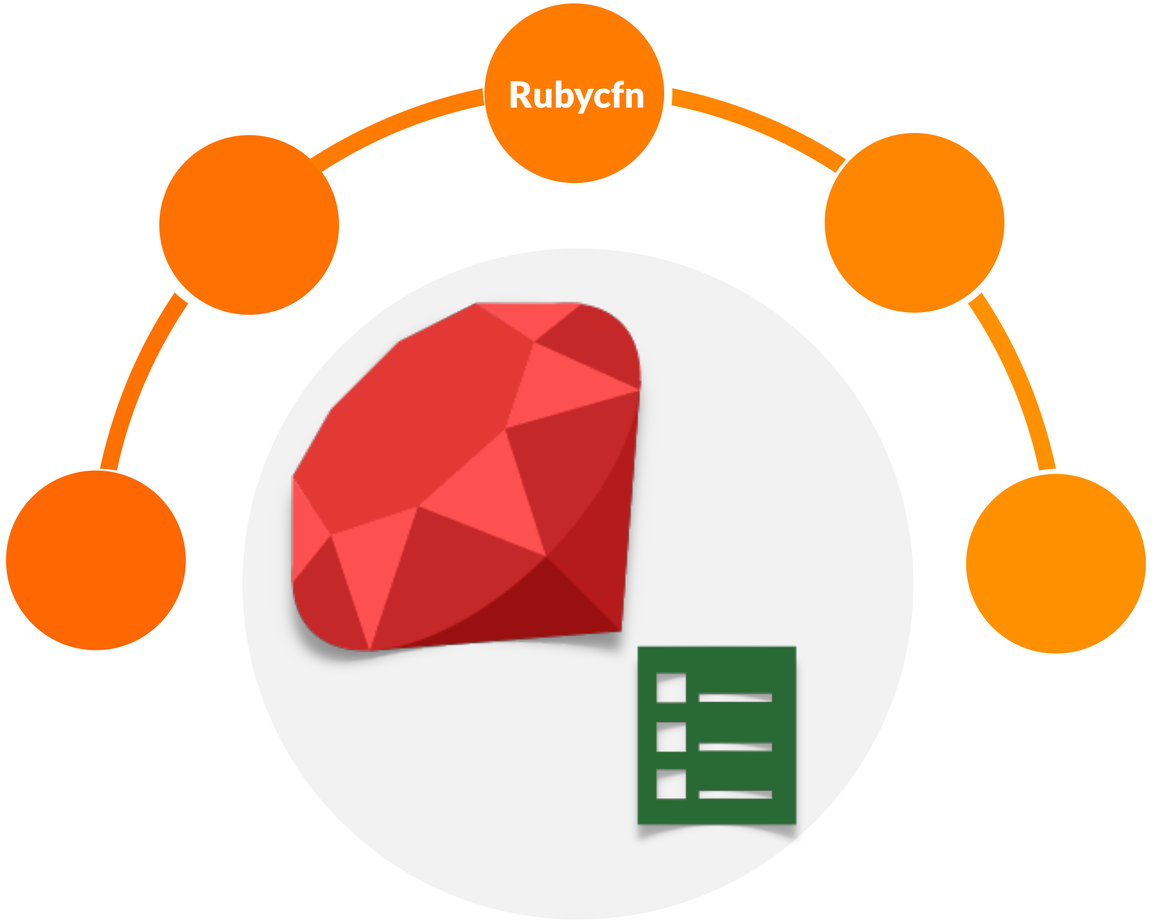
A happy, better world.

Rubywha-t?!

TALK COVERAGE

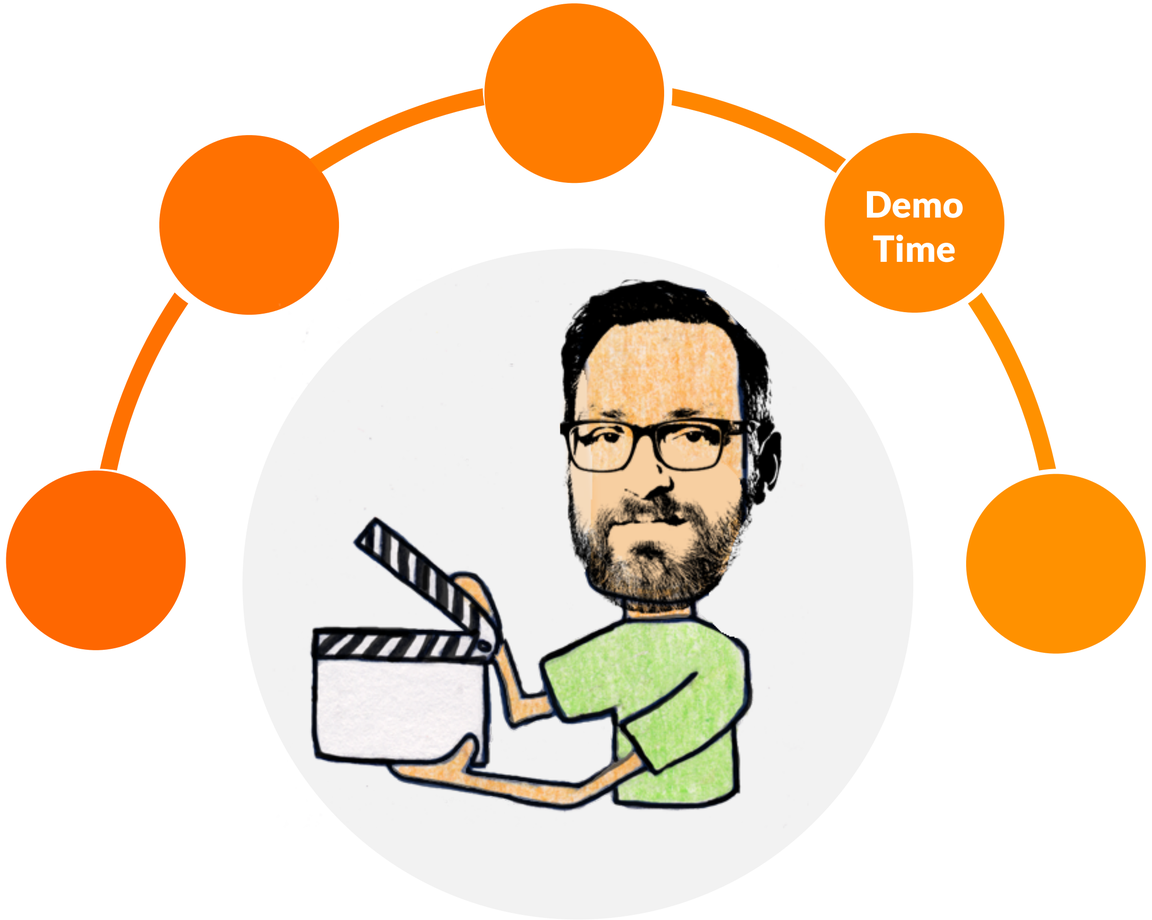
I will talk about Rubycfn, a tool that helps you to Write deterministic developer-friendly code to Produce CloudFormation templates.

And why using Rubycfn or not is not important at all.



Demo Time!

I will give a demo of a Rubycfn project in practice.



42

Life, the Universe, and Everything.

7 reasons to hate Unicorns

1

They are all different

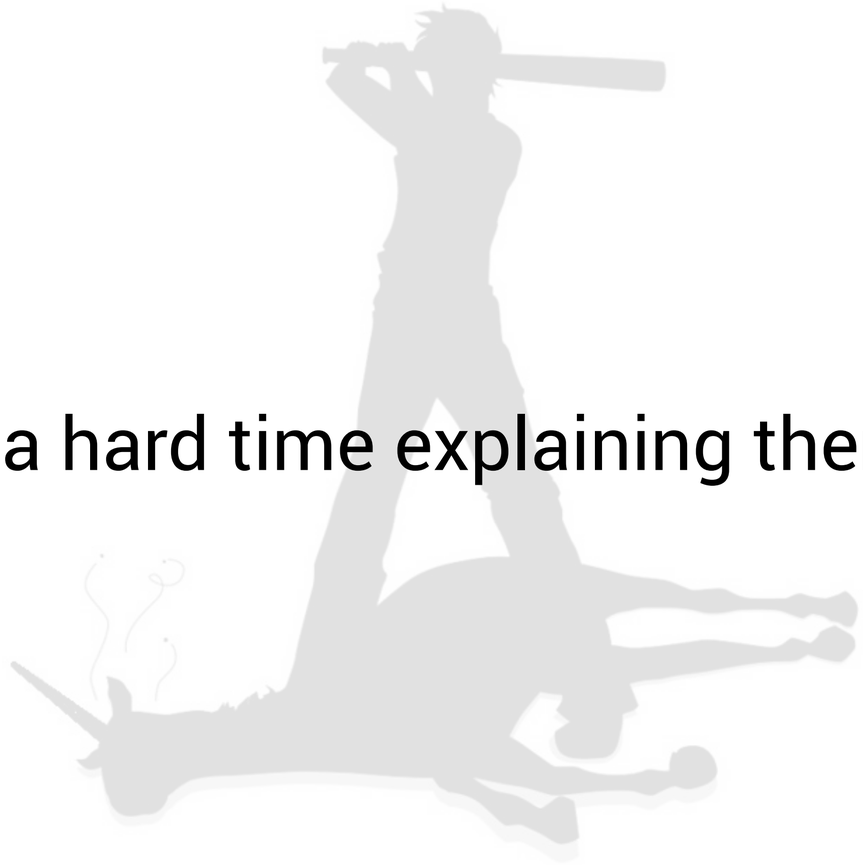
2

They are high maintenance

3

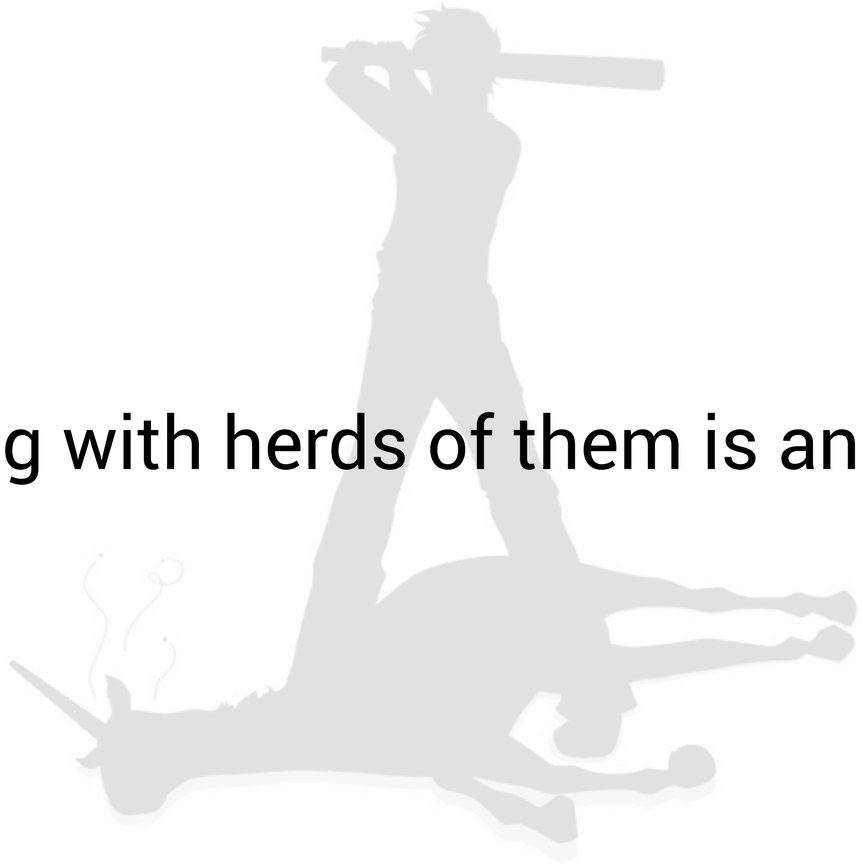
It takes a long time to get to know them

4



You have a hard time explaining them to your peers

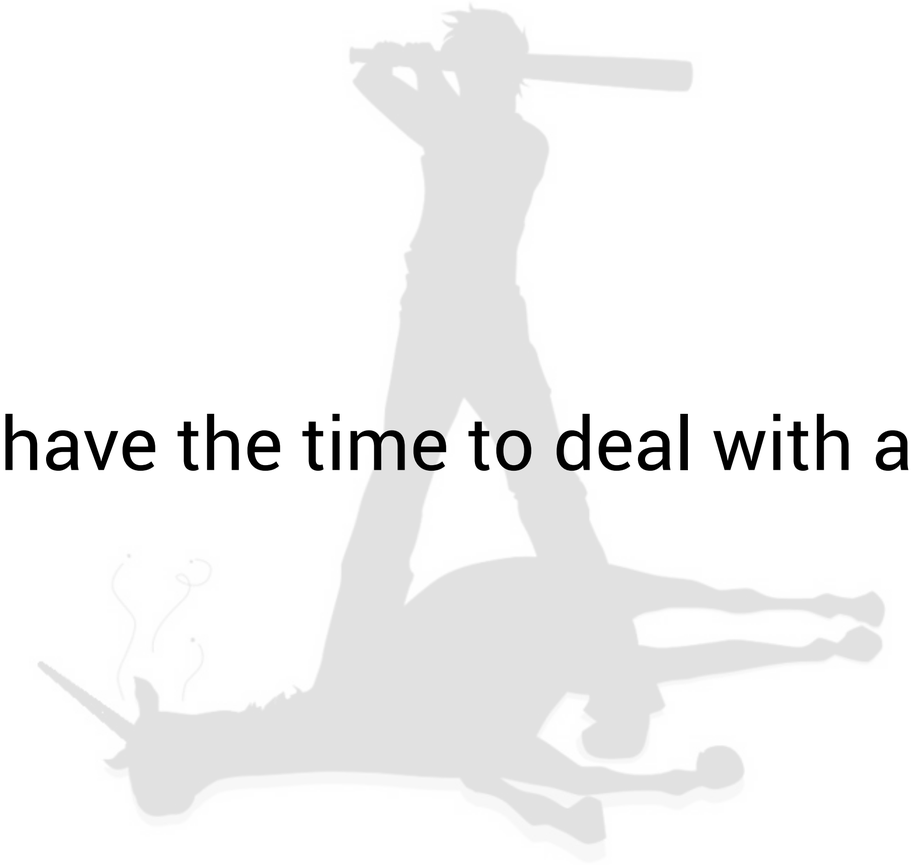
5

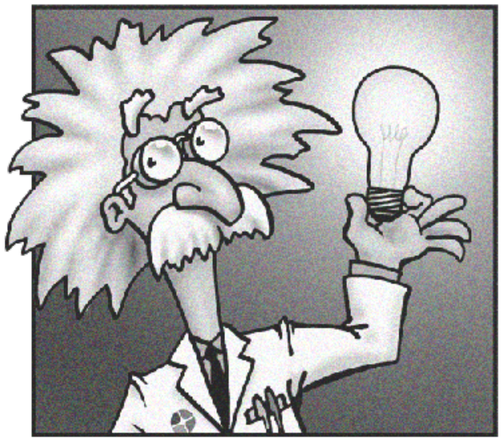


Hanging with herds of them is an energy drain

6

You don’t have the time to deal with all of their issues



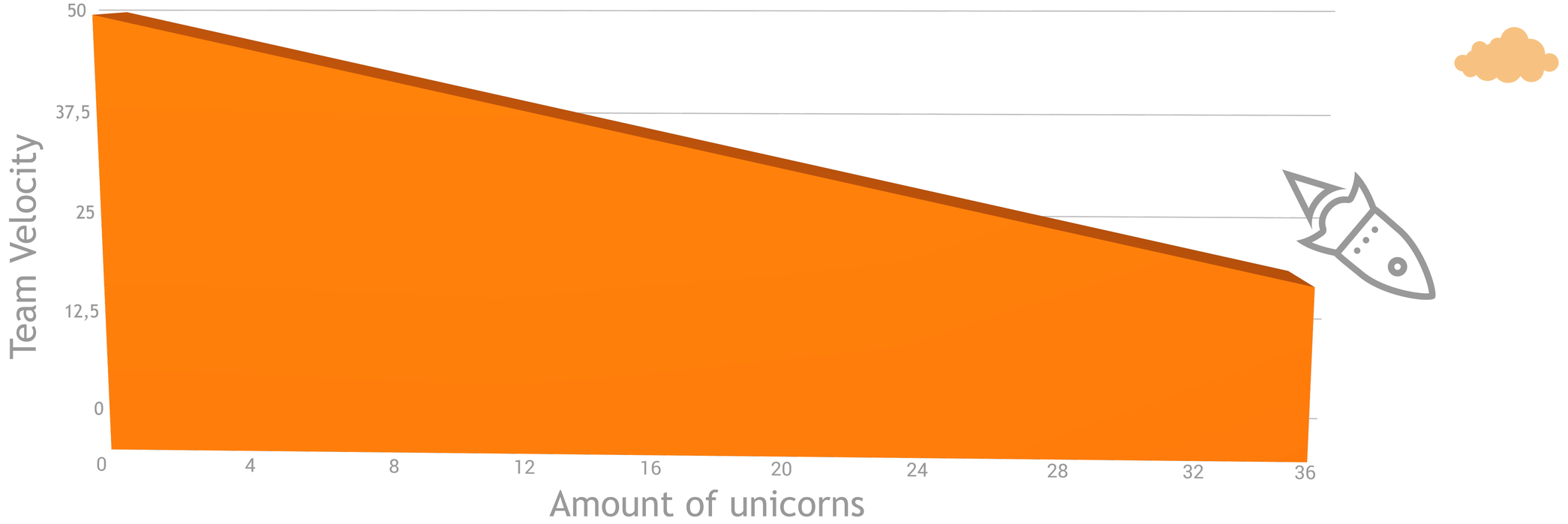


7

Science proofs:

Unicorns are detrimental to your velocity

SCIENTIFIC EVIDENCE



UNICORNS ARE:

Projects that differ in structure, maintenance or implementation from other projects.

VELOCITY IS:

A measure of the amount of work a team can tackle during a single iteration.

TIME IS A SEQUENTIAL EPHEMERAL CURRENCY

For Cloud Engineers to remain agile and thus elastic they must adopt a franchise mentality. Everything you do must be repeatable by others, and have a deterministic outcome.

THE MORE UNICORNS THE LESS WORK THAT ACTUALLY GETS “DONE”

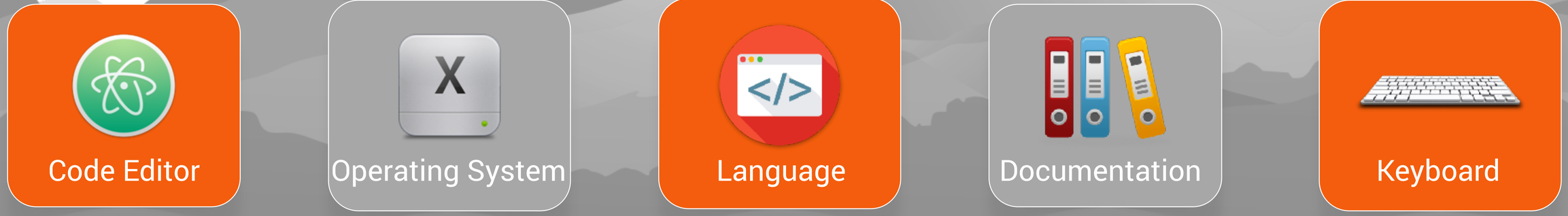
A WORLD WITHOUT UNICORNS

Eradicating Exceptions

1. Enforce **consistent tooling** within teams

Colleagues that are not able to do what you are doing at any given moment are not elastic.

For Elastic Cloud Engineers consistency is key!

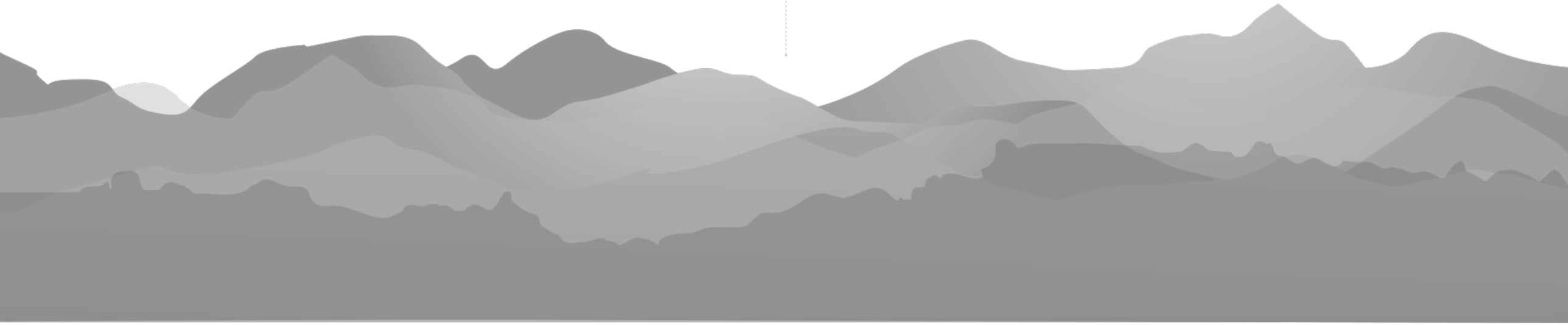


Eradicating Exceptions

5 PRINCIPLES

2 Standardise how projects are structured

- Automate new project structure
- Design for modularity

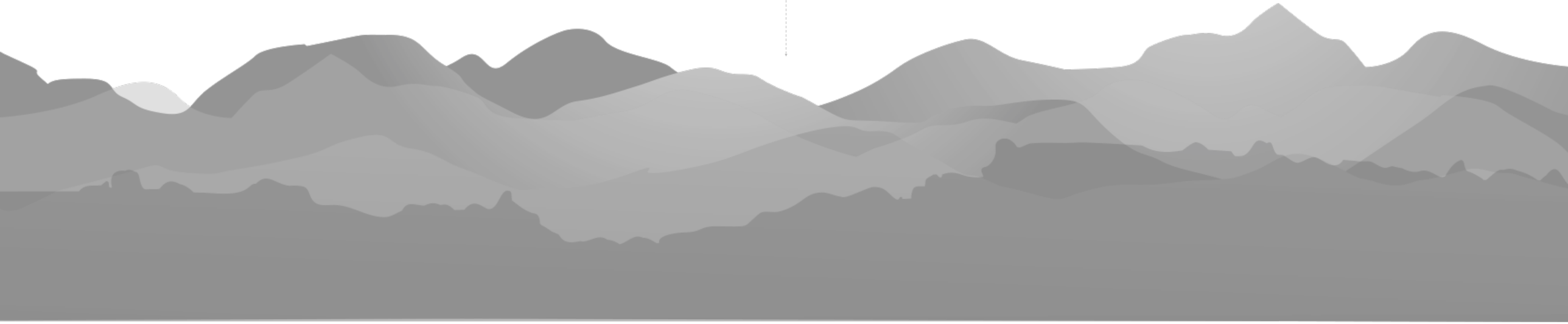


Eradicating Exceptions

5 PRINCIPLES

3 Drink your own medicine

- Standardise how developers build code
- Standardise how developers test code
- Standardise how developers deploy code
- **All code must go through a CI/CD pipeline**



Eradicating Exceptions

5 PRINCIPLES

4 Automate code quality

- Make unit tests part of your project code
- Enforce style guides ruthlessly
- LINTers are your friend
- **Protect your branches**

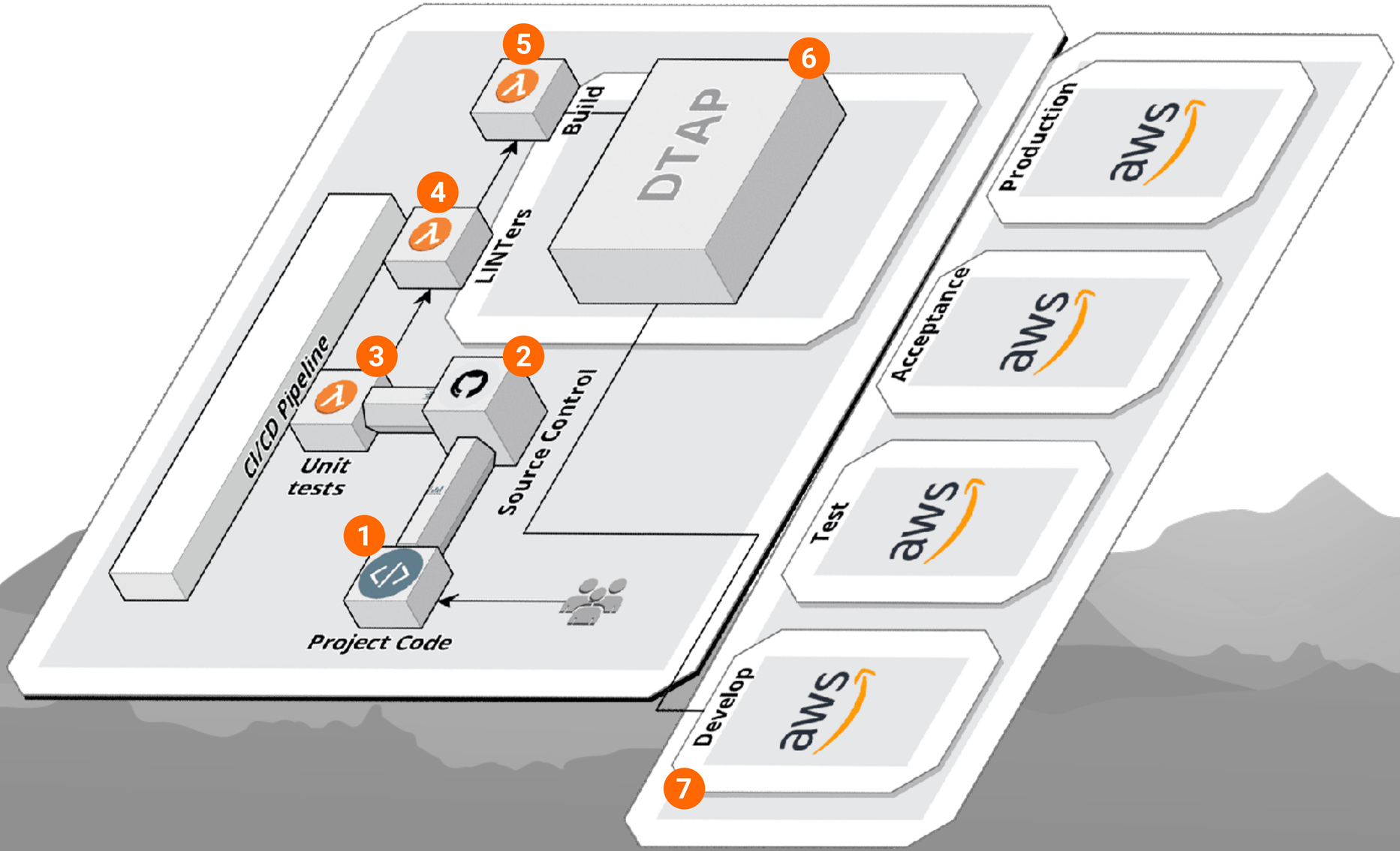
Eradicating Exceptions

5 PRINCIPLES

5 Automate solution quality

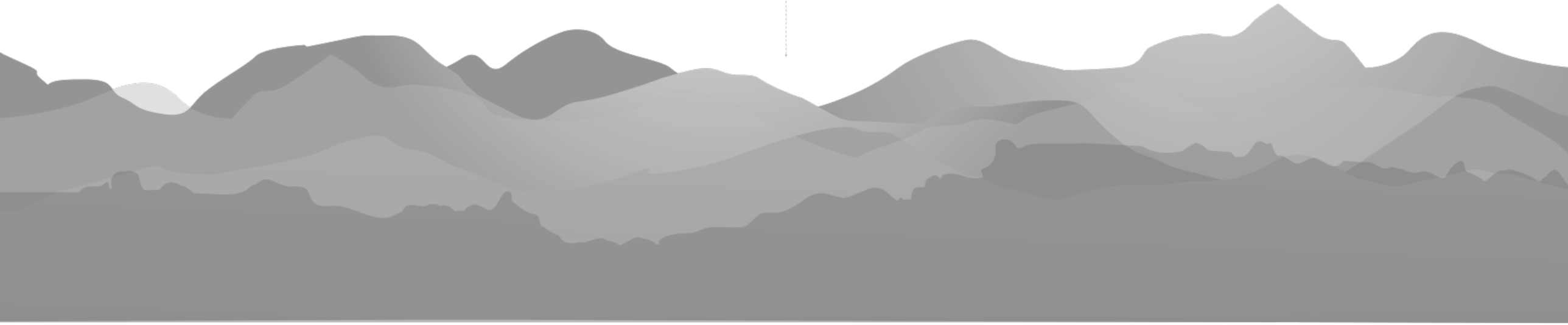
- Treat infrastructure as immutable
- No access to production systems
- Automate patch management
- **Enforce DTAP deployment strategy**

WHAT DO THESE PRINCIPLES LOOK LIKE IN PRACTICE ?


5
Build
6
DTAP
4
LINTers
3
2
CI/CD Pipeline
Unit tests
Source Control
1
Project Code
Production
Acceptance
Test
Develop
7

1 PROJECT CODE

- Produces CloudFormation as build artifact
- Contains tests for resulting template
- Has style guide standard configuration
- Validates template validity and style



2 SOURCE CONTROL

- Integrates with CI/CD pipeline
- Protected branches
- Requires build to succeed before merging
- Requires peer reviews

3 UNIT TESTING

- Tests are part of your code base
- Write tests before you write code
- Automatically generate test reports

4 LINTERS

- LINTer configuration as part of your code base
- Enforces style guides
- Reports on errors in your CloudFormation templates
- Provides useful warnings

5 BUILDS

- Compiles your code
- Runs tests against the resulting template
- Runs LINTers to enforce standards
- Stores the artifact

6 DEPLOYMENTS

- Uses a build artifact as input
- Changes parameters such as instance sizing based on the environment you deploy to
- Enforced DTAP flow

7 AWS Stack

- One account per stack
- A stack for each environment
- Unique address space per stack
- No AWS console access

A DETERMINISTIC UNIVERSE

DETERMINISTIC UNIVERSE

Building your project code results in the same template, every time.

Because you have a test driven approach, you have a good idea about the outcome, even when you deploy for the first time.

Tests are performed against the versioned collective of modules and tools contained within your project's code.

Roll backs are easy, because your code is versioned and the outcome is deterministic.

A DETERMINISTIC UNIVERSE IS AN IMMUTABLE UNIVERSE

So… how do we shape this world?

RUBYCFN CAN HELP YOU

Ruby is designed to help you enjoy your life by helping you get your job done more quickly and with more fun

Yukihiro Matsumoto - https://www.artima.com/intv/rubyP.html

So what is RubyCfn ?

- It is an abstraction layer for CloudFormation
- Written in Ruby, for maximum happiness
- Comes with a structured project by default
- That is test-driven and style checked

To ensure you write top quality code!

Who wrote RubyCfn ?

Who wrote RubyCfn ?

“So aren’t you biased ?”

JUST AS LONG AS…

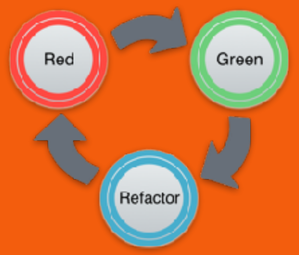
Output

CloudFormation

Result

Deterministic

Utilises



TDD

Assures

Quality Code

GETTING STARTED WITH RUBYCFN

https://github.com/dennisvink/rubycfn/

1 Installing RubyCfn:

Command line:

```
gem install rubycfn
```

https://github.com/dennisvink/rubycfn/

2 Creating a new project:

Command line:

```
rubycfn
```

https://github.com/dennisvink/rubycfn/

2 Creating a new project:

```
Macbook-Pro-Dennis-2:cfncode dennis$ rubycfn
 _____ _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
|     |     |     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |     |     |
|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
                                                              [v0.1.6]
Project name? example
AWS Account ID? 1234567890
Select AWS region
  US West (N. California)
  US West (Oregon)
  Canada (Central)
  EU (Frankfurt)
▸ EU (Ireland)
  EU (London)
(Move up or down to reveal more choices)
```

https://github.com/dennisvink/rubycfn/

Command line:

3 Installing dependencies:

```
bundle
```

https://github.com/dennisvink/rubycfn/

4 Compile, test and build

Command line:

```
rake
```

https://github.com/dennisvink/rubycfn/

Every stack consists of one or more modules. A stack is a collection of modules that collectively renders into a CloudFormation template. This allows you to structure your code more efficiently.

```
module ExampleStack
  extend ActiveSupport::Concern
  include Rubycfn

  included do
    include ExampleStack::Main
  end
end
```

https://github.com/dennisvink/elastic-cloud-engineering/

Writing CloudFormation templates in Rubycfn looks like this:

```
module ExampleStack
  module Main
    extend ActiveSupport::Concern

    included do
      variable :sqs_queue_name,
               default: "defaultname",
               value: ENV["SQS_QUEUE_NAME"]

      resource :my_awesome_queue,
               type: "AWS::SQS::Queue" do |r|
          r.property(:queue_name) { sqs_queue_name }
      end
    end
  end
end
```

https://github.com/dennisvink/elastic-cloud-engineering/

The resource names and properties relate 1:1 to their CloudFormation counterpart:

```ruby
module ExampleStack
  module Main
    extend ActiveSupport::Concern

    included do
      variable :sqs_queue_name,
               default: "defaultname",
               value: ENV["SQS_QUEUE_NAME"]

      resource :my_awesome_queue,
               type: "AWS::SQS::Queue" do |r|
        r.property(:queue_name) { sqs_queue_name }
      end
    end
  end
end
```

Variables you can use in your code

Read from .env or .env.<environment>

Resource name, Camel Cases on compile

MyAwesomeQueue

QueueName

Camel cased on compile to CloudFormation as well

https://github.com/dennisvink/elastic-cloud-engineering/

When you `rake` (build) the project the resulting template looks like this:

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "MyAwesomeQueue": {
      "Properties": {
        "QueueName": "defaultname"
      },
      "Type": "AWS::SQS::Queue"
    }
  }
}
```

my_awesome_queue in rubycfn

queue_name in rubycfn

https://github.com/dennisvink/elastic-cloud-engineering/

https://www.cfnflip.com

But you can also:

```
module ExampleStack
  module Main
    extend ActiveSupport::Concern

    included do
      parameter :sqs_queue_name,
                default: "defaultname"

      resource :my_awesome_queue,
               type: "AWS::SQS::Queue" do |r|
        r.property(:queue_name) { "SqsQueueName".ref }
      end
    end
  end
end
```

Define stack parameters

And ‘Ref’ to them in your code

https://github.com/dennisvink/elastic-cloud-engineering/

RubyCfn supports many of AWS intrinsic functions, such as:

Fn::GetAtt

```
r.property(:foobar) { "Resource".ref("Arn") }
r.property(:foobar2) { "Reso,urce".fnsplit(",") }
r.property(:foobar3) { ["Reso", "urce"].fnjoin(",") }
r.property(:foobar4) { "Something".fnbase64 }

# And also supports Fn::GetAZs, Fn::Sub, Fn::ImportValue,
# Fn::Cidr, Fn::Select, Fn::FindInMap and more.
```

https://github.com/dennisvink/elastic-cloud-engineering/

You can create multiple resources of the same time without code duplication, Output values, and export them.

```ruby
module ExampleStack
  module Main
    extend ActiveSupport::Concern

    included do
      parameter :sqs_queue_name,
                default: "defaultname"

      resource :my_awesome_queue,
               amount: 3,
               type: "AWS::SQS::Queue"

      output :some_output,
             description: "SQS Queue",
             value: "MyAwesomeQueue".ref,
             export: "SQSQueueName"
    end
  end
end
```

```json
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "SqsQueueName": {
      "Default": "defaultname",
      "Type": "String"
    }
  },
  "Resources": {
    "MyAwesomeQueue": {
      "Type": "AWS::SQS::Queue"
    },
    "MyAwesomeQueue2": {
      "Type": "AWS::SQS::Queue"
    },
    "MyAwesomeQueue3": {
      "Type": "AWS::SQS::Queue"
    }
  },
  "Outputs": {
    "SomeOutput": {
      "Description": "SQS Queue",
      "Export": {
        "Name": "SQSQueueName"
      },
      "Value": {
        "Ref": "MyAwesomeQueue"
      }
    }
  }
}
```

https://github.com/dennisvink/elastic-cloud-engineering/

By default RubyCfn comes with a serverless CI/CD stack that integrates with Github.

	Name	Status	Duration
▸	SUBMITTED	Succeeded	
▸	PROVISIONING	Succeeded	17 secs
▸	DOWNLOAD_SOURCE	Succeeded	2 secs
▸	INSTALL	Succeeded	23 secs
▸	PRE_BUILD	Succeeded	
▸	BUILD	Succeeded	
▸	POST_BUILD	Succeeded	5 secs
▸	UPLOAD_ARTIFACTS	Succeeded	
▸	FINALIZING	Succeeded	2 secs
▸	COMPLETED	Succeeded	

https://github.com/dennisvink/elastic-cloud-engineering/

It is part of your project. You configure the build steps in config/buildspec.yml:

```
version: 0.1

phases:
  install:
    commands:
      - gem install cfn-nag
      - bundle install
  pre_build:
    commands:
      - echo Do nothing
  build:
    commands:
      - echo Build started on `date`
      - rake
      - cfn_nag_scan --input-path build
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - ./**
```

https://github.com/dennisvink/elastic-cloud-engineering/

When you open a pull request in Github, this will trigger Codebuild to perform
The steps you defined in the ‘buildspec.yml’ and report back to Github.

Create PR

CodeBuild
Tests & Linters

Codebuild
Compile code

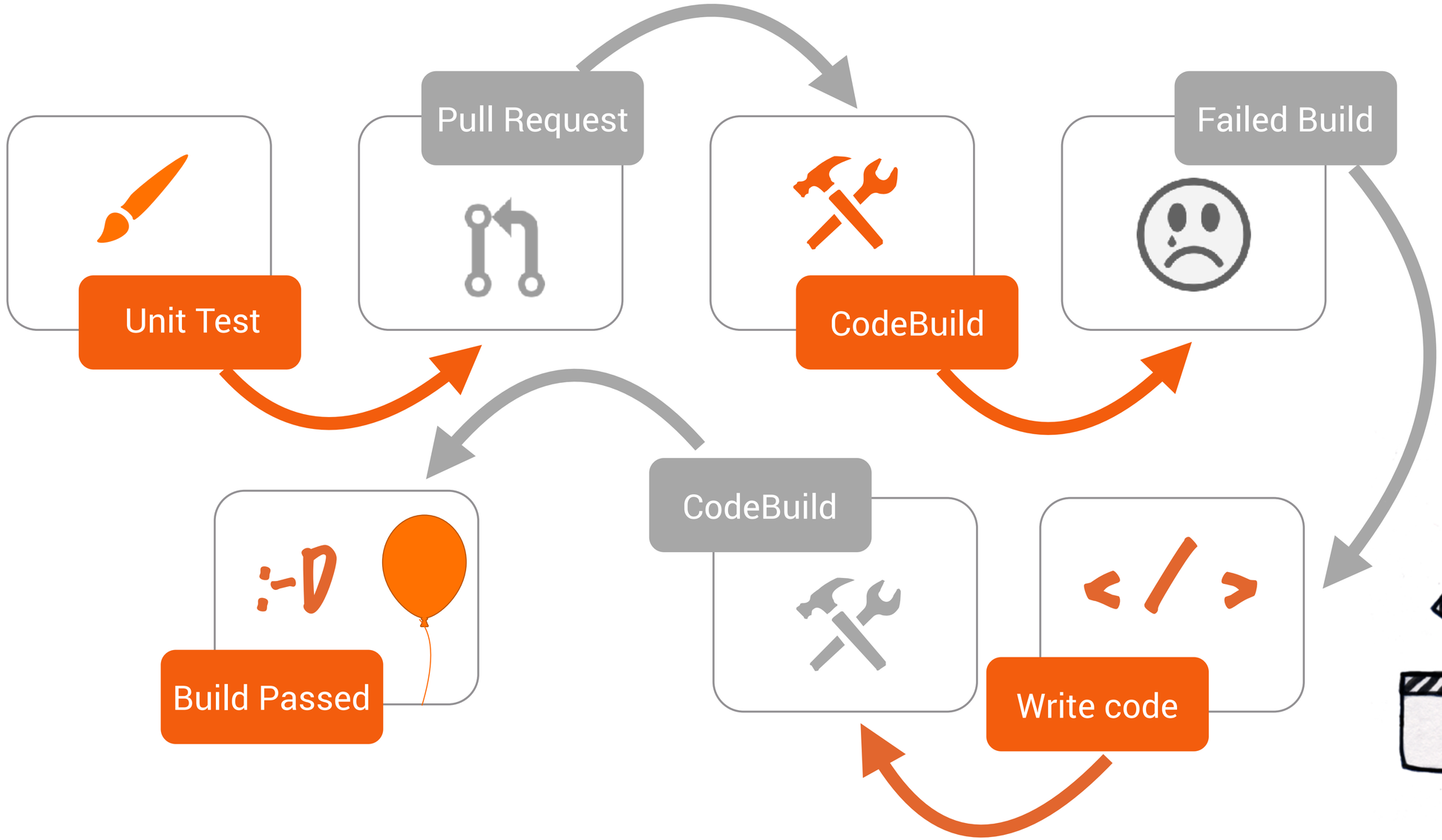
Report back
to Github

https://github.com/dennisvink/elastic-cloud-engineering/

A default RubyCfn projects comes with:

- A serverless CI/CD pipeline with Github integration
- Automated testing using `rspec`
- Automated code style testing using `rubocop`
- Pre-written unit tests which you can extend upon to test your CloudFormation templates
- Organised modular structure so you can keep your projects clean and readable

DEMO TIME


Unit Test
Pull Request
CodeBuild
Failed Build
Write code
CodeBuild
:-D
Build Passed

PHILOSOPHICAL PONDERING

IF THE UNIVERSE IS DETERMINISTIC AREN'T UNICORNS JUST A PART OF IT ?

THE ANSWER IS OBVIOUSLY “YES”

BUT REMEMBER…

SO AM I.

ANYTHING TO ADD ?