Infrastructure as Code

Software Architecture

Brae Webb & Richard Thomas

March 17, 2025

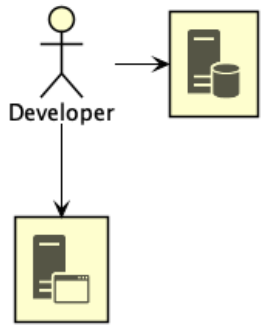
- Do a quick poll.
- Who has heard the term IaC before this course?
- Who has used IaC before this course?

How did we get here?

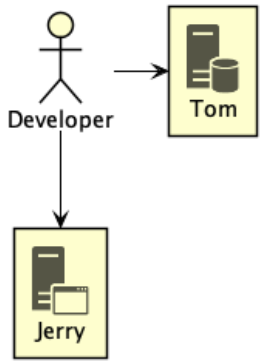
Pre-2000

The *Iron Age*

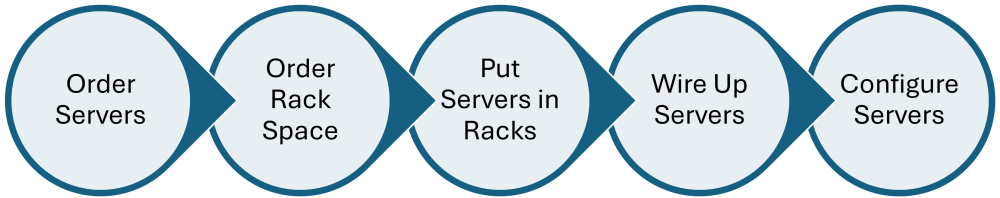
Iron Age



Iron Age



Developer only had a few machines — so few the machines often got fun names.

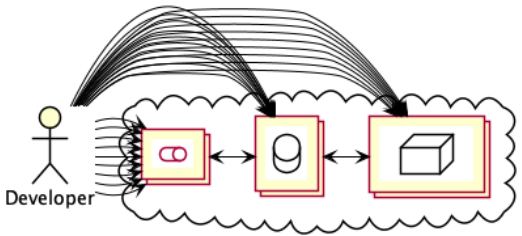


I used to have scheduling examples for software projects, where ordering servers was on the critical path, and had to happen before software development started.

Introducing...

The *Cloud Age*

The Cloud Age



- Summarise: things got complicated quickly, we need more hardware and it's easier to provision.
- Largely thanks to virtualisation — no physical activity for a new machine.

When faced with complexity

Automate it!

- We have too much to manage to do it manually.
- We're about to start enumerating automation techniques.

The larger story

Server Config Config Management

The larger story

Server Config	Config Management
Application Config	Config Files

The larger story

Server Config	Config Management
Application Config	Config Files
Provisioning	Infrastructure Code

The larger story

Server Config	Config Management
Application Config	Config Files
Provisioning	Infrastructure Code
Building	Continuous Integration

The larger story

Server Config	Config Management
Application Config	Config Files
Provisioning	Infrastructure Code
Building	Continuous Integration
Deployment	Continuous Deployment

The larger story

Server Config	Config Management
Application Config	Config Files
Provisioning	Infrastructure Code
Building	Continuous Integration
Deployment	Continuous Deployment
Testing	Automated Tests

The larger story

Server Config	Config Management
Application Config	Config Files
Provisioning	Infrastructure Code
Building	Continuous Integration
Deployment	Continuous Deployment
Testing	Automated Tests
Database Administration	Schema Migration

The larger story

Server Config	Config Management
Application Config	Config Files
Provisioning	Infrastructure Code
Building	Continuous Integration
Deployment	Continuous Deployment
Testing	Automated Tests
Database Administration	Schema Migration
Specifications	Behaviour Driven Development

Definition 0. Infrastructure Code

Code that provisions and manages *infrastructure resources*.

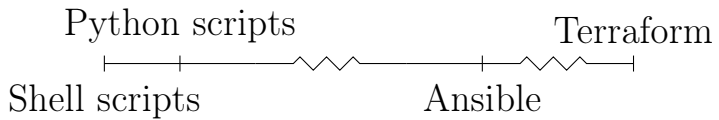
Definition 0. Infrastructure Code

Code that provisions and manages *infrastructure resources*.

Definition 0. Infrastructure Resources

Compute resources, networking resources, and storage resources.

Infrastructure Code



IC often thought of as the right-hand side but includes all.

```
#!/bin/bash

SG=$(aws ec2 create-security-group ...)

aws ec2 authorize-security-group-ingress --group-id "$SG"

INST=$(aws ec2 run-instances --security-group-ids "$SG" \
        --instance-type t2.micro)
```

Using AWS CLI to create EC2 access like the practical.

Python

```python
import boto3

def create_instance():
    ec2_client = boto3.client("ec2", region_name="us-east-1")
    response = ec2.create_security_group(...)
    security_group_id = response['GroupId']

    data = ec2.authorize_security_group_ingress(...)

    instance = ec2_client.run_instances(
        SecurityGroups=[security_group_id],
        InstanceType="t2.micro",
        ...
    )
```

Using AWS Python library (boto3).

Terraform

```
resource "aws_instance" "hextris-server" {
    instance_type = "t2.micro"
    security_groups = [aws_security_group.hextris-server.name]
    ...
}

resource "aws_security_group" "hextris-server" {
    ingress {
        from_port = 80
        to_port = 80
        ...
    }
    ...
}
```

Finally, Terraform.

Question

Notice anything different?

- Prompting for declarative.
- Might notice verbosity.

The main difference

Imperative vs. Declarative

- *Imperative* – Describe the steps to take to deploy the infrastructure
- *Declarative* – Describe the desired infrastructure
- IC is heading towards a more *declarative* paradigm.

Declarative IaC

- Define your *desired* infrastructure state
  - as code
- Engine interprets difference between the *desired* and *actual* state
  - Modifying infrastructure to deliver *desired* state

Infrastructure Code

- Provisions and manages *infrastructure resources*.

Infrastructure Code

- Provisions and manages *infrastructure resources*.
- Only one part of the movement to *automate* the complexities of development.

Infrastructure Code

- Provisions and manages *infrastructure resources*.
- Only one part of the movement to *automate* the complexities of development.
- Ranges from simple shell scripts up to. . . ?

Infrastructure Code

- Provisions and manages *infrastructure resources*.
- Only one part of the movement to *automate* the complexities of development.
- Ranges from simple shell scripts up to. . . ?
- Tendency to be *declarative*.

Summarising what we've already covered.

Typo?

Infrastructure Code ≠ Infrastructure *as* Code

- Mention that this distinction is ours.
- Real world unfortunately mixes the two.

Definition 0. Infrastructure as Code

Following the same *good coding practices* to manage Infrastructure Code as standard code.

Warning!

Infrastructure as Code still *early* and quite *bad*.

- Code reuse is low.
- Importing existing resources is non-trivial.
- Refactoring is painful.
- State management can be tricky.

Question

What are *good coding practices*?

Ask the class.

Good Coding Practice #1

Everything as Code

A practice we do but barely discuss in 'regular' programming because it doesn't make sense not to do it.

```
#!/bin/bash

./download-dependencies
./build-resources
cp -r output/* artifacts/
```

```
#!/bin/bash

./download-dependencies
./build-resources
cp -r output/* artifacts/
```

```
$ cp: directory artifacts does not exist
```

An example of relying on external state in ‘regular’ programming.

```
resource "aws_instance" "hextris-server" {
    instance_type = "t2.micro"
    security_groups = ["sg-6400"]
    ...
}
```

Draw a parallel to the bash example and this, which relies on ‘sg-6400’ existing.

```
resource "aws_instance" "hextris-server" {
    instance_type = "t2.micro"
    security_groups = [aws_security_group.hextris-server.name]
    ...
}

resource "aws_security_group" "hextris-server" {
    ingress {
        from_port = 80
        to_port = 80
        ...
    }
    ...
}
```

The better approach.

Everything as code avoids

Configuration drift

Configuration drift creates

Snowflakes

- Snowflakes: magical machines that 'just work' and everyone is afraid to touch.
- Snowflake because they're unique and easy to break, because no one knows how it works.

Benefits

1. Reproducible

Good Coding Practice #2

Version Control

Benefits

1. Restorable
2. Accountable

Good Coding Practice #3

Automation

Benefits

1. Consistent

Automatically applying or checking IC is in sync means the main branch is consistent with reality.

Good Coding Practice #4

Code Reuse

Benefits

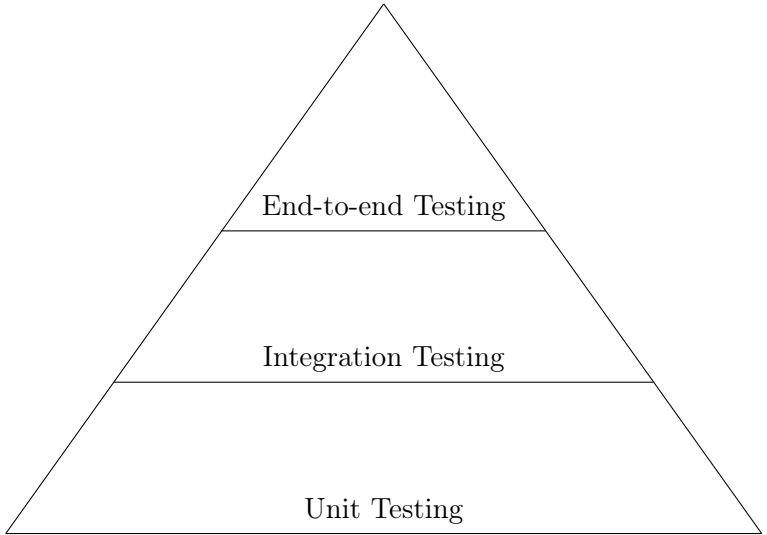
1. Better[1] code
2. Less work
3. Only one place to update (or verify)

[1] generally

Good Coding Practice #5

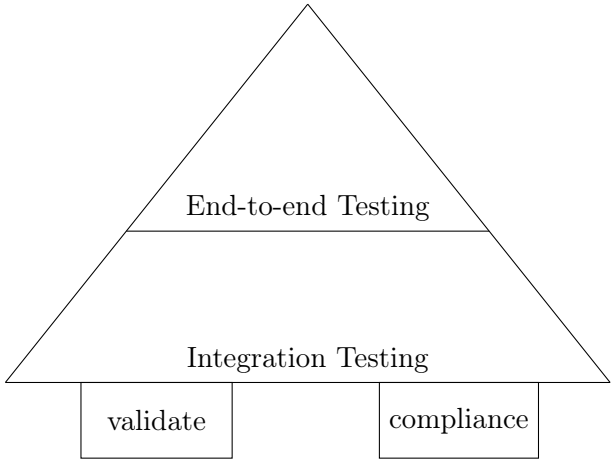
Testing

Test Pyramid



- Traditional test pyramid.
- Unit testing relies on isolated testing.
- But... isolated testing doesn't make *much* sense for IaC.

IaC Test Pyramid



```go
func TestTerraformAwsInstance(t *testing.T) {
    terraformOptions := terraform.WithDefault(t, &terraform.Options{
        TerraformDir: "../week03/",
    })

    defer terraform.Destroy(t, terraformOptions)
    terraform.InitAndApply(t, terraformOptions)

    publicIp := terraform.Output(t, terraformOptions, "public_ip")
    url := fmt.Sprintf("http://%s:8080", publicIp)

    http_helper.HttpGetWithCustomValidation(t, url, nil, 200,
        func(code, resp) { code == 200 &&
                           strings.Contains(resp, "hextris")})
}
```

An example of validation.

```
Feature: Define AWS Security Groups

Scenario: Only selected ports should be publicly open
    Given I have AWS Security Group defined
    When it contains ingress
    Then it must only have tcp protocol and port 22,443 for 0.0.0.0/0
```

An example of compliance testing.

Benefits

1. Trust

Prac Next Week

Learn how to use Terraform to write IaC and deploy resources on AWS.