

# Infrastructure as Code

CSSE6400

Brae Webb

March 21, 2022

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

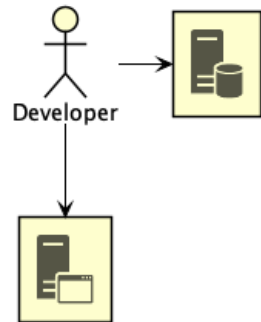
Infrastructure as Code

# 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.

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 virtualization — 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
- Application Config
- Provisioning
- Config Management
- Config Files
- 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 1. Infrastructure Code

Code that provisions and manages *infrastructure resources*.

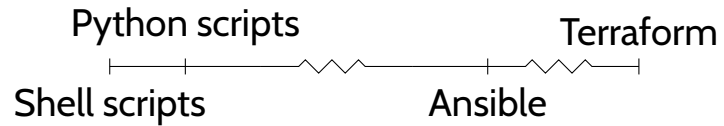
## Definition 2. Infrastructure Code

Code that provisions and manages *infrastructure resources*.

## Definition 3. Infrastructure Resources

Compute resources, networking resources, and storage resources.

## Infrastructure Code



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

```
1  #!/bin/bash
3  SG=$(aws ec2 create-security-group ...)
5  aws ec2 authorize-security-group-ingress --group-id "$SG"
7  INST=$(aws ec2 run-instances --security-group-ids "$SG" \
8      --instance-type t2.micro)
```

Using aws CLI to create EC2 access like the practical.

```
1 import boto3

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

8     data = ec2.authorize_security_group_ingress(...)

10     instance = ec2_client.run_instances(
11         SecurityGroups=[security_group_id],
12         InstanceType="t2.micro",
13         ...
14     )
```

Using aws python library (boto3).

```
1 resource "aws_instance" "hextris-server" {
2     instance_type = "t2.micro"
3     security_groups = [aws_security_group.hextris-server.name]
4     ...
5 }

7 resource "aws_security_group" "hextris-server" {
8     ingress {
9         from_port = 80
10        to_port = 80
11        ...
12    }
13    ...
14 }
```

Finally, terraform.

Question

Notice anything different?

- Prompting for declarative.
- Might notice verbosity.

The *main* difference

## Imperative vs. declarative

IC is heading towards a more declarative paradigm.



## 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...?
- Tendancy to be *declarative*.

Summarising what we've already covered.

Typo?

Infrastructure Code  $\neq$  Infrastructure *as* Code

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

#### Definition 4. 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.

```
1  #!/bin/bash
3  ./download-dependencies
4  ./build-resources
5  cp -r output/* artifacts/
```

```
1  #!/bin/bash
3  ./download-dependencies
4  ./build-resources
5  cp -r output/* artifacts/
```

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

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

```
1 resource "aws_instance" "hextris-server" {  
2     instance_type = "t2.micro"  
3     security_groups = ["sg-6400"]  
4     ...  
5 }
```

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

```
1 resource "aws_instance" "hextris-server" {
2     instance_type = "t2.micro"
3     security_groups = [aws_security_group.hextris-server.name]
4     ...
5 }

7 resource "aws_security_group" "hextris-server" {
8     ingress {
9         from_port = 80
10        to_port = 80
11        ...
12    }
13    ...
14 }
```

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<sup>1</sup> code.
2. Less work.
3. Only one place to update (or verify).

---

<sup>1</sup>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



```
1 func TestTerraformAwsInstance(t *testing.T) {
2     terraformOptions := terraform.WithDefault(t, &terraform.Options{
3         TerraformDir: "../week03/",
4     })
5
6     defer terraform.Destroy(t, terraformOptions)
7     terraform.InitAndApply(t, terraformOptions)
8
9     publicIp := terraform.Output(t, terraformOptions, "public_ip")
10    url := fmt.Sprintf("http://%s:8080", publicIp)
11
12    http_helper.HttpGetWithCustomValidation(t, url, nil, 200,
13        func(code, resp) { code == 200 &&
14            strings.Contains(resp, "hextris")})
15 }
```

```
1 Feature: Define AWS Security Groups

3 Scenario: Only selected ports should be publicly open
4     Given I have AWS Security Group defined
5     When it contains ingress
6     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.