Infrastructure as Code

March 16, 2026

Brae Webb

Presented for the Software Architecture course
at the University of Queensland

THE UNIVERSITY
OF QUEENSLAND
AUSTRALIA

Infrastructure as Code

Software Architecture

March 16, 2026

Brae Webb

1 Introduction

Configuration management can be one of the more challenging aspects of software development. In sophisticated architectures there are roughly two layers of configuration management: machine configuration; and stack configuration.

Machine configuration encompasses software dependencies, operating system configuration, environment variables, and every other aspect of a machine's environment.

Stack configuration encompasses the configuration of an architecture's infrastructure resources. Infrastructure includes compute, storage, and networking resources.

In a sense, machine configuration is a subset of the stack configuration, however, stack configuration tends to be focussed on a higher level of abstraction. In this course, our focus when looking at Infrastructure as Code (IaC) is *stack configuration*. We rely on containerization tools, such as Docker, to fill the hole left by the lack of treatment of machine configuration.

2 Brief History

In the 'Iron Age' of computing, installing a new server was rate limited by how quickly you could shop for and order a new physical machine. Once the machine arrived at your company, some number of weeks after the purchase, someone was tasked with setting it up, configuring it to talk to other machines, and installing all the software it needed. Compared to the weeks it takes to acquire the machine, a day of configuration is not so bad. Furthermore, because so much physical effort was needed to provision a new machine, a single developer would only be responsible for a few machines.

With virtualization one physical machine can be the home of numerous virtual machines. Each one of these machines requires configuration. Now, with new machines available within minutes and no physical labour involved, spending a day of configuring is out of the question — introducing: *infrastructure code*.

3 Infrastructure Code

Infrastructure code arose to ease the burden of the increased complexity where each developer configured and maintained many more machines. Infrastructure code encompasses any code that helps manage our infrastructure. Infrastructure code can often be quite simple. A shell script which installs dependencies is infrastructure code. There's an assortment of infrastructure code out in the world today, ranging from simple shell scripts up to more sophisticated tools such as Ansible and Terraform.

Definition 1. Infrastructure Code

Code that provisions and manages infrastructure resources.

Definition 2. Infrastructure Resources

Compute resources, networking resources, and storage resources.

To explore the range of infrastructure code available, skim the below examples of infrastructure code in BASH, Python, and Terraform. Each snippet performs the same functionality. Infrastructure Code does not have to be strictly tools such as Terraform, although the two are often conflated. One thing which should be noted is that Infrastructure Code specific languages, unlike most traditional software, is tending towards a declarative paradigm.

```bash
#!/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)
```

```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",
        ...
    )
```

```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
        ...
    }
    ...
}
```

4 Infrastructure as Code

In this course, we have chosen to make a distinction between *Infrastructure Code* and *Infrastructure as Code*. We define *Infrastructure Code* as above, as code which manages infrastructure. We separately define *Infrastructure as Code* as the practices of treating Infrastructure Code as if it were traditional code. In the real world no distinction is made. We now introduce *Infrastructure as Code* and clarify why this distinction is important.

Definition 3. Infrastructure as Code

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

Given the above definition of IaC, the natural follow-up question is 'what are good coding practices?'. Before reading further, pause for a moment and consider what coding practices you follow when developing software that you would consider good practice. Got some ideas? Good, let's enumerate a few that we think are important.

4.1 Everything as Code

Everything as Code is a good coding practice which seems so obvious in regular programming that it is rarely discussed. The idea is to ensure that your code does not depend on manual tasks being performed. Unfortunately, as developers are already familiar with their cloud platform's UI, it can be easy to take the easy way and to quickly provision a resource in the UI.

Introducing manual changes outside of your Infrastructure Code creates *configuration drift*, infrastructure configuration which has diverged from your Infrastructure Code. Configuration drift can produce *snowflake* machines, unique machines which your project depends on but no one can safely recreate. Once you introduce just one snowflake machine into your infrastructure, the benefit of reproducibility afforded by IaC is lost.

4.2 Version Control

We hope that everyone can agree that version control in regular code is a luxury we would not wish to live without. It should then be common sense to apply the benefits of version control; restorable code, accountable code, and working collaboratively to Infrastructure Code.

There is however, one significant catch with versioning infrastructure code: *state*. Most Infrastructure Code tools utilize some form of state. In Terraform, applying changing updates a file called *terraform.tfstate*. The state is important to map the resources specified in Infrastructure Code to their real-world counterparts. However, the state is updated frequently and needs to be kept in live-sync between all developers. These requirements make version controlling state a generally bad idea. The solution is counter-intuitively a remote state which exists outside of Terraform controlled infrastructure and out of reach of version control. A remote state might exist on an S3 bucket which all developers can access and update in real time. An non version controlled file such as this might seem like a step backwards, however, if you consider it as a cache sitting between your Infrastructure Code and your infrastructure it is a *slightly* nicer pill to swallow.

4.3 Automation

The goal of Infrastructure Code is to help developers manage the complexity of resource management via automation. In order to maximize our confidence that Infrastructure Code is reflected in reality, it is

important that our build pipelines provide an assurance of consistency.

There are two methods for achieving consistency through automation: fail the pipeline if the Infrastructure Code would change the reality; or make changes to reality in our pipeline. Either of these options would ensure that the checked-in Infrastructure Code is an accurate reflection of reality that developers can safely build atop.

4.4 Code Reuse

A good developer should embody the practice of working smarter not harder — it's the core practice of our industry. When developing Infrastructure Code it is important to remember that you are (likely) not the first to solve this problem. You're not the first person to connect a public EC2 instance to an S3 bucket. A good coding practice that happens to be a win-win for everyone is code reuse. You can't copy and paste another developer's mouse clicks to configure their server via a UI, but you can, and in most cases should, import the Terraform module they published to configure servers in just the way you needed.

4.5 Testing

We'll cover testing in the lecture :)

5 Conclusion

We have looked at Infrastructure Code which enables developers to manage the vast amount of infrastructure resources necessitated by the Cloud Age. Then we explored the benefits of applying the good coding practices we know and love from regular programming to Infrastructure Code. We termed this practice Infrastructure as Code despite the tools (Infrastructure Code) and practices (Infrastructure as Code) being combined in the real world as Infrastructure as Code. It is important to note that IaC is still in relatively early stages, there are a number of issues which need to be addressed, such as: importing existing infrastructure; improving state management; and enabling proper refactoring. Despite these issues, it is the best system we have to date and a significant improvement on manual management.

> Infrastructure as Code is the worst form of infrastructure management except for all those other forms that have been tried from time to time.
>
> – Paraphrased Winston Churchill