about an Ansible or Chef codebase, you need to know the full history of every change that has ever happened.

Procedural code limits reusability

The reusability of procedural code is inherently limited because you must manually take into account the current state of the infrastructure. Because that state is constantly changing, code you used a week ago might no longer be usable because it was designed to modify a state of your infrastructure that no longer exists. As a result, procedural codebases tend to grow large and complicated over time.

With Terraform's declarative approach, the code always represents the latest state of your infrastructure. At a glance, you can determine what's currently deployed and how it's configured, without having to worry about history or timing. This also makes it easy to create reusable code, since you don't need to manually account for the current state of the world. Instead, you just focus on describing your desired state, and Terraform figures out how to get from one state to the other automatically. As a result, Terraform codebases tend to stay small and easy to understand.

General-Purpose Language Versus Domain-Specific Language

Chef and Pulumi allow you to use a *general-purpose programming language* (GPL) to manage infrastructure as code: Chef supports Ruby; Pulumi supports a wide variety of GPLs, including JavaScript, TypeScript, Python, Go, C#, Java, and others. Terraform, Puppet, Ansible, CloudFormation, and OpenStack Heat each use a *domain-specific language* (DSL) to manage infrastructure as code: Terraform uses HCL; Puppet uses Puppet Language; Ansible, CloudFormation, and OpenStack Heat use YAML (CloudFormation also supports JSON).

The distinction between GPLs and DSLs is not entirely clear-cut—it's more of a helpful mental model than a clean, separate categorization—but the basic idea is that DSLs are designed for use in one specific domain,

whereas GPLs can be used across a broad range of domains. For example, the HCL code you write for Terraform works only with Terraform and is limited solely to the functionality supported by Terraform, such as deploying infrastructure. This is in contrast to using a GPL such as JavaScript with Pulumi, where the code you write can not only manage infrastructure using Pulumi libraries but also perform almost any other programming task you wish, such as run a web app (in fact, Pulumi offers an Automation API you can use to embed Pulumi within your application code), perform complicated control logic (loops, conditionals, and abstraction are all easier to do in a GPL than a DSL), run various validations and tests, integrate with other tools and APIs, and so on.

DSLs have several advantages over GPLs:

Easier to learn

Since DSLs, by design, deal with just one domain, they tend to be smaller and simpler languages than GPLs and therefore are easier to learn than GPLs. Most developers will be able to learn Terraform faster than, say, Java.

Clearer and more concise

Since DSLs are designed for one specific purpose, with all the keywords in the language built to do that one thing, code written in DSLs tends to be easier to understand and more concise than code written to do the exact same thing but written in a GPL. The code to deploy a single server in AWS is usually going to be shorter and easier to understand in Terraform than in Java.

More uniform

Most DSLs are limited in what they allow you to do. This has some drawbacks, as I'll mention shortly, but one of the advantages is that code written in DSLs typically uses a uniform, predictable structure, so it's easier to navigate and understand than code written in GPLs, where every developer might solve the same problem in a completely different