way. There's really only one way to deploy a server in AWS using Terraform; there are hundreds of ways to do the same thing with Java.

GPLs also have several advantages over DSLs:

Possibly no need to learn anything new

Since GPLs are used in many domains, there's a chance you might not have to learn a new language at all. This is especially true of Pulumi, as it supports several of the most popular languages in the world, including JavaScript, Python, and Java. If you already know Java, you'll be able to jump into Pulumi faster than if you had to learn HCL to use Terraform.

Bigger ecosystem and more mature tooling

Since GPLs are used in many domains, they have far bigger communities and much more mature tooling than a typical DSL. The number and quality of Integrated Development Environments (IDEs), libraries, patterns, testing tools, and so on for Java vastly exceeds what's available for Terraform.

More power

GPLs, by design, can be used to do almost any programming task, so they offer much more power and functionality than DSLs. Certain tasks, such as control logic (loops and conditionals), automated testing, code reuse, abstraction, and integration with other tools, are far easier with Java than with Terraform.

Master Versus Masterless

By default, Chef and Puppet require that you run a *master server* for storing the state of your infrastructure and distributing updates. Every time you want to update something in your infrastructure, you use a client (e.g., a command-line tool) to issue new commands to the master server, and the master server either pushes the updates out to all of the other servers or those servers pull the latest updates down from the master server on a regular basis.

A master server offers a few advantages. First, it’s a single, central place where you can see and manage the status of your infrastructure. Many configuration management tools even provide a web interface (e.g., the Chef Console, Puppet Enterprise Console) for the master server to make it easier to see what’s going on. Second, some master servers can run continuously in the background and enforce your configuration. That way, if someone makes a manual change on a server, the master server can revert that change to prevent configuration drift.

However, having to run a master server has some serious drawbacks:

Extra infrastructure

> You need to deploy an extra server, or even a cluster of extra servers (for high availability and scalability), just to run the master.

Maintenance

> You need to maintain, upgrade, back up, monitor, and scale the master server(s).

Security

> You need to provide a way for the client to communicate to the master server(s) and a way for the master server(s) to communicate with all the other servers, which typically means opening extra ports and configuring extra authentication systems, all of which increases your surface area to attackers.

Chef and Puppet do have varying levels of support for masterless modes where you run just their agent software on each of your servers, typically on a periodic schedule (e.g., a cron job that runs every five minutes), and use that to pull down the latest updates from version control (rather than from a master server). This significantly reduces the number of moving parts, but, as I discuss in the next section, this still leaves a number of unanswered