You’ll see an example of this approach in Chapter 7.

Conclusion

Putting it all together, Table 1-4 shows how the most popular IaC tools stack up. Note that this table shows the *default* or *most common* way the various IaC tools are used, though as discussed earlier in this chapter, these IaC tools are flexible enough to be used in other configurations, too (e.g., you can use Chef without a master, you can use Puppet to do immutable infrastructure, etc.).

Table 1-4. A comparison of the most common ways to use the most popular I

	Chef	Puppet	Ansible	Pulumi
Source	Open	Open	Open	Open
Cloud	All	All	All	All
Type	Config mgmt	Config mgmt	Config mgmt	Provisioning
Infra	Mutable	Mutable	Mutable	Immutable
Paradigm	Procedural	Declarative	Procedural	Declarative
Language	GPL	DSL	DSL	GPL
Master	Yes	Yes	No	No
Agent	Yes	Yes	No	No
Paid Service	Optional	Optional	Optional	Must-have
Community	Large	Large	Huge	Small
Maturity	High	High	Medium	Low

At Gruntwork, what we wanted was an open source, cloud-agnostic provisioning tool with a large community, a mature codebase, and support for immutable infrastructure, a declarative language, a masterless and agentless architecture, and an optional paid service. Table 1-4 shows that Terraform, although not perfect, comes the closest to meeting all of our criteria.

Does Terraform fit your criteria? If so, head over to Chapter 2 to learn how to use it.

---

1 From *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations* (IT Revolution Press, 2016) by Gene Kim, Jez Humble, Patrick Debois, and John Willis.

2 On most modern operating systems, code runs in one of two "spaces": *kernel space* or *user space*. Code running in kernel space has direct, unrestricted access to all of the hardware. There are no security restrictions (i.e., you can execute any CPU instruction, access any part of the hard drive, write to any address in memory) or safety restrictions (e.g., a crash in kernel space will typically crash the entire computer), so kernel space is generally reserved for the lowest-level, most trusted functions of the OS (typically called the *kernel*). Code running in user space does not have any direct access to the hardware and must use APIs exposed by the OS kernel instead. These APIs can enforce security restrictions (e.g., user permissions) and safety (e.g., a crash in a user space app typically affects only that app), so just about all application code runs in user space.

3 As a general rule, containers provide isolation that's good enough to run your own code, but if you need to run third-party code (e.g., you're building your own cloud provider) that might actively be performing malicious actions, you'll want the increased isolation guarantees of a VM.

4 This is where the term *bus factor* comes from: your team's bus factor is the number of people you can lose (e.g., because they got hit by a bus) before you can no longer operate your business. You never want to have a bus factor of 1.

5 Check out the Gruntwork Infrastructure as Code Library for an example.

6 Docker, Packer, and Kubernetes are not part of the comparison, because they can be used with any of the configuration management or provisioning tools.

7 The data on contributors and stars comes from the open source repositories (mostly GitHub) for each tool. Because CloudFormation is closed source, this information is not available.