Changes from the Second Edition to the Third Edition

The first edition of this book came out in 2017, the second edition came out in 2019, and although it's hard for me to believe it, I'm now working on the third edition in 2022. Time flies. It's remarkable how much has changed over the years!

If you've read the second edition of the book and want to know what's new, or if you're just curious to see how Terraform has evolved between 2019 and 2022, here are some of the highlights of what changed between the second and third editions:

Hundreds of pages of updated content

The third edition of the book is about a hundred pages longer than the second edition. I also estimate that roughly one-third to one-half of the pages originally in the second edition were updated as well. Why so much churn? Well, Terraform went through six major releases since the second edition came out: 0.13, 0.14, 0.15, 1.0, 1.1, and 1.2. Moreover, many Terraform providers went through major upgrades of their own, including the AWS Provider, which was at version 2 when the second edition came out and is now at version 4. Plus, the Terraform community has seen massive growth over the last few years, which has led to the emergence of many new best practices, tools, and modules. I've tried to capture as much of this change as I could in the third edition, adding two completely new chapters and making major updates to all the existing chapters, as described next.

New provider functionality

Terraform has significantly improved how you work with providers. In the third edition, I've added an entirely new chapter, Chapter 7, that describes how to work with multiple providers: e.g., how to deploy into multiple regions, multiple accounts, and multiple clouds. Also, by popular demand, this chapter includes a brand-new set of examples showing how to use Terraform, Kubernetes, Docker, AWS, and EKS to run containerized apps. Finally, I've also updated all the other chapters to highlight new provider features from the last several releases, including the `required_providers` block introduced in Terraform 0.13, the lock file introduced in Terraform 0.14, and the `configuration_aliases` parameter introduced in Terraform 0.15.

Better secrets management

When using Terraform code, you often have to deal with many types of secrets: database passwords, API keys, cloud provider credentials, TLS certificates, and so on. In the third edition, I added an entirely new chapter, Chapter 6, dedicated to this topic, including a comparison of common secret management tools, as well as lots of new example code that shows a variety of techniques for securely using secrets with Terraform, including environment variables, encrypted files, centralized secret stores, IAM roles, OIDC, and more.

New module functionality

Terraform 0.13 added the ability to use `count`, `for_each`, and `depends_on` on `module` blocks, making modules considerably more powerful, flexible, and reusable. You can find examples of how to use these new features in Chapters 5 and 7.

New validation functionality

In Chapter 8, I've added examples of how to use the `validation` feature introduced in Terraform 0.13 to perform basic checks on variables (such as enforcing minimum or maximum values) and the `precondition` and `postcondition` features introduced in Terraform 1.2 to perform basic checks on resources and data sources, either before running `apply` (such as enforcing that the AMI a user selected uses the x86_64 architecture) or after running `apply` (such as checking that the EBS volume you're using was successfully encrypted). In Chapter 6, I show how to use the `sensitive` parameter introduced in Terraform 0.14 and 0.15, which ensures that secrets won't be logged when you run `plan` or `apply`.

New refactoring functionality

Terraform 1.1 introduced the `moved` block, which provides a much better way to handle certain types of refactoring, such as renaming a resource. In the past, this type of refactoring required users to manually run error-prone `terraform state mv` operations, whereas now, as you'll see in a new example in Chapter 5, this process can be fully automated, making upgrades safer and more compatible.

More testing options

The tools available for automated testing of Terraform code continue to improve. In Chapter 9, I've added example code and comparisons of static analysis tools for Terraform, including `tfsec`, `tflint`, `terrascan`, and the `validate` command; `plan` testing tools for Terraform, including Terratest, OPA, and Sentinel; and server testing tools, including `inspec`, `serverspec`, and `goss`. I also added a comparison of all the testing approaches out there, so you can pick the best ones for your use cases.

Improved stability

Terraform 1.0 was a big milestone for Terraform, not only signifying that the tool had reached a certain level of maturity but also coming with a number of compatibility promises. Namely, there is a promise that all the 1.x releases will be backward compatible, so upgrading between v1.x releases should no longer require changes to your code, workflows, or state files. Terraform state files are now cross-compatible with Terraform 0.14, 0.15, and all 1.x releases, and Terraform remote state data sources are cross-compatible with Terraform 0.12.30, 0.13.6, 0.14.0, 0.15.0, and all 1.x releases. I've also updated Chapter 8 with examples of how to better manage versioning of Terraform (including