at test time from several hundred to just a handful so that these tests will be faster and less brittle.

Moreover, this approach to end-to-end testing more closely mimics how you'll be deploying those changes in production. After all, it's not like you tear down and bring up your production environment from scratch to roll out each change. Instead, you apply each change incrementally, so this style of end-to-end testing offers a huge advantage: you can test not only that your infrastructure works correctly but also that the *deployment process* for that infrastructure works correctly, too.

Other Testing Approaches

Most of this chapter has focused on testing your Terraform code by doing a full `apply` and `destroy` cycle. This is the gold standard of testing, but there are three other types of automated tests you can use:

- Static analysis
- Plan testing
- Server testing

Just as unit, integration, and end-to-end tests each catch different types of bugs, each of the testing approaches just mentioned will catch different types of bugs as well, so you'll most likely want to use several of these techniques together to get the best results. Let's go through these new categories one at a time.

Static analysis

Static analysis is the most basic way to test your Terraform code: you parse the code and analyze it without actually executing it in any way. Table 9-1 shows some of the tools in this space that work with Terraform and how they compare in terms of popularity and maturity, based on stats I gathered from GitHub in February 2022.