CASE STUDY

Mozilla uses Terraform and Atlas by HashiCorp to embrace infrastructure as code

500	1	99%
ENGINEERING HOURS SAVED ANNUALLY	DAY OF ENGINEERING TIME TO SETUP	LESS MANUAL OPERATIONS ACTIONS

"We initially rolled our own solution for infrastructure CI and deployment. As the team grew, our homebrew solution struggled to centrally manage configuration, Terraform state, and access control," said Chris Lonnen, Internal Systems Architect at Mozilla. "Atlas remedies all of these challenges for us and provides a beautiful interface for audit logs, infrastructure history, systemwide monitoring, and more. We've never lost 9s betting on a HashiCorp product."

Mozilla's Terraform configurations are **available on GitHub**.

Cast & Crew

Chris Lonnen
Internal Systems Architect

Daniel Maher
Devops Engineer

Robert Helmer
Senior Staff Engineer

Responsible operations and infrastructure as code

Infrastructure failures follow a common pattern. Usually they include a manual step that was executed improperly: a wrong software package update, a one-off deploy, or a web GUI command with insufficient change control.

In addition to the risk of manual steps, most infrastructure change management lacks a consistent peer review process. As an industry, we've embraced peer review for application code and have reaped significant improvements in quality and maintainability, but our infrastructure management tooling has yet to embrace the same practice. With the web GUI mentality of many infrastructure management options, the 'click here to immediately apply your change that only you can see' button has laid waste to many an infrastructure.

With the adoption of infrastructure-as-code tools in Mozilla's current generation of tooling, we've made significant progress. Terraform has been a transformative technology for the three DevOps engineers on Mozilla's Tools and Services team. We now embrace many software engineering best practices with our infrastructure code — versioning, repeatability, and collaboration. However, even with this new, saner methodology, we still faced a friction point when it came to applying changes to running environments.

The importance of the Terraform statefile

Terraform has two key aspects. The first is a statefile which defines the current state of infrastructure as Terraform sees it. The second is a configuration file which declares the desired state of infrastructure. Terraform calculates the difference between the configuration file and statefile, then runs the necessary set of Amazon Web Services (AWS) API commands to get the current infrastructure to update to the desired configuration. When multiple engineers are working on the same infrastructure, it's extremely important for those engineers to be working off of the same Terraform state. Otherwise, merge conflicts could occur.

We initially tried to solve storing state centrally with Jenkins and S3 to persist our statefile. Inconsistencies in our home-grown setup became a weekly affair, eating up several hours of engineering time and clogging the deployment pipeline for developers and devops until we could rebuild or reconcile the system.

Using Atlas to manage Terraform state in a central, protected location

Prior to Mozilla using Atlas, we used Travis-CI to run a terraform plan on every pull request, which was part of the code review. Sometimes the state of the world would change out from under us as we juggled multiple pull requests. Jenkins would listen for updates to master, and also run a terraform plan to detail what changes would actually get made immediately before applying. The details of the output would end up buried in a few thousand lines of other build and deploy logs, and we would frequently miss important details. If someone was watching and the output of the plan didn't look right, we had a very tiny window to fix the issue and it was very high risk to interrupt the process.

One of our biggest concerns when moving to infrastructure-as-code was the declarative nature of the tools. No one wants a seemingly benign change to drop a stateful production service. This risk went up with the number of pull requests in flight, because the output of terraform plan was generated against the state of the world when the change was proposed. The state of the world would change, the pull request would land, and suddenly the changes are being applied in a new and potentially unsafe context.

When Atlas was announced we quickly saw it could solve a number of these problems. Atlas manages a queue of all the changes, and waits for manual sign off on each plan to ensure each plan is representative. This prevents a stampede of updates from introducing a risky change wrapped in a stale terraform plan output. It has also eliminated our weekly statefile sync issues, which alone has saved us several hours a week.

Operational transparency with Terraform and Atlas

Terraform and Atlas represent a significant improvement to transparency and readability over our previous setup. Environment variables for applying infrastructure changes are stored securely in Atlas, rather than in an S3 bucket as we had them previously. It's easier to update those variables in Atlas, and all the credentials are handled in an encrypted form. There is no pull-request-like signoff mechanism for changing variables, but there is an audit log which tracks changes.

QA, developers, and managers have all applauded the improved understandability of Atlas over our previous setup. With Atlas managing secrets, there's no longer a need for sensitive credentials on multiple machines. The UI is a significant improvement over the patchwork GUIs and command line tools needed to get the same information before. We haven't needed the audit trails yet (thankfully), but they have proven useful for individuals who want to catch up with the recent history and the iterative state of our environments.

We've seen incredible benefits of Atlas over our homebrew setup. It's saving us an ongoing four to ten hours per week, and it took less than a day of engineering time to set up. Atlas has eliminated high-risk manual work, made our system more transparent and understandable to its users, and improved overall infrastructure stability by helping us avoid common infrastructure failure scenarios.