weaveworks

CASE STUDY

INDUSTRY: Technology Services

LOCATION: Global Presence

HIGHLIGHTS

+75%
Deployment speed

+50%
Deployment frequency

+50%
Developer productivity

KEY BENEFITS

- Greater developer autonomy and alignment using a self-service platform
- Faster release with end-to-end deployment automation
- Incorporated and enforced security policies using OPA to prevent the risk of misconfiguration
- Tremendous cost savings with policies that identified poorly configured resources
- GitOps to improve Terraform Plan best practices
- Comprehensive visibility into the development pipeline
- Easy error detection and rapid remediation

AppsFlyer Improves Developer Productivity by 50% by combining GitOps with Terraform IaC

The customer

AppsFlyer, the market leader in mobile marketing and analytics industry, caters to over 14,000 customers. Besides 1000 microservices, its architecture operates over 250,000 cloud resources, including EC2 instances and EKS clusters. The AppsFlyer Engineering team of over 400 engineers is organized into smaller groups called Squads, with each Squad having complete autonomy.

Challenges

Manual and time-consuming deployment process

AppsFlyer's developers were dealing with productivity slowing down, as they had to navigate several manual steps and interact with multiple systems for deployments. A typical deployment process started with a commit to Git, which invoked CI flow in Jenkins. Testing then takes place, in a tool built internally.

Once a build is approved and validated by the team, a developer accesses another internal tool that deploys the artifact to hosts. An additional internal tool is then used to operate and view (through dashboards) the deployed system. System performance tracking and monitoring is done from a different system as well.

For what is seemingly a simple developer task - committing code - there were many tasks and systems involved. From reviews and validations by concerned team members to the use of various internal systems - all requiring human intervention and monitoring.

Developer autonomy was needed, but with greater alignment

Although the AppsFlyer's Platform groups primary focus was to enhance developer experience, their approach turned out to be affecting productivity. With a siloed development approach, AppsFlyer's developers found it difficult to be synced despite working towards the same goal. Having grown so quickly, their operational model process had soon become chaotic, and made it difficult for the platform team to collect feedback from developers. The platform team was often unaligned as to what each developer was working on, and if the developers were able to function at optimum levels. They wanted to transition towards a 'high autonomy, high alignment' model that enabled greater collaboration between developers, and more transparency on developer tasks.

Risk of the platform becoming the developer's bottleneck

The engineering department had adopted the platform-model for provisioning resources. According to this model, an internal platform team delivers self-service APIs, tools, services, knowledge and support to developers which they use for resource provisioning (**read more in this blog post**). However, this platform was hindering development-related tasks. After all, if the platform team were busy doing something else, developers were unable to use the required infrastructure for their services or complete their tasks.

Operational challenges around Terraform

Most of their operations happen through Terraform, a popular multi-cloud IaC framework, which they used to build and manage their platform with. However, Terraform had its own challenges. Engineers needed shared access to the state file and knowledge of existing resources to update the infrastructure. But having access to the same file can lead to data loss, accidental errors, and other forms of file corruption. Moreover, the data stored in a state file is usually in plain text, which presents security issues in a version control system.

Post production security guardrails

Cloud resource misconfigurations, coupled with manual deployments, can increase the risks of supply chain attacks, especially where Terraform modules are concerned. Developing and managing a platform is a complex endeavor, both for the security and platform team, and misconfiguring cloud resources is undesirable. To reduce the risks of misconfigured resources making it into the cloud, a more proactive approach to securing the infrastructure was needed.

Developers had to check and pull data from different tools or solutions to keep track of infrastructure changes. It took time to correlate between changes in the infrastructure and system degradations, resulting in longer root cause analysis and sub- optimal mean time to recovery (MTTR).

Solution

GitOps - A new operational model

AppsFlyer Platform's Cloud Native team, led by Eliran Bivas, enabled the adoption of GitOps to secure developer alignment, and improve AppsFlyer's resilience. This would greatly improve developer productivity, and ensure security guardrails at every step of the software delivery process.

Being an organization that only did CI (continuous integration), they decided to start small and adopt GitOps for about 20 platform services. Teams underwent comprehensive training to understand GitOps concepts, self-serve flow, and the pitfalls to avoid along the way.

After the teams started adopting GitOps for application development, they witnessed a significant boost in productivity. Their mean time to deployment (MTD) increased by 50-75%, and deployment frequency was up by 20-50%. This was due to the wholly automated workflows and infrastructure migration to GitOps, which made management easy and saved time.

> "GitOps isn't for everyone. It is the best fit for organizations with a need to reduce the development cycle times and also have the ability to deploy code and infrastructure constantly. GitOps allows you to ease the operational cost of managing large-scale infrastructure by adopting a pattern that continually validates your desired state."
>
> **— Eliran Bivas, Cloud Native Leader, AppsFlyer**

An innovative approach of extending GitOps over Terraform

AppsFlyer has been using Terraform to manage various parts of their stack - infrastructure provisioning, networking, and monitoring with tools like EKS and PagerDuty. After an exhaustive search for a solution that could act as a gateway between Terraform and GitOps, they opted to build a unique solution that integrates with **Flux**, a CNCF open source project created and maintained by Weaveworks. It was primarily due to its great extensibility and the industry-wide adoption it has received. Flux is run at massive scale in production by many **organizations** including Weaveworks.

AppsFlyer believed that extending GitOps to support Terraform would ensure seamless collaboration between developers and platform operators. It would automate code commits using Terraform. Once developers merged code into GitLab, AppsFlyer created Kube-controllers managing Terraform lifecycle and Flux applied the changes made in Git to either a Kubernetes cluster, a SaaS integration, or a cloud platform.

With GitOps, their developers achieved greater autonomy through a self-service platform which eliminated interference between different teams. It provided access to pre-approved and ready-made templates for services such as Kafka, Druid, and Airflow. These are some of their most frequently-used services, and the list of available templates is growing by the day. The self-service platform allows developers to select a template and provision the exact resources in that template to run their applications. The provisioning is done by Terraform, but the template is configured and managed in Git.

The practice of versioning enabled tracking of every change made to the codebase. Further, the fully automated process to implement changes and roll them back in case of failures gave developers complete control to deal with accidents or mishaps. They also implemented policies within GitOps workflows, proofing their systems against security and compliance risks.

It started with migrating 20 platform services and infrastructure to GitOps; today they manage more than 100 microservices in GitOps workflows and over 50 clusters operating data-intensive workloads in services like Kafka.

> “The main reason we chose FluxCD is that it is much more extendable than the alternatives. We looked at it as a “batteries included but a replaceable solution” that fit our architecture needs. We felt that AppsFlyer and Weaveworks could collaborate towards creating a shared vision for the future of the GitOps community.”
>
> **— Eliran Bivas, Cloud Native Leader, AppsFlyer**

This process helped the organization achieve the speed they so desired while improving security and developer experience.

Results

1 Greater developer autonomy

Implementing GitOps allowed AppsFlyer engineers to enjoy a self-service experience as they can now create resources using the pre-approved templates and configurations. This improved their productivity by eliminating significant delays due to back-and-forths with the platform team. It accelerated the software development process through developer autonomy. Today, it takes just minutes to provision new resources that otherwise used to take days or weeks using the old model. This is a key reason for the mean time to deployment increasing to 50-75%.

> “Infrastructure migrated to GitOps makes it a lot easier to maintain, saving us a lot of time to work on new features and plans. If you’re looking for a way to streamline your development life cycle - GitOps is a great approach for your organization”.
>
> **— Eliran Bivas, Cloud Native Leader, AppsFlyer**

With GitOps, you can manage your entire development pipeline in a declarative way defining the infrastructure, networking, storage, and more within Git. The GitOps agent ensures that the system state in production always stays as declared in Git. This is managed by the GitOps automatic reconciliation capability, and its ability to notice drift in production. In the case of a drift from the declared state, the GitOps agent automatically reverts to the original declared state in Git.

2

GitOps facilitates fully automated deployments

GitOps facilitated deployment automation - every change committed to Git, if it passes all checks, is automatically pushed to production, ensuring that the desired state in Git is implemented in production. With GitOps, the team automated various platform services, including Kafka, Airflow, Kubernetes, RDS, DNS registration, Kubernetes deployments, HashiCorp's Vault policies, and more. This greatly contributes to the positive results they see with a 20-50% increase in deployment frequency.

Having a reliable, repeatable, and secure deployment process allows development teams to move faster, and reduces the risk of failed deployments. With deployment issues out of the way, developers can spend more time on higher priority tasks, and writing more code than managing operational issues. The teams spend much less time than before (as low as 20%) on operational tasks.

In case of any issues, the GitOps agent would automatically roll back the change. Further, as everyone was aware of what changes were made to the code and by whom, it enabled developers to trace issues to their source and fix them. This information eased root cause analysis.

3

Boosting developer productivity

The fully automated workflow allowed developers to focus on crucial tasks instead of toggling between writing code and tinkering with infrastructure. Developers have reduced the time spent on operational tasks by 50%. This level of focus is essential for teams to function at peak levels, and be able to make the most of a streamlined, high-velocity software delivery pipeline.

4

Easy to implement security and compliance policies

AppsFlyer secured its system by putting in place guardrails to implement policy-as-code using OPA (Open Policy Agent). Further, the developers included security and governance policies within Git to enforce best practices. This significantly reduced the risk of flawed configurations. With policies enforced, the platform team was quick to review infrastructure changes and the cognitive load, due mostly to frequent context switching, was reduced. The team has implemented policies that have identified poorly configured resources, saving the team thousands of dollars.

By embedding security guardrails and enforcing community policies using OPA, the AppsFlyer team were able to reap the benefits of shifting left. Poorly misconfigured resources were flagged early in the development lifecycle, saving time and resources. In addition, the platform team was able to take concrete steps in maximizing developer productivity while ensuring infrastructure resilience and compliance.

With the team using GitOps along with Terraform to deploy services and provision Kubernetes clusters automatically, the recovery in case of an incident happens quickly. All it takes is rolling back to a previous state or version in Git.