Dagger

Building Grafana for Multiple Architectures in 8 Minutes or Less

Grafana

Industry
Analytics & monitoring solution

Website
grafana.com

> ***With Dagger, we were able to go from something that takes an hour to something that takes 8 minutes."***
>
> Kevin Minehart
> Grafana Labs

We're always interested in hearing how our community is using Dagger – their use cases, their challenges, and their experiences with deploying Dagger in different environments. Our Discord is a great place to find these stories, and to benefit from the knowledge and experience of the Dagger community.

In this blog post, we'll share the story of Dagger community member Kevin Minehart (aka @Kevin on Discord). Kevin, a software engineer at Grafana Labs, uses Dagger to create bespoke Grafana packages for end-users. Previously, Kevin built every package using a remote CI engine, which was a time-consuming process. With Dagger, Kevin is able to build packages locally in 8 minutes or less...a build-time improvement of ~75%!

The problem: Time-consuming, remote-only builds

Kevin and his team are often asked to deliver bespoke, pre-packaged versions of Grafana to potential customers. These packages share the same application codebase, but are customized to end-user requirements - for example, packages with support for multiple architectures, specific feature flags or pre-activated plugins. Before switching to Dagger, they built these packages using a standalone Go program (migrated from Bash scripts) containing multiple sub-commands.

There were a few challenges with this approach:

- Building an application package required multiple Git operations to be performed in a specific sequence. This was time-consuming and tedious, requiring manual coordination and control by an engineer.

> ***Whenever I had to build Grafana for one or two different versions, it consumed most of my day."***

> ***The caching has already proven to be a really big performance increase for us, especially whenever we're doing a lot of the same things locally."***

> ***Dagger being open source was very helpful. I discovered quite a few functions and things that I should have been using in different places just by inspecting the code or looking at the Go docs."***

The Go program and sub-commands were themselves opaque, with limited documentation and visibility into how they worked. Engineers had to inspect the code to understand what each command did.

- Although the packaging code was written in Go, it could not run locally due to highly-specific environment requirements. In practice, the team could only run it in a customized Docker container on their CI engine.

The solution: Fast, consistent local and remote builds with Dagger

Kevin had heard about Dagger previously and had experimented with it in some personal projects. As he was already familiar with it, he decided to try and automate the Grafana packaging process using Dagger's Go SDK. With Dagger, he was able to consolidate all the packaging tasks into a standalone Dagger pipeline that could be invoked with a single command.

After trying the new approach for a month, the team concluded that using Dagger offered numerous advantages over the previous system. Key benefits are

- Dagger's built-in caching produces a significant increase in performance. The previous packaging process would, very often, take an hour or more; the Dagger pipeline typically delivers its results in 8 minutes.
- Dagger's multi-platform support allows the team to quickly and consistently build Grafana packages for multiple architectures and platforms: Linux (5), Mac OS (2) and Windows.
- Dagger's built-in caching produces a significant increase in performance. The previous packaging process would, very often, take an hour or more; the Dagger pipeline typically delivers its results in 8 minutes.
- Dagger's multi-platform support allows the team to quickly and consistently build Grafana packages for multiple architectures and platforms: Linux (5), Mac OS (2) and Windows.
- The same Dagger pipeline works consistently locally and in remote CI runners, eliminating much of the FUD of the previous process. With Dagger, every member of the team can build and test packages locally, despite having diverse development environments.

> ***I don't think I can emphasize enough the value that Dagger has given us. We were not able to even build a Grafana package locally before this. So Dagger was a huge win for us."***

- The Dagger pipeline is invoked as a single process which contains all the business logic (including Git operations) necessary to produce a final package. This means that an engineer can trigger it and come back later to collect the result - no manual intervention or ongoing attention required.
- Dagger's Go SDK makes it easier for the team of mostly Go engineers to continuously improve the pipeline by adding tests, reusing Go modules and adopting existing best practices.

The future: Using Dagger for all Grafana packaging processes

Going forward, Kevin plans to add service containers to his team's Dagger pipeline for integration testing. He is also hoping to use Dagger's support for remote caching when it becomes available. In the near future, he plans to start using Dagger to build Grafana automatically, and then replace the existing packaging processes with the new Dagger pipeline.