

It's hard to be here at KubeCon without hearing the word "observability"-- a word that we almost never use in any other context but somehow, when we're talking about complex and modern computer systems, we all want to observe. We're all voyeurs. We don't just want to watch-- we want to watch the watchers. But I wonder sometimes if we're missing the point.



Here's a real example of something that happened in production. ==More explanation of graph==

In some ways, this was a success. We had everything instrumented. Our stack was observable. So why wasn't this enough? Why was it that this problem was not a once-off but a recurring problem that we, with our highly observable everything, couldn't resolve for years?

Well, it's because observability isn't enough. Sure, it's observable, but when you have complex systems, you don't know *where* to observe. You don't know what to look at. And being observable says nothing about getting to the root cause of issues, much less fixing them.

So if our goal is simply "observability", it's a pretty limited one. Observability is not the end quality we want our systems to have; it's an incidental one. The goal should be continuous reliability. Observability is just *one of* the means to that end.



Anyone who's played a fantasy RPG knows that to get the good weapons, you need to go to a blacksmith. Blacksmiths have forges. A forge consists of some sort of furnace where metal is heated to make it malleable and hammered into shape. Forges let you make durable weapons, but they require you to put them through fire first.

Image from: Green Beetle, Forging A Sword Pt 1: Every Stroke:  
<https://www.youtube.com/watch?v=ha1PwOzuo4k>

# The Continuous Reliability FORGE



- Framework
- Observability
- Recovery
- Growth
- Engagement

Continuous reliability is the process of putting our system through the FORGE.

I'll give examples for how we're doing all of these at Grafana, but first, maybe I should give you a summary of who we even are.

## The Grafana stack



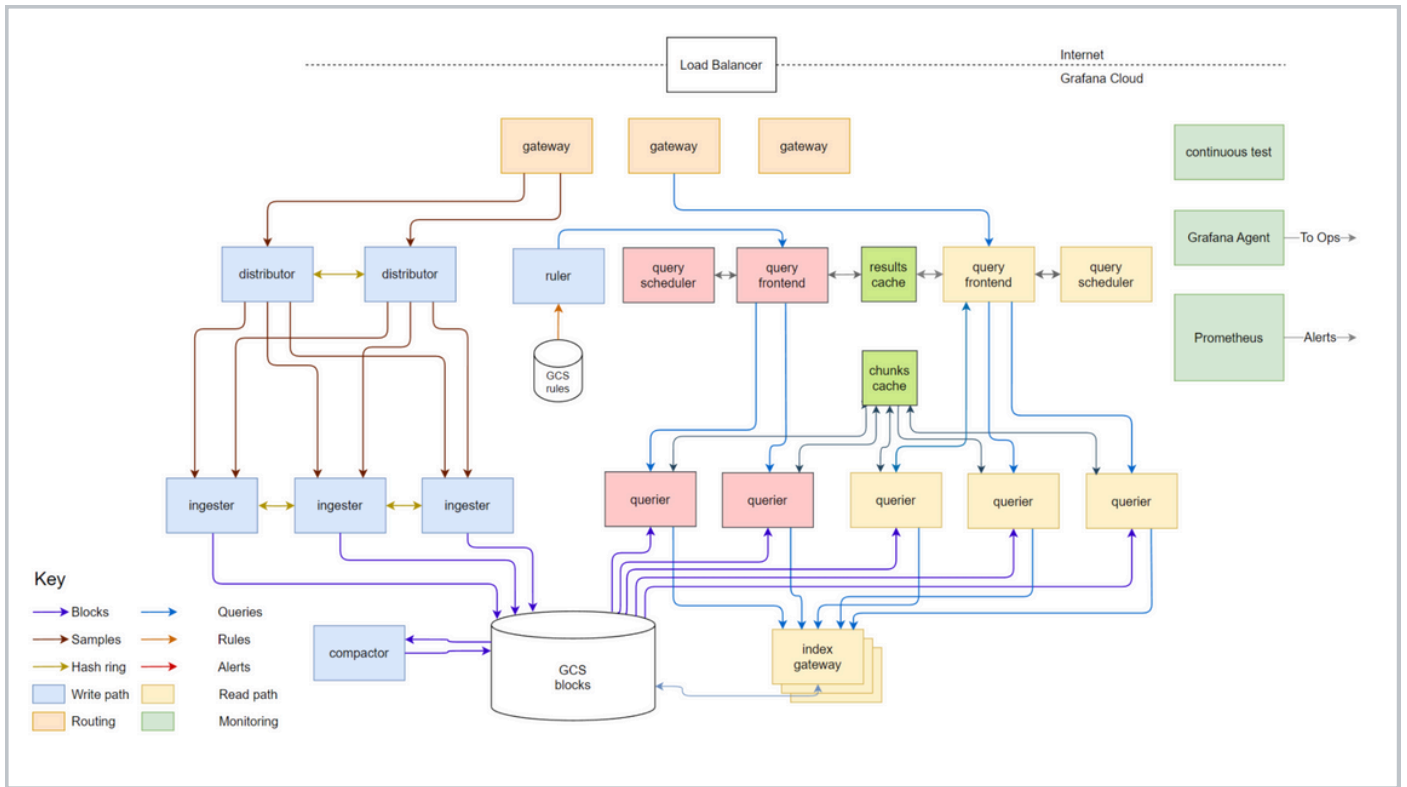
Grafana is more than just Grafana

Everything here is open source, but we also have hosted versions of these, on Grafana Cloud. We maintain some combination of this stack for people so that they don't have to.

# Framework



Resilient infrastructure design



(7m)

Example of Loki's microservices architecture  
 - 324 TB of logs a day

# Grafana Enterprise Metrics

## How we responded to a 2-hour outage in our Grafana Cloud Hosted Prometheus service



Tom Wilkie · 2021-03-26 · 4 min



- We have about 15 large Kubernetes clusters of GEM around the world
  - Clusters are multi-tenanted
  - We set limits for how much data and how quickly each customer can send, but a bug in limits occurred
  - Turns out that handling the initial error was more CPU-intensive than the happy path, and it caused a cascading failure
- Solutions:
- fix limits bug
  - recreate overload scenario through k6 (we used it to test Mimir to hold 1.3 billion time series for metrics)

<https://grafana.com/blog/2021/03/26/how-we-responded-to-a-2-hour-outage-in-our-grafana-cloud-hosted-prometheus-service/>



# Observability



## Instrumentation

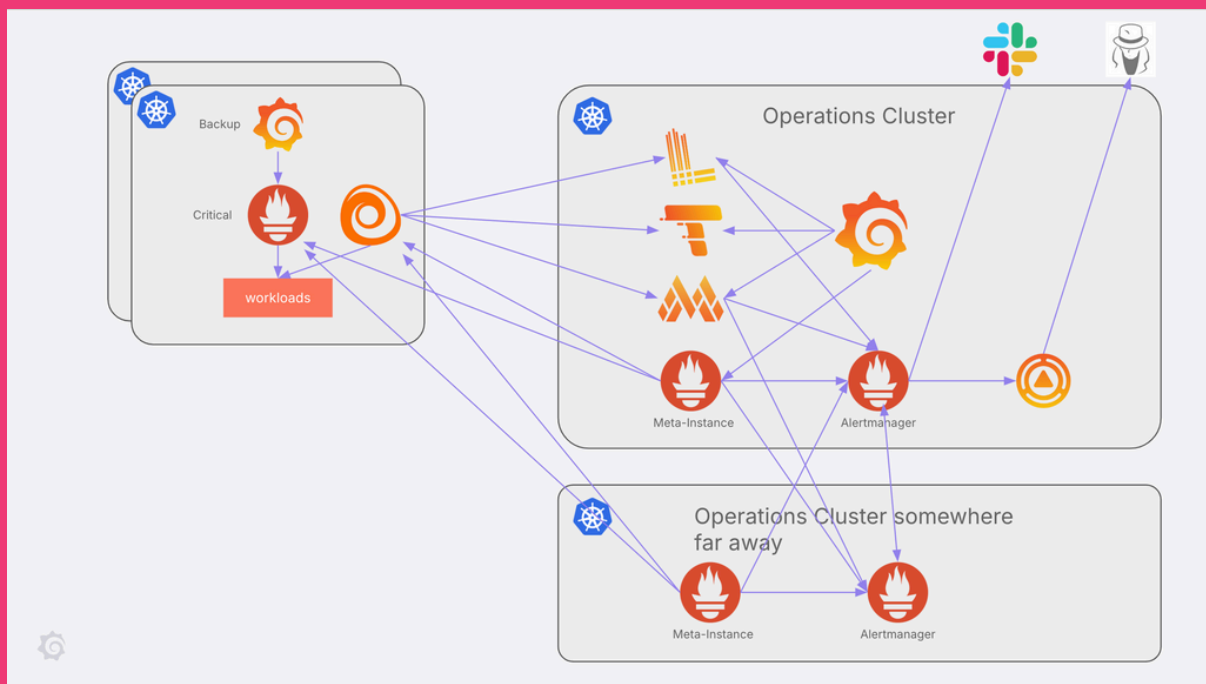
- Beyla (eBPF)
- Alloy
- OTel
- manual

When you start a Kubernetes cluster, it's automatically instrumented with eBPF



We use Grafana to measure our stack

- This is an actual screenshot of our ops Mimir cluster
- Check out the 14M samples/sec that we ingest



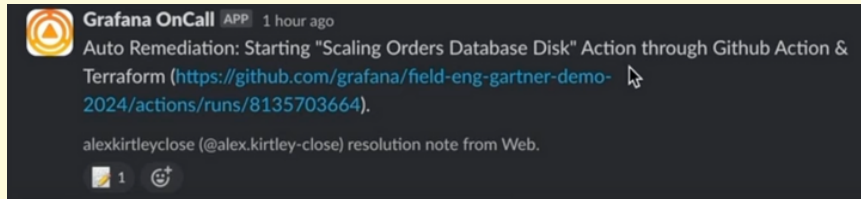
(13m)

We do quite a bit of metamonitoring. We've strayed a bit into paranoid territory, but is it paranoid if bad stuff actually happens?

- On the left is our production cluster and backup. These clusters are instrumented through Alloy, which routes information to our operations cluster. Ops cluster has real data, so it's not a test environment, but it is also where we deploy things first (ex: Adaptive Logs)
- Ops cluster monitors the monitors. We have logs, traces, metrics
- For every Prometheus, we have a meta-Prometheus. it's always an HA pair (one in the Americas, one in Europe). They scrape each other and Alertmanager
- We have a Prometheus that is monitoring Mimir which is monitoring Mimir
- Global Alertmanager cluster with instances in the EU and US that gossip among themselves
- We use OnCall to handle incidents
- Dead Man's Snitch in case OnCall fails (this has happened) - heartbeat
- BTW this exists across three cloud providers: AWS, GCP, and Azure
- 13% of our costs are just on our own observability

# Recovery

How do we recover from failures?



(19m)

OnCall

- managing on-call rotations
- auto remediation for common issues: in this case, a workflow was started via webhook to GitHub Action that upscaled a disk through Terraform
- Alerts: Slack and mobile

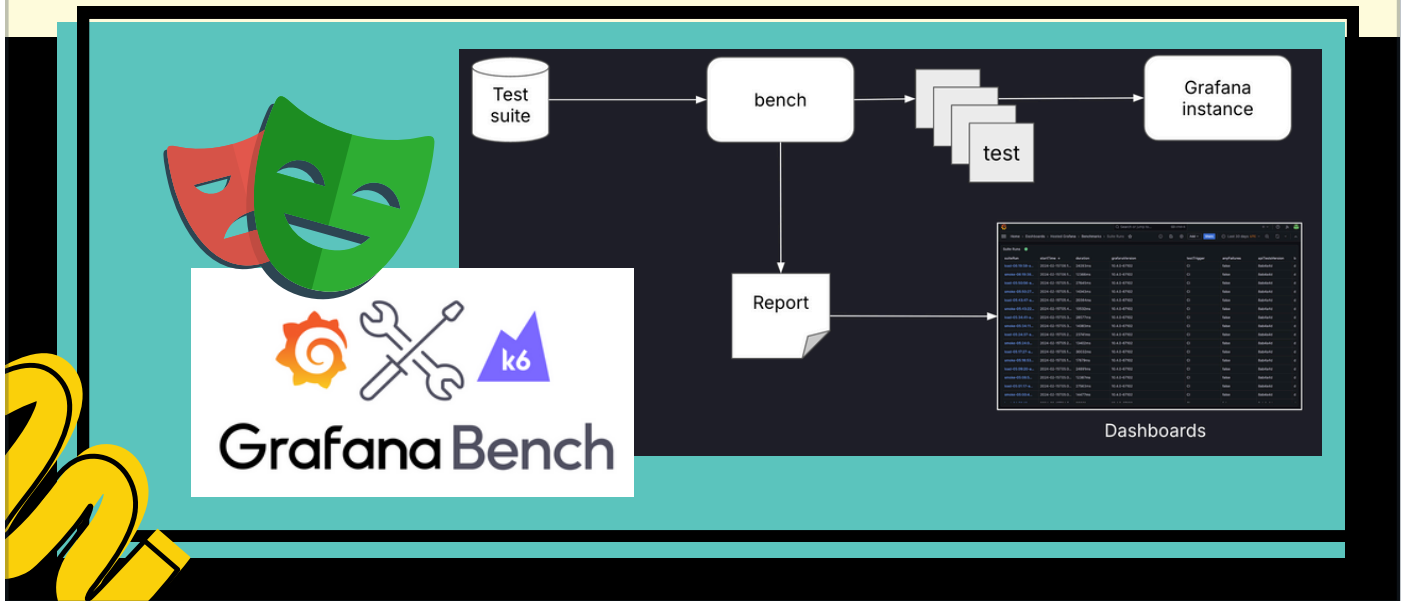
Every team is responsible for what they build. There's no wall that they chuck their code over to some ops team that have sole responsibility for everything

# Growth



- k6 to test API endpoints, synthetic monitoring
- Faro for frontend observability of: Grafana Cloud, OnCall, Grafana Cloud k6, synthetic monitoring, Kubernetes monitoring app + more
- Keda to autoscale our stuff. Ex: A Prometheus instance monitors our ops Mimir and the metrics in that Prometheus are used to determine whether to scale Mimir

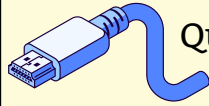
# Testing Grafana



## Grafana Bench

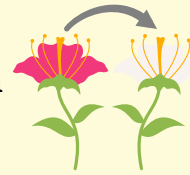
- wrapper around k6
- Runs in local development, CI/CD, and rolling release channels
- Consistent log output between k6 and Playwright
- Docker image with all dependencies

# Engagement



Quarterly hackathons

Cross-pollination



Relentless  
dogfooding

Default to  
transparency



- Hackathons: 60% moved forward, 30% shipped. Ex: graphical query builder, k6 Studio, GrotBot, Explore apps, FlameGrot AI
- dogfooding is highly encouraged
- cross-pollination among teams is encouraged
- Default to transparency... even when it's "too much" for some people's comfort
  - Learning in public: RAD videos, k6 "week of load testing"

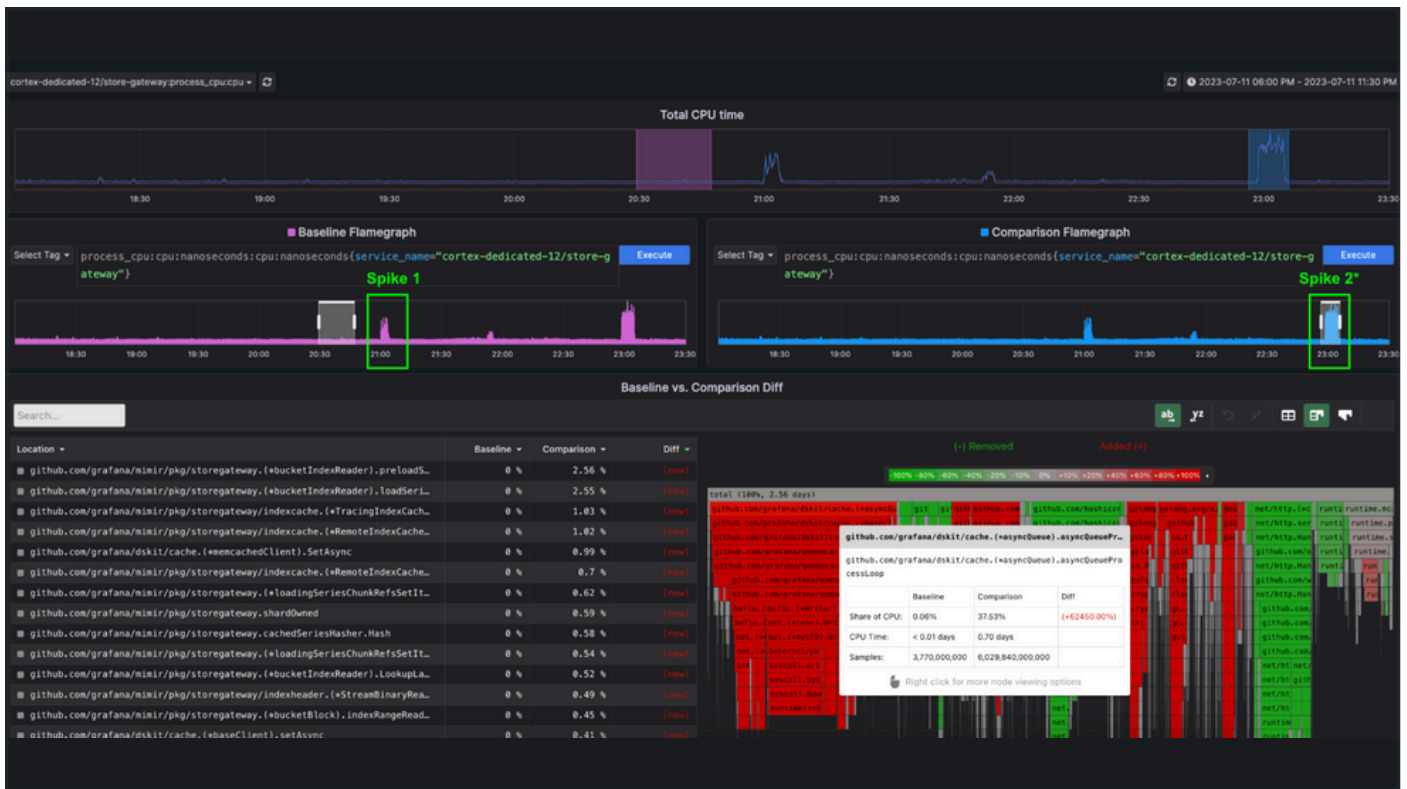


# Cortex



Back to this production incident. We finally solved this mystery: "There's an OSS project for that". Pyroscope.





- Pyroscope is a continuous profiler, but it doesn't save every single measurement. Instead it saves the shape of the utilisation as a whole, to make these flamegraphs.
- You can diff the flamegraphs (red) to see what changed.
- In this case, we were able to figure out exactly what process was causing the issue.
- Because of eBPF, everything in Grafana is profiled using Pyroscope now
- It wasn't just observability, it was continuous reliability that got us there

# The Continuous Reliability FORGE



F is for  
Framework



O is for  
Observability



R is for  
Response



G is for Growth



E is for  
Engagement

Specifically, the continuous reliability forge requires these five factors:

(RIGHT) It means having the right *\*Framework\** in place: an underlying architecture that is resilient from the get-go and doesn't need to be rearchitected later when the system grows.

(RIGHT) It means always having continuous observability through the use of auto- and manual instrumentation that let us peek under the hood at any time. It involves meta-monitoring and paranoid observability.


(RIGHT) It also means having *\*Recovery\** mechanisms set up so that when things do go wrong, there is a process for the system to be fixed, reforged if needed, and sent off into battle again.

(RIGHT) It means leaving room for *\*Growth\**: making sure that there's enough slack between components for new, better measures to be implemented. It means a system of continuous testing to make sure regressions don't get introduced.


(RIGHT) And continuous reliability also depends on the *\*Engagement\** of SREs, of software developers, of stakeholders, and of people like us who are interested in making sure our systems keep improving.

It's not about observability. It's not about watching. It's about what you do afterwards. We still have production incidents. Swords still break in battle. But those that do, get melted down or reforged into something more reliable. We watch the watchers, but we use what we see to inch

towards continuous reliability.



nicole.to/kubeconslc  
grafana.com  
@grafana, @nicolevdh  
@nicole@pkm.social  
nicole@grafana.com



# References

(blog) [How we use metamonitoring Prometheus servers to monitor all other Prometheus servers at Grafana Labs](#)

(blog) [How we scaled our new Prometheus TSDB Grafana Mimir to 1 billion active series](#)

(blog) [How we responded to a 2-hour outage in our Grafana Cloud Hosted Prometheus service](#)

(blog) [How adding Kubernetes label selectors caused an outage in Grafana Cloud Logs — and how we resolved it](#)

(blog) [How a Production Outage Was Caused Using Kubernetes Pod Priorities](#)

(slides) [Pyroscope demo](#)

(video) [How to do continuous profiling right, Grafana Office Hours](#)

(people) Erik Sommer, Bryan Boreham, Ryan Perry, Pablo Chacin, Kostas Pelelis, Dee Kitchen, Jenny Lam, Bret Barker, Kristian Deppe

(image credit) [Green Beetle, Forging A Sword Pt 1: Every Stroke](#)