How a Production Outage Was Caused Using Kubernetes Pod Priorities

Published: 24 Jul 2019

by Tom Wilkie

On Friday, July 19, Grafana Cloud experienced a ~30min outage in our Hosted Prometheus service. To our customers who were affected by the incident, I apologize. It's our job to provide you with the monitoring tools you need, and when they are not available we make your life harder. We take this outage very seriously. This blog post explains what happened, how we responded to it, and what we're doing to ensure it doesn't happen again.

Background

The Grafana Cloud Hosted Prometheus service is based on Cortex, a CNCF project to build a horizontally scalable, highly available, multi-tenant Prometheus service. The Cortex architecture consists of a series of individual microservices, each handling a different role: replication, storage, querying, etc. Cortex is under very active development, continually adding features and increasing performance. We regularly deploy new releases of Cortex to our clusters so that customers see these benefits; Cortex is designed to do so without downtime.

To achieve zero-downtime upgrades, Cortex's Ingester service requires an extra Ingester replica during the upgrade process. This allows the old Ingesters to send in-progress data to the new Ingesters one by one. But Ingesters are big: They request 4 cores and 15GB of RAM per Pod, 25% of the CPU and memory of a single machine in our Kubernetes clusters. In aggregate, we typically have much more than 4 cores and 15GB RAM of unused resources available on a cluster to run these extra Ingesters for upgrades.

However, it is often the case that we don't have 25% of any single machine "empty" during normal operation. And we don't want to – this is CPU and memory we could use for other processes. To solve this, we chose to use Kubernetes Pod Priorities. The idea is to designate Ingesters as a higher priority than other (stateless) microservices. When we need to run N+1 Ingesters, we temporarily preempt other, smaller pods. These smaller pods will reschedule in the spare resources on other machines, leaving a big enough "hole" to run the extra Ingester.

On Thursday, July 18, we deployed 4 new priority classes to our clusters: **critical**, **high**, **medium**, and **low**. We had been running these priorities on an internal cluster with no customer traffic for ~1 week. The **medium** priority class was set to be the default for pods that didn't specify an explicit priority, and Ingesters were set to the **high** priority class. **Critical** was reserved for monitoring jobs (Prometheus, Alertmanager, node-exporter, kube-state-metrics, etc.). As our config is open source, the PR for this is here.

The Incident

On Friday, July 19, one of our engineers spun up a new dedicated Cortex cluster for a large customer. The config for the new Cortex cluster did not include the new Pod Priorities, so all the new Pods were given the default priority, medium.

There were not enough resources on the Kubernetes cluster to fit the new Cortex cluster, and the existing production Cortex cluster had not been updated to include the **high** priority designation for their Ingesters. As the new cluster's Ingesters had **medium** priority (the default) and the existing production Pods had no priority, the new cluster's Ingesters preempted an Ingester from the existing production Cortex cluster.

The ReplicaSet for the preempted Ingester in our production Cortex cluster noticed the preempted Pod and created a new Ingester Pod to maintain the correct number of replicas. This new Pod was given the default (**medium**) priority, and as such preempted another production Ingester. This triggered a **cascading failure** that eventually caused the preemption of all the Ingester Pods for the production Cortex clusters.

Cortex Ingesters are stateful, holding up to the last 12 hours of data. This allows us to compress the data more efficiently before writing it to long term storage. To enable this, Cortex shards the data on a series-by-series basis using a Distributed Hash Table (DHT), and replicates each series to three ingesters using Dynamo-style quorum consistency. Cortex does not write data to ingesters that are shutting down. As such, when there are a large number of ingesters leaving the DHT, Cortex is unable to achieve sufficient replication of writes, and they will fail.

Detection and Resolution

Our new error-budget-based Prometheus alerts (details to come in a future blog post) paged the oncall within 4 minutes of the start of the outage. After a further ~5 minutes, we had diagnosed and scaled up the underlying Kubernetes cluster to accommodate both the new cluster and the existing production clusters.

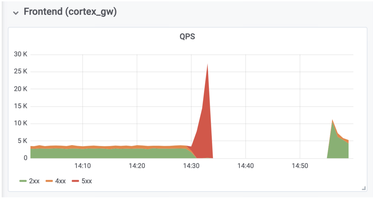
After another ~5 minutes, the old ingesters had successfully written out their data and the new ingesters had successfully scheduled; the Cortex clusters became available again.

It took a further ~10 minutes to diagnose and fix the out-of-memory (OOM) errors from our authenticating reverse proxies that sit in front of Cortex. This OOMing was caused by a >10x increase in QPS, we believe from overly aggressive retries from the client's Prometheus servers.

Impact

The total length of the outage was 26 minutes. No data was lost; the ingesters succeeded in flushing all their in-memory data to long-term storage. During the outage, customers' Prometheus servers buffered remote writes using the new WAL-based remote write code (written by Callum Styan at Grafana Labs) and replayed the failed writes after the outage.

Prod cluster's writes:



Takeaways

It is important that we learn from this outage and put in place steps to ensure it does not happen again.

In hindsight we should not have made **medium** the default priority until we had added the **high** priority designation to the production Ingesters. We should also have rolled out the **high** priority designation to production Ingesters sooner. This has now been fixed. We hope that by publishing this blog post, others considering deploying Kubernetes Pod Priorities can learn from our mistakes.

We will be adding an extra layer of scrutiny to the deployment of any "extra" namespace objects, i.e. config that is global to the cluster. Going forward, changes like this will be reviewed by more people. In addition, this change was considered too minor for a design document – it was only discussed in a GitHub issue. All extra-namespace config changes will require a design document going forward.

Finally, we will be automating the sizing of our frontend authenticating reverse proxy to prevent the OOMing behavior on overload that we saw, and investigate the default backoff and scaling behavior of Prometheus to prevent the stampeding herd.

This outage is not all bad news: Once given enough capacity, Cortex recovered automatically with no additional intervention. We also gained valuable experience using Grafana Loki, our new log aggregation system, to confirm the Ingesters exited gracefully.

Recent Posts

Grafana and NGINX are partnering to give the open source community a turnkey experience for visibility

New Enterprise features in Grafana 7.0: Usage insights and user presence indicator

Getting started with the Grafana Cloud Agent, a remote_write-focused Prometheus agent

Why optimizing for MTTR over MTBF is better for business

Learning by doing: Making virtual offsites work

Related Posts

Meet the Grafana Labs team: Prometheus maintainer Ganesh Vernekar

How to visualize Prometheus histograms in Grafana

How we made working with Prometheus easier with metric metadata in Grafana's Explore view

New in Prometheus v2.19.0: Memory-mapping of full chunks of the head block reduces memory usage by as much as 40%

Cortex v1.1 released with improved reliability and performance

Categories

Cloud Community Company Cortex Dashboard Design Devops Elasticsearch Engineering Enterprise Events Explore Grafana Grafana-Buzz Grafanacon Graphite Guest How-We-Work Influx Jaeger Kubernetes Logging Loki Metrictank Monitoring Mysql Open-Source Prometheus Release Security Snap-Telemetry Talks Tanka Team Time-Series-Databases Timeshift Tracing Tutorials Ux Video Worldping Xxxxxx

Related Posts

Ask Us Anything: The Most Popular Grafana Community Questions Answered!

Check out three of the most popular questions—and answers!—on the Grafana Labs community board.

Read more →

Grafana Labs at KubeCon: All the Highlights

In case you missed it, all the highlights from KubeCon + CouldNativeCon EU from the Grafana Labs team.

Read more →

Grafana Labs at KubeCon: The Latest on Cortex

At KubeCon this week, Tom Wilkie is giving two talks on Cortex. ICYMI, here's the latest news about the technology used in Grafana Cloud's Hosted Prometheus.

Read more →

Related Case Studies

Hiya migrated to Grafana Cloud to cut costs and gain control over its metrics

To scale Prometheus, says Senior Software Engineer Jake Utley, Grafana Cloud was 'the most in line with what we wanted to accomplish.'

Learn more →

How Cortex helped REWE digital ensure stability while scaling grocery delivery services during the COVID-19 pandemic

Cortex's horizontal scaling has been crucial; reads and writes increased significantly, and the platform was able to handle the added load.

Learn more →

How Gojek is leveraging Cortex to keep up with its ever-growing scale

Gojek's Lens monitoring system has 40+ tenants, for which Cortex handles about 1.2 million samples per second.

Learn more →