Scaling OpenTelemetry Collectors using Kafka

Pranay Prateek (Co-Founder, SigNoz)

About Me

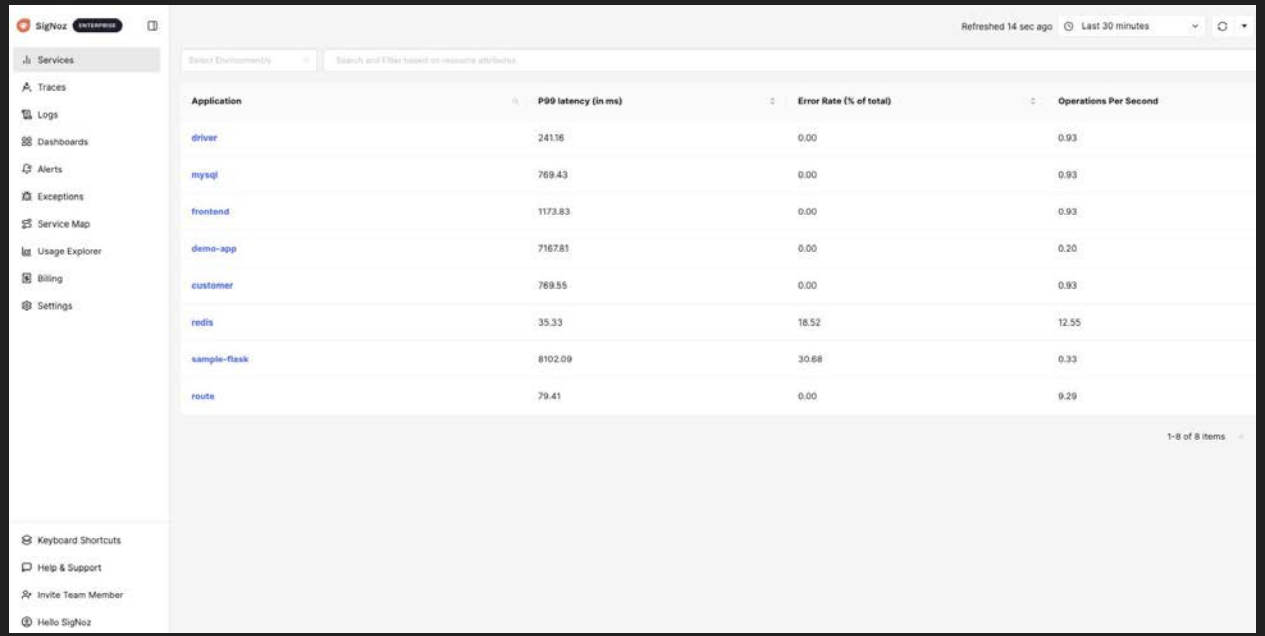
Co-Founder @ SigNoz

ex-Product @ Microsoft

Love reading & trekking

SigNoz - Open Source Observability Platform

OpenTelemetry-Native Traces, Metrics & Logs in a single pane



🌟🌟 17K+ GitHub stars

👫👫 4000+ members in slack community

💻💻 130+ contributors

What is OpenTelemetry?

Why is OpenTelemetry important?

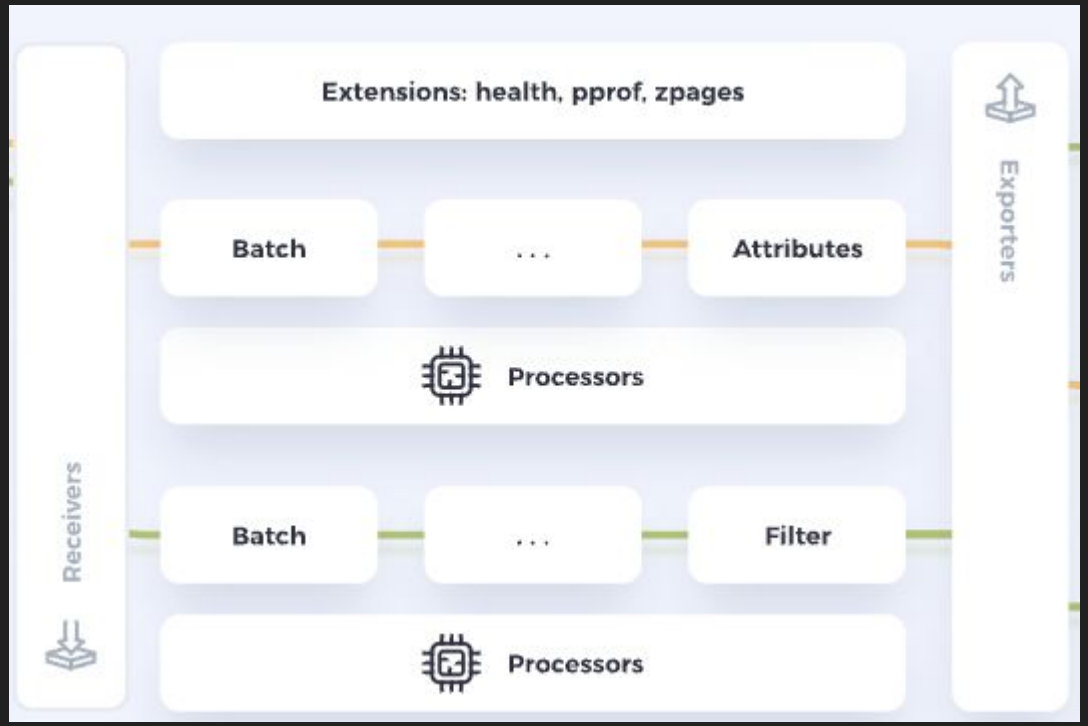
- 2nd fastest growing project in CNCF (only after Kubernetes)
- No vendor lock-in, open source instrumentation
- Becoming the default standard for instrumentation and introducing open standards for instrumentation
- Provides instrumentation sdks for metrics, traces and logs. New signals like profiling in progress.
- SigNoz is based natively on OpenTelemetry for instrumentation

More details - https://opentelemetry.io/

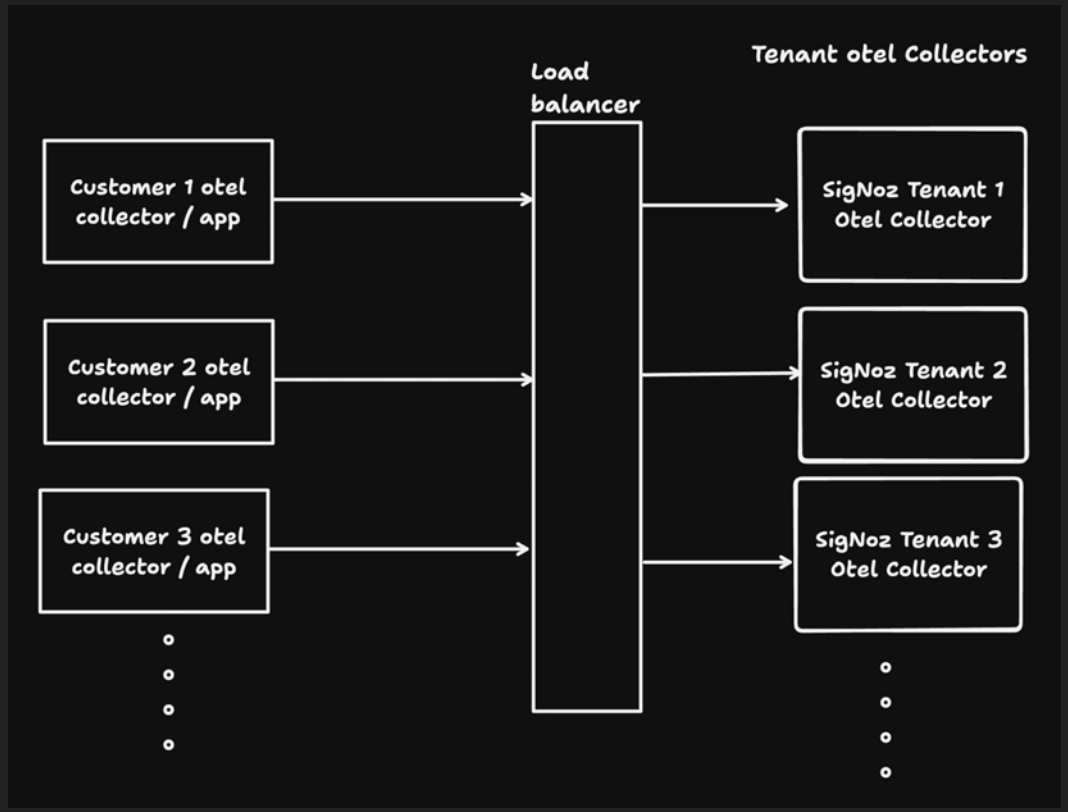
Introduction to OpenTelemetry Collector

Three key components

1. Receivers
2. Processors
3. Exporters



Architecture of SigNoz Cloud (single tenant) without Kafka

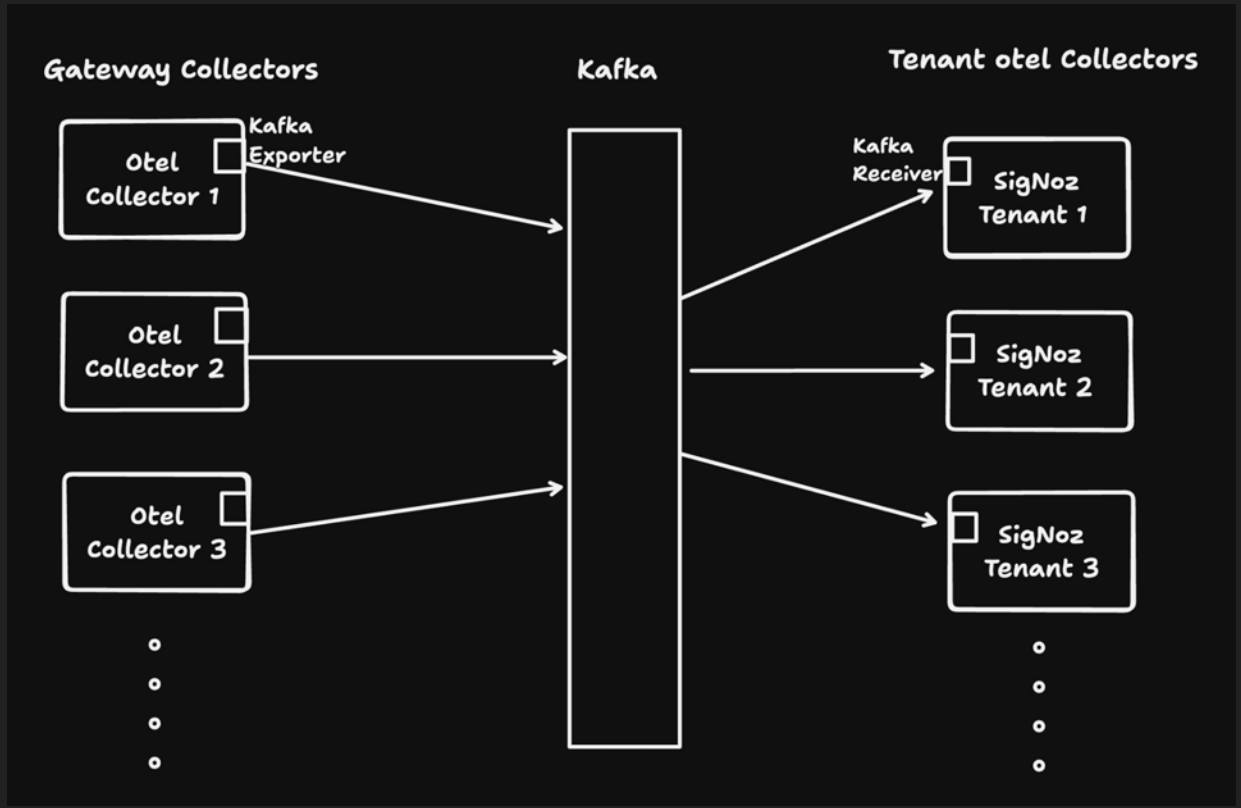


Issues with scaling with just Opentelemetry Collector

- Tenant or DB downtime caused the agents report 5xx and possibly loss of data after a few minutes

- Tenant has to scale according to the ingestion rate. If ingestion rate spikes to 10x and drops to limits in a few minutes, it caused slowdown at tenant and possible rejection of data until the tenant DB scales up

Architecture of SigNoz Cloud with Kafka



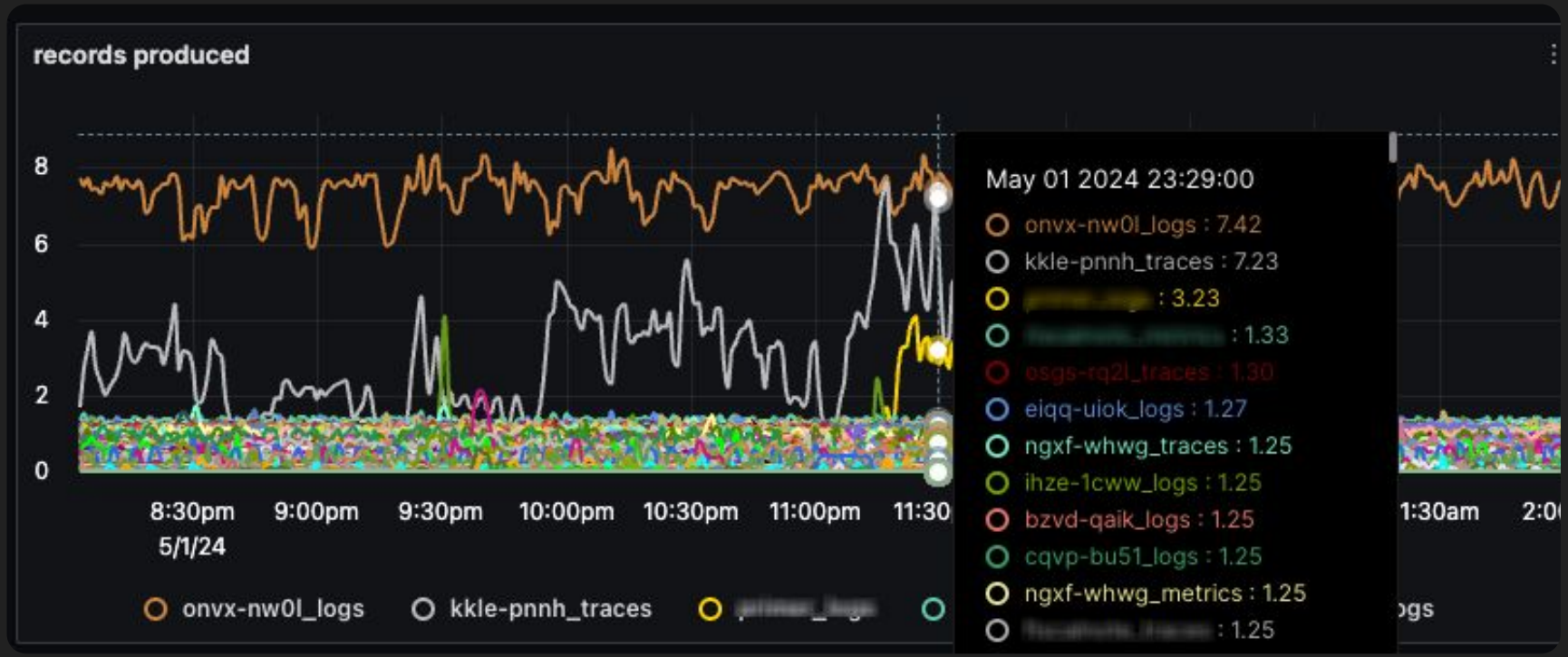
How Kafka can help

- Highly available ingestion. Kafka acts as a buffer for 6hrs retention period configured
- Kafka can handle bursty high ingestion and tenant can continue consuming at fixed speed and has time to scale up if needed
- Additional processing can be done at Kafka.
  e.g We can use traceID as partition key to send all spans of a trace to a partition
- This is typically a challenge otherwise as all spans need to arrive at the same otel-collector for the tail-sampling decision

Current Kafka Setup

- 6 hrs retention period
- Replication factor 3
- 10 MB max message size

Records Produced



Records Consumed

Monitoring Consumer lag is important

Scaling based on Consumer lag

- We get alerted if consumer lag of any partition increases to a threshold
- This metric can be used to scale up your consumer group (defined at the tenant otel-collector) and tenant.

  e.g add more partitions for the topic so that more tenant otel-collectors can be deployed as number of partitions is the limit of parallelism in kafka

Monitoring Producer - Consumer Latency

Kafka based architecture is working well so far…

- Very fast at ingestion. Data is retained for 6 hrs only
- We even get a compression factor of 10 to 15 as the data is batched before ingesting to Fafka. The otel-collector agent at a user's infra also sends a batched data to the gateway otel-collectors
- Able to handle spikes in customer ingestion

Potential Improvements

- Automatic increase of partitions based on scale of ingestion at a topic
- A partition can get stuck if a tenant otel-collector throws a permanent failure. Solution is to drop after a few retries. Or send the message to a DLQ (Dead Letter Queue) and move to the next message
- Making the tenant otel-collector (kafka receiver -> processors -> exporter) a synchronous module so that consumer commits an offset only after the message is successfully written to the DB.
- Make the complete write path an exactly-once delivery model

Get involved in a growing community

SigNoz repo - https://github.com/SigNoz/signoz

Slack Community - signoz.io/slack

Create an issue - https://github.com/SigNoz/signoz/issues

Thank You

Let’s chat more in our slack community
signoz.io/slack

pranay@signoz.io

@pranay01