SREcon24 EMEA

Evolution of observability @Riot Games

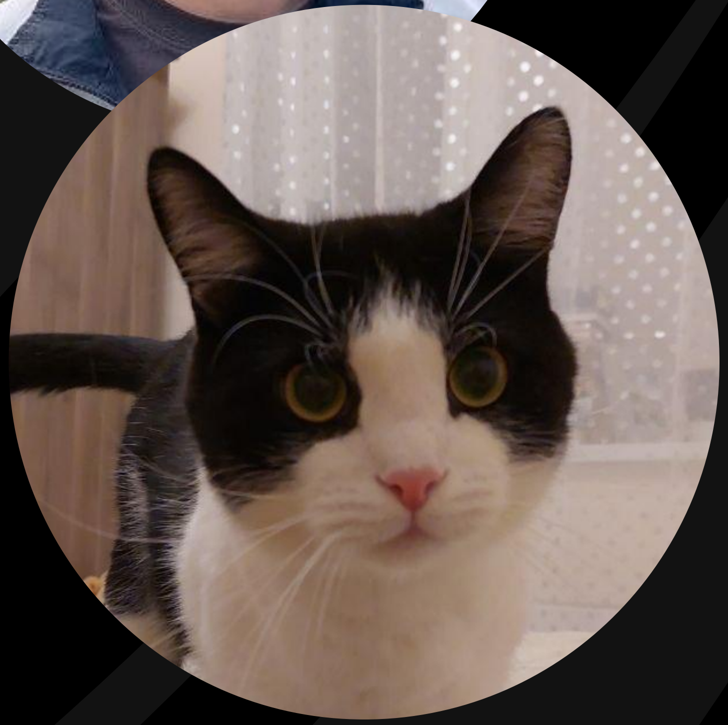
RIOT GAMES

About us

Erick Moreira

Senior Software
Engineer
LiveOps Organisation

Lorenzo, il Magnifico,
& *Luigi*
Junior food eaters



Kirill Mikhailov

Senior Software
Engineer
LiveOps Organisation

Whiskey

01

What is Riot Games

SREcon24 EMEA

Riot Games

Founded
Riot Games was founded in 2006

2006

Riot Games

Founded
Riot Games was founded in 2006

2006

2009

League of Legends released
One of the most-played PC games in the world

SREcon24 EMEA

Riot Games

Founded
Riot Games was founded in 2006

2006

2009

League of Legends released
One of the most-played PC games in the world

League of Legends becomes a global leader in esports

~

Riot Games

Founded
Riot Games was founded in 2006

2006

2009

League of Legends released
One of the most-played PC games in the world

League of Legends becomes a global leader in esports

~

2019

Teamfight Tactics released
Put the S in Riot Games

Riot Games

Founded
Riot Games was founded in 2006

2006

2009

League of Legends released
One of the most-played PC games in the world

League of Legends becomes a global leader in esports

~

2019

Teamfight Tactics released
Put the S in Riot Games

Release of Valorant, Wildrift and Legends of Runeterra

2020

02

What makes observability different at Riot

Diverse landscape of technologies

Game engines and platforms

Riot operates multiple game engines, from the alikes of giants of the industry such as Unreal and Unity to our own in-house grown. We also deploy to different platforms such as Windows, Mac, iOS, Android, PS5, XBOX, and browsers.

Highly distributed and fragmented applications

We operate more than 1000 traditional backend services, written in Python, Java, Golang, JavaScript, and C#, running on Kubernetes using databases, queues, load balancers, and so on.

Esports events and streaming

Riot is responsible for delivering many of the biggest eSports tournaments out there, and as such, they need different SLAs; they have different monitoring points and separate infrastructure.

Outposts and networking infrastructure

While Riot is actively working on decommissioning most of its own physical infrastructure around the world, we still collaborate with our cloud partners to provide the best network and routing infrastructure for our players.

Latency and packet loss measurements are essential

Latency is crucial

100ms spikes during an online game of Valorant feel much worse than magnitudes higher latency on the regular web page. We take active measurements both on the client side and game servers to detect and react to any possible issue. ISP issues, ocean cable cuts, and other issues have a significant impact on our operations.

Data volume limitations

Due to the high volume of individual data points from the game server and game clients, we have to work on pre-aggregation, throttling, and filtering. The pipeline should be robust and have a high variety of instruments to work with the data.

Example of peeking in game with a different ping

Image shows how far a player who starts fully behind cover is able to peek around a corner before their opponent sees the movement.

Source: https://technology.riotgames.com/news/peeking-valorants-netcode

We run our games close to our players

Due to the number of regions we operate, it presents a lot of challenges. Players login mostly at night and weekends in their regions. Rollouts can take days due to A/B testing, canary releases, and more. Since we have to interact with game publishers, it also presents unique challenges on how we expose service telemetry to be consumed.

>100 clusters

Only counting Kubernetes clusters

>30 regions

Including both Cloud regions and self-managed outposts

Publishers

We distribute our games to some regions through publishers

03

History of observability in Riot Games

History of observability

Early Days

Characterised by team-specific observability stacks

ZABBIX graphite Nagios® Grafana

And others

Teams were building and supporting observability pipelines by themselves.

History of observability

First attempts to build a centralised solution

Zabbix was selected as the main monitoring solution.

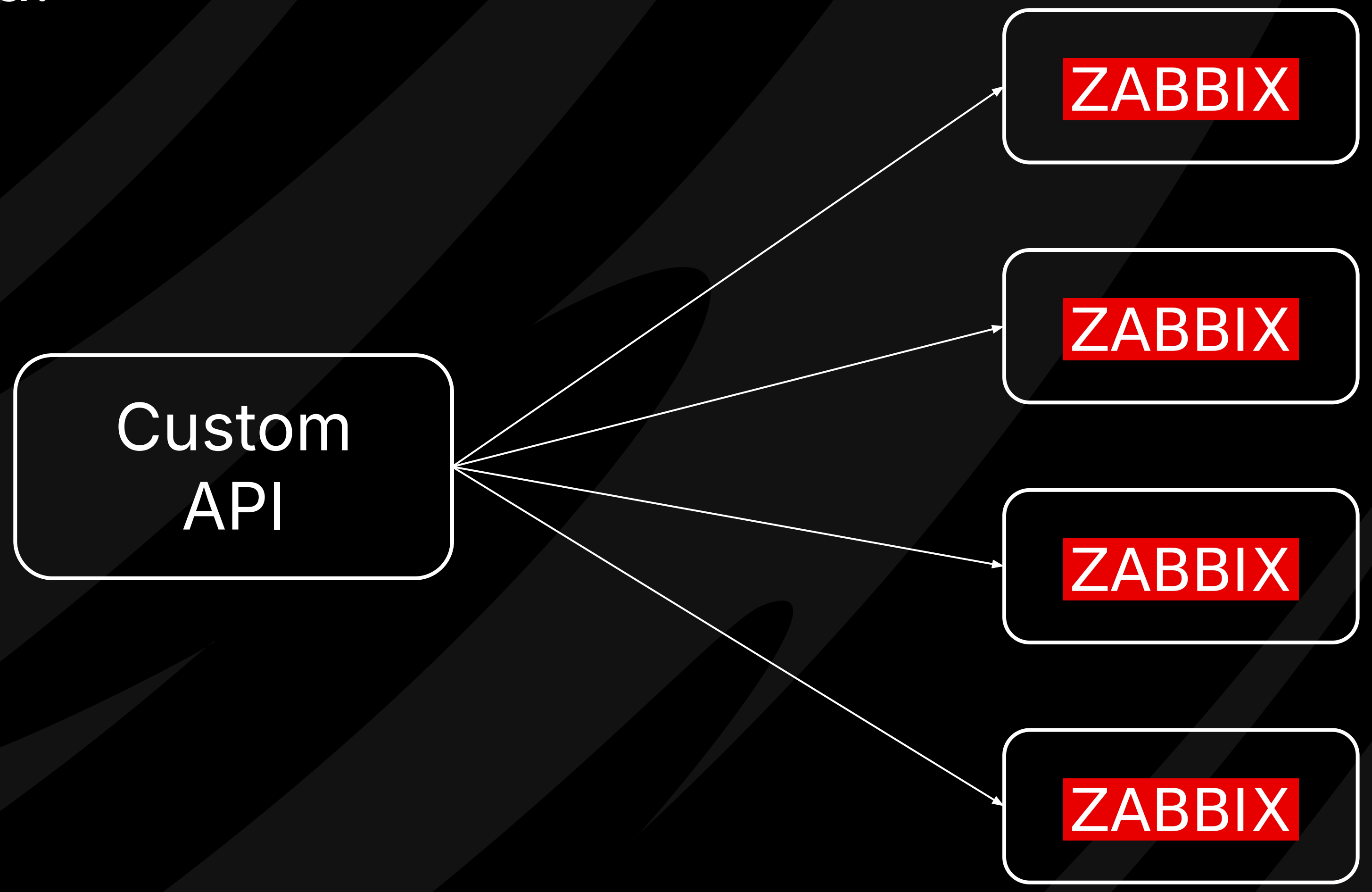
Multiple Zabbix instances were scattered throughout the infrastructure.

SREcon24 EMEA

History of observability

First attempts to build a centralised solution

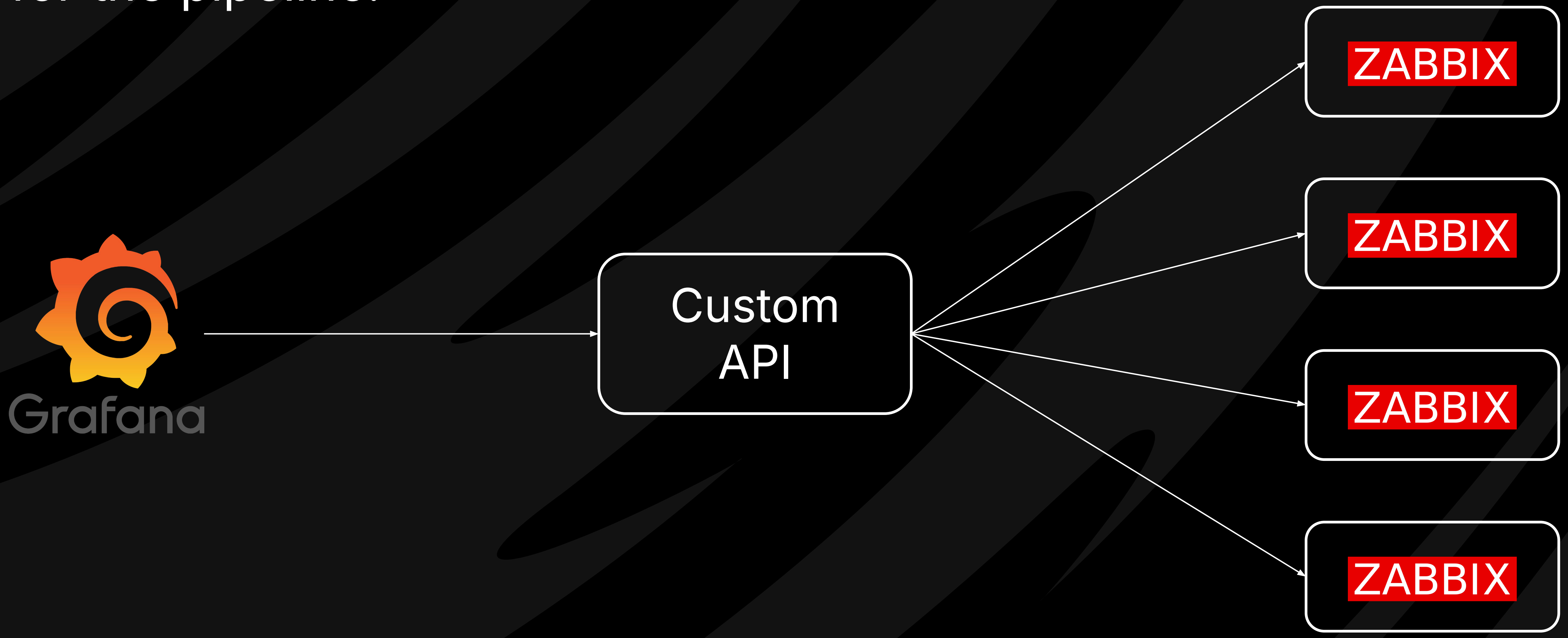
Along the way, a custom API to query multiple Zabbix instances was developed.



History of observability

First attempts to build a centralised solution

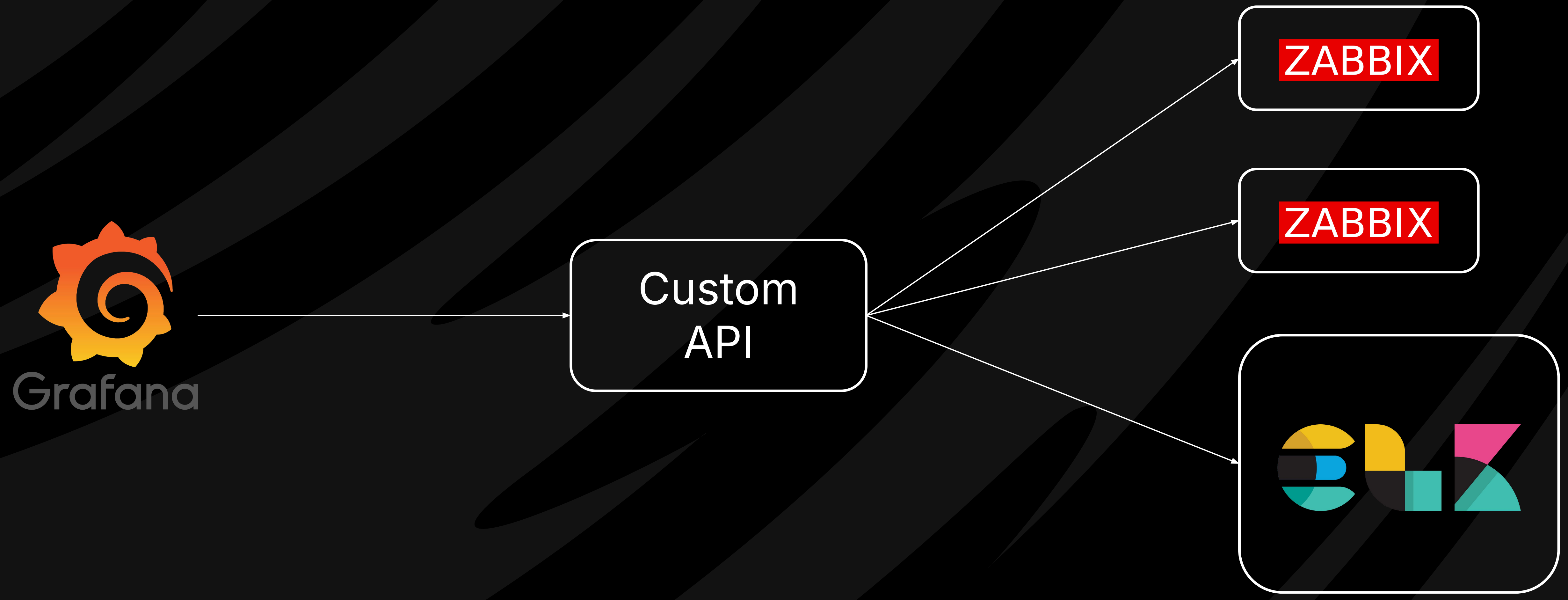
And Grafana was chosen as a main visualisation tool for the pipeline.



History of observability

In-house centralized ELK solution

ELK stack was introduced as a centralised solution.

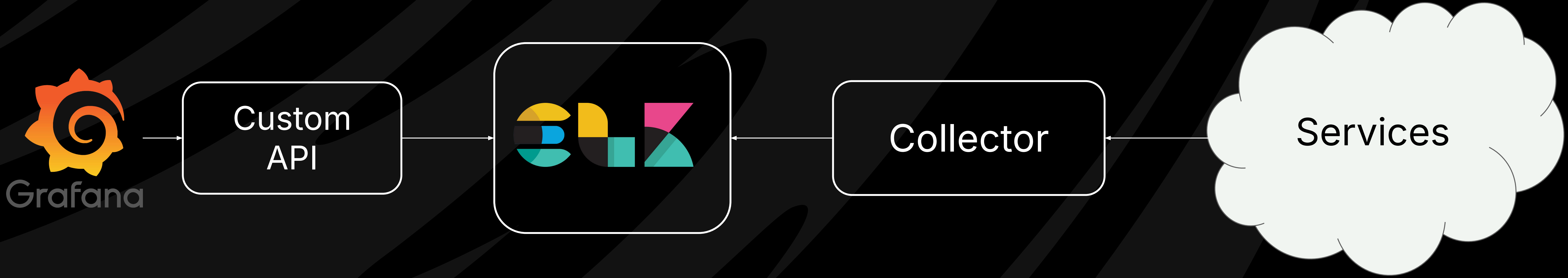


Custom API supported both Zabbix and ELK stack.

History of observability

In-house centralized ELK solution

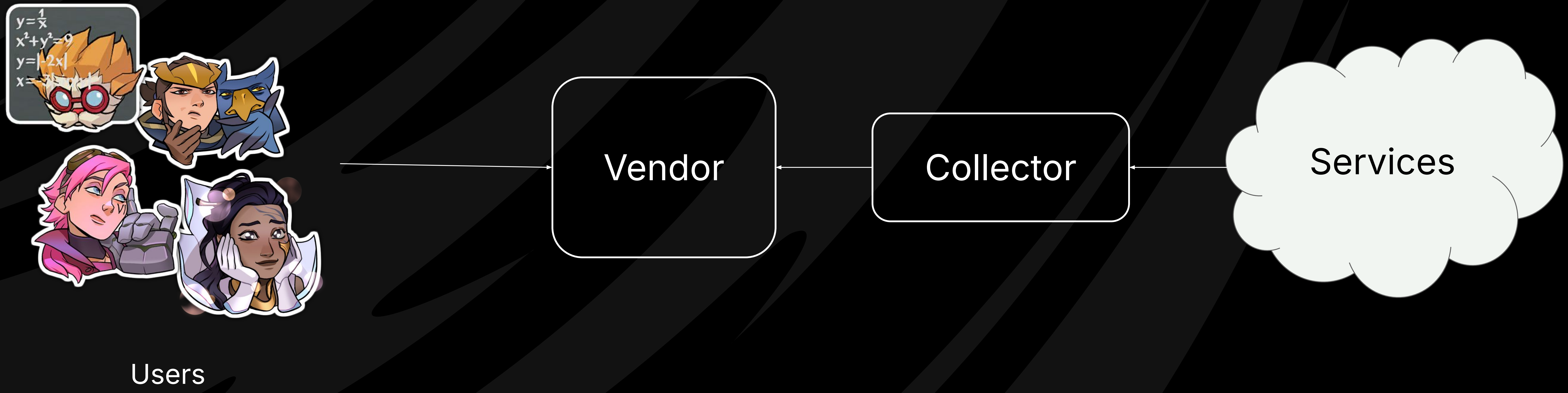
Zabbix was omitted
Services were pushing data through the collector to ELK stack



History of observability

Vendored solution

Collector was switched to use new observability solution from one of the SaaS Vendors



04

Problems with the last approach

SREcon24 EMEA

Hard to integrate, slow to adapt, governance problems

01

Hard to integrate

- Mainly integrated with internal frameworks
- If you were using Python, Lambdas, or anything else, you were out of luck.
- Changes took ages to propagate as they were delivered with framework changes.
- Lack of unified documentation on observability and telemetry

02

Slow to adapt

- Small team with burden to develop and load test internal service.
- Low time to keep up with the latest standards, such as eBPF and other emerging technologies.
- No clear ownership on vetting technologies and distributing/enforcing standards.

03

Governance was difficult

- Accounts were separate, and API keys were easy to access.
- There were no clear standards or guidance on usage of technology and functionality from the vendor.
- With no control over metric names, it was hard to pinpoint common offenders.

Fragmentation on observability

Standards

Each team had a different set of metric names, tags, and service catalogue setup. A lot of teams were relying on synthetics and logs instead of metrics.

Tracing

No unified approach for tracing. Difficulties with the triaging experience.

Access control

Each team was responsible for managing its own account. Querying across multiple accounts was hard.

Code access

Each team managed their own repository for their infrastructure as code and had their own CICD pipeline for monitors, dashboards, etc.

Wastage was hard to control

During the service development lifecycle, it is easy to add a few log lines here and there. To add extra metrics that are forgotten. With fragmentation on account management and not a single point for telemetry ingestion, it was easy for teams to explore but hard for us to control and turn off the tap.

150TB never queried

A single metric accounted for 150TB/month ingest and was never queried

1PB in a weekend

A single load test generated over a petabyte of logs in a single weekend

Tags and name mix

With tags being appended to metric names, it was hard to pin down sources of wastage

3.5 petabytes

Logs and metrics ingested/monthly by Riot.

05 New approach

We changed vendors

Tech approach

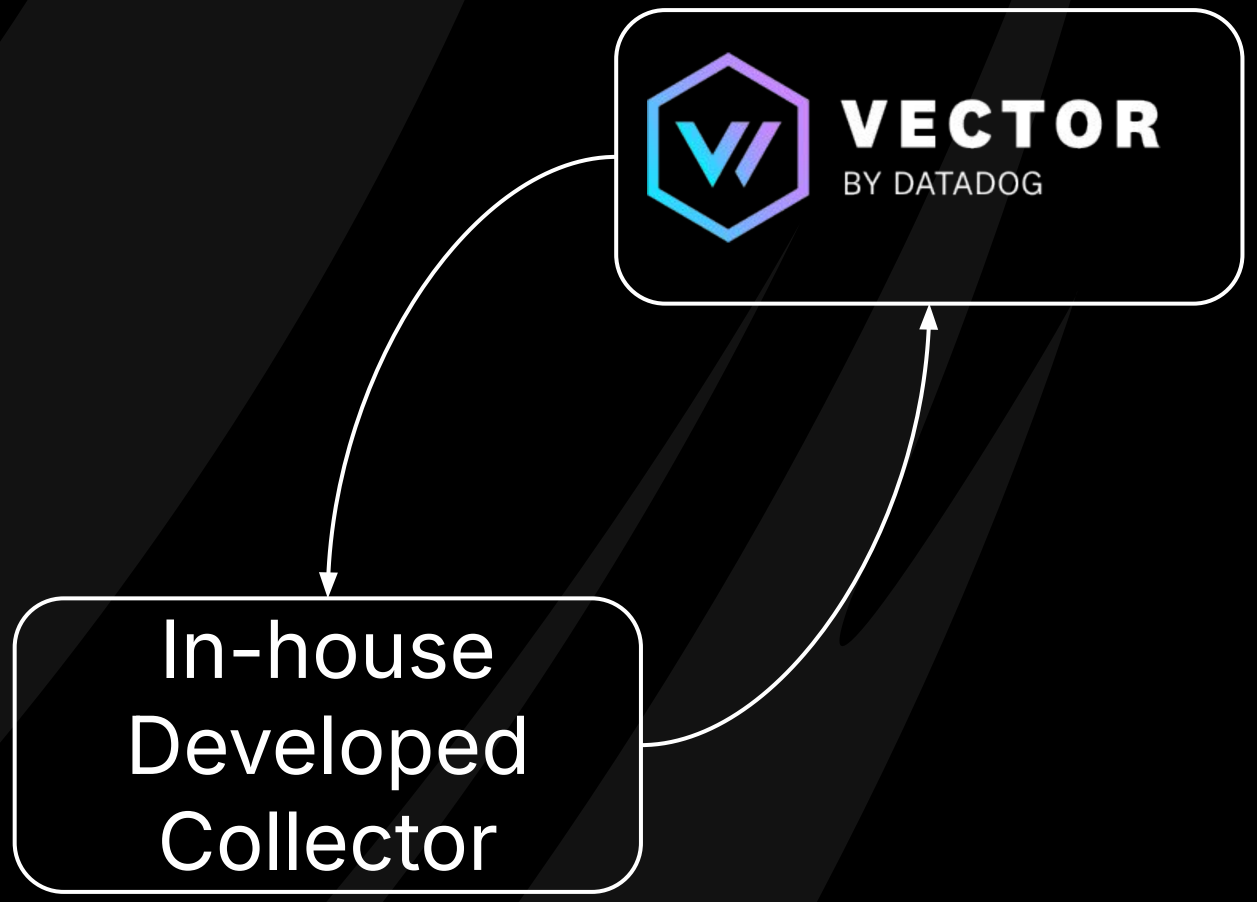
Single telemetry ingestion with Vector

- Good set of sources,sinks and transformations including OpenTel, deduping, throttling, sampling, buffering and more.

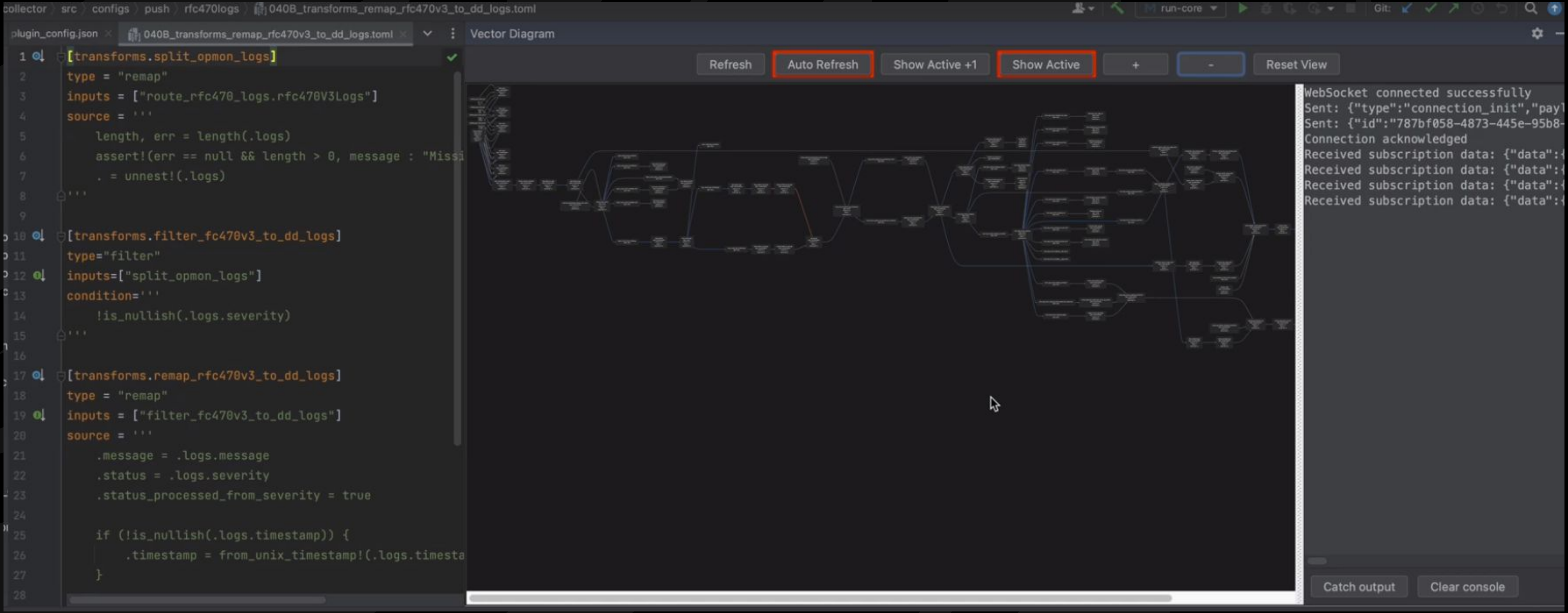
- Good performance. During the load tests we observed almost 3x performance compared to our previous collector

- Open source with mature community and documentation.

- Easy to develop and has great built-in tools such as querying api, auto-reloading, tapping, graphing, testing and more.



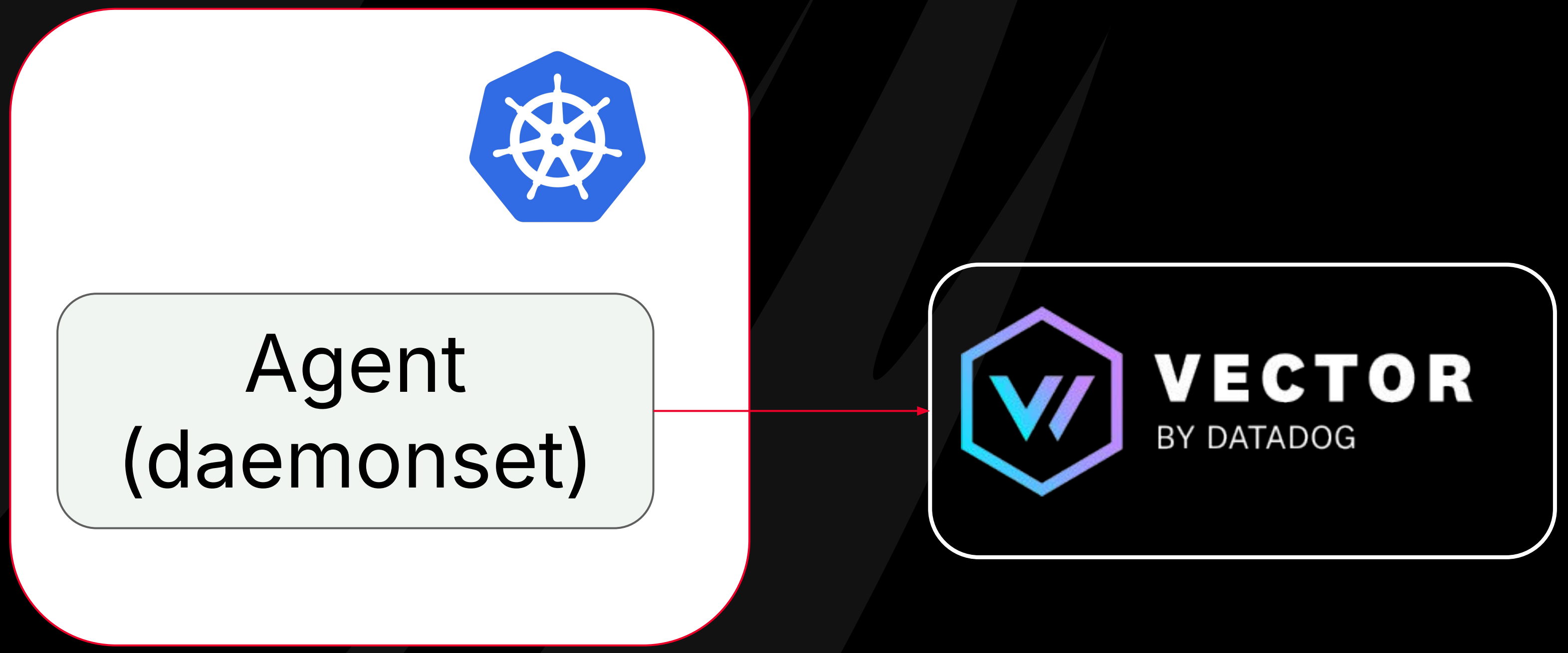
Riot internal tooling for Vector



Tech approach

Deployed agent in our environments

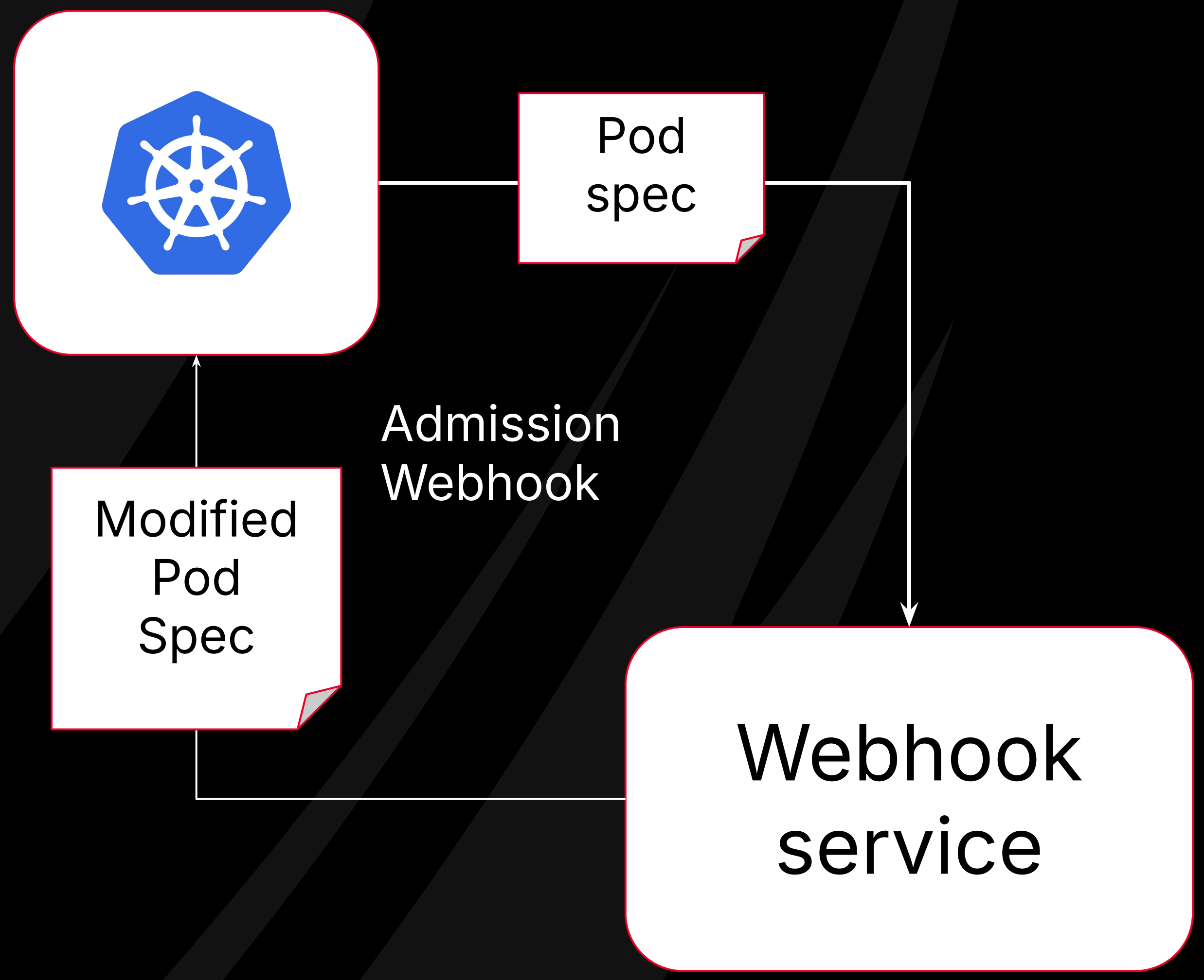
- It extracts all infrastructure metrics, such as CPU, memory and so on. Uses eBPF to extract all network metrics.
- With network metrics and information it provides us the ability to visualize all our service map and monitor all http requests with 0 changes.
- It can scrape prometheus metrics endpoints, collect traces and push it to Vector.
- It support all environments we have, such as K8s, Mesos, EC2, Fargates and lambdas.



Tech approach

Admission Webhook to stay vendor agnostic

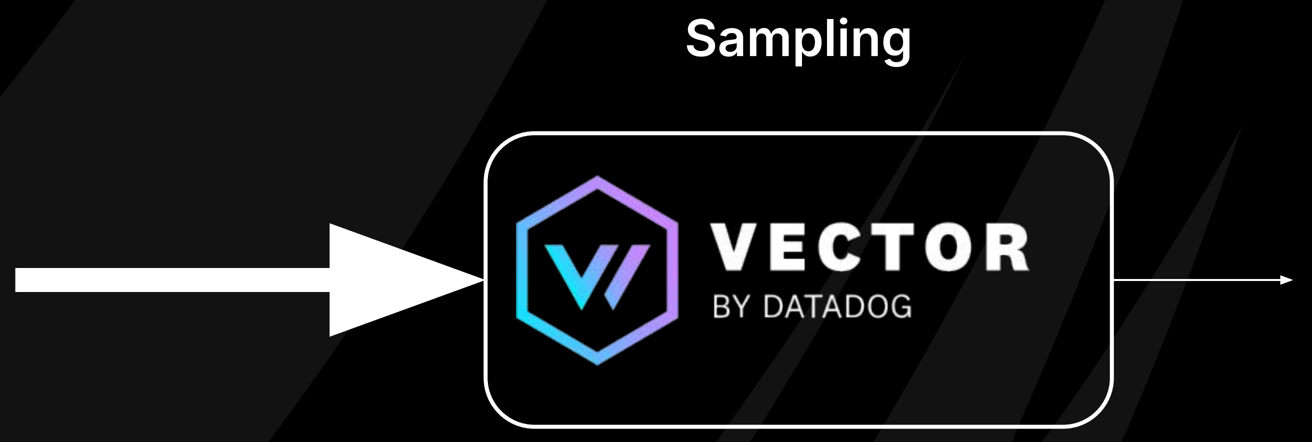
- Inject annotation, attributes and vendor specific configuration with mostly 0 effort from service owners.
- It allow us to create our own interface of annotations and configuration for K8s resources. Allow us to swap the tech implementation and version behind the scenes
- It is able to intercept and modify pods to enable injection of vendor and open source APMs. Traces everywhere!



Log and traces sampling

Enter aggressive Sampling by Default

- We introduced default aggressive sampling for logs and traces. For example, WARN logs 10%, INFO logs 1%, and so on.
- Traces are dynamically sampled according to usage with errors being prioritized.
- Teams can opt out or fine-tune the sampling ratio by service, region, environment, severity, and custom logic. This is backed by GitOps and is remotely synced to our collector instances.



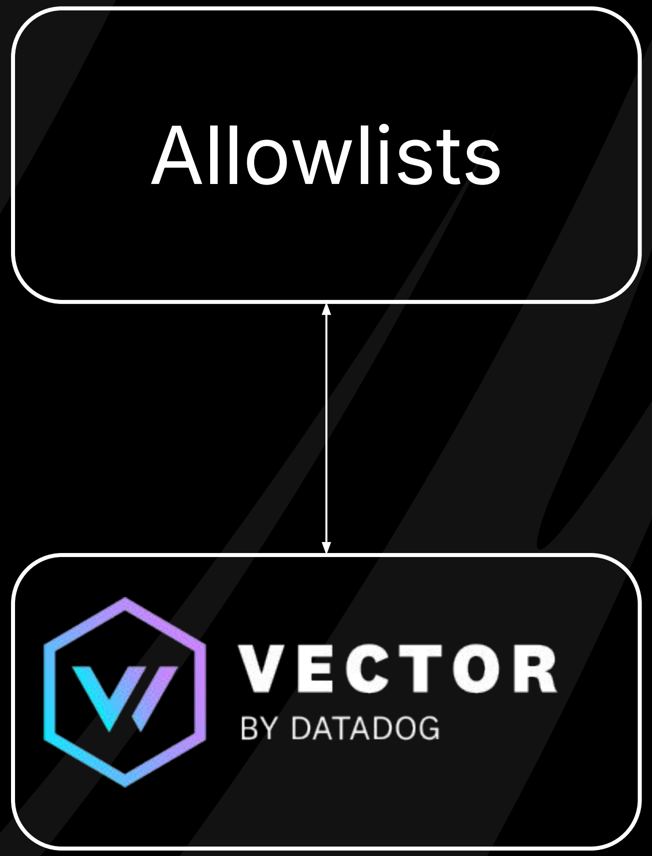
Example of log sampling configuration

316	loop.golang-pull	qa.aws-usw2-dev.canary	DEBUG	1
317	loop.golang-pull	qa.aws-usw2-dev.canary	INFO	1
318	loop.golang-pull	qa.aws-usw2-dev.canary	WARN	1
319	loop.golang-pull	qa.aws-usw2-dev.canary	ERROR	1
320	loop.golang-push	qa.aws-usw2-dev.canary	DEBUG	1
321	loop.golang-push	qa.aws-usw2-dev.canary	INFO	1
322	loop.golang-push	qa.aws-usw2-dev.canary	WARN	1
323	loop.golang-push	qa.aws-usw2-dev.canary	ERROR	1
324	loop.java-push	qa.aws-usw2-dev.canary	DEBUG	1
325	loop.java-push	qa.aws-usw2-dev.canary	INFO	1
326	loop.java-push	qa.aws-usw2-dev.canary	WARN	1
327	loop.java-push	qa.aws-usw2-dev.canary	ERROR	1
328	loop.java-pull	qa.aws-usw2-dev.canary	DEBUG	1
329	loop.java-pull	qa.aws-usw2-dev.canary	INFO	1
330	loop.java-pull	qa.aws-usw2-dev.canary	WARN	1
331	loop.java-pull	qa.aws-usw2-dev.canary	ERROR	1
332	loop.java-spring-pull	qa.aws-usw2-dev.canary	DEBUG	1
333	loop.java-spring-pull	qa.aws-usw2-dev.canary	INFO	1
334	loop.java-spring-pull	qa.aws-usw2-dev.canary	WARN	1
335	loop.java-spring-pull	qa.aws-usw2-dev.canary	ERROR	1

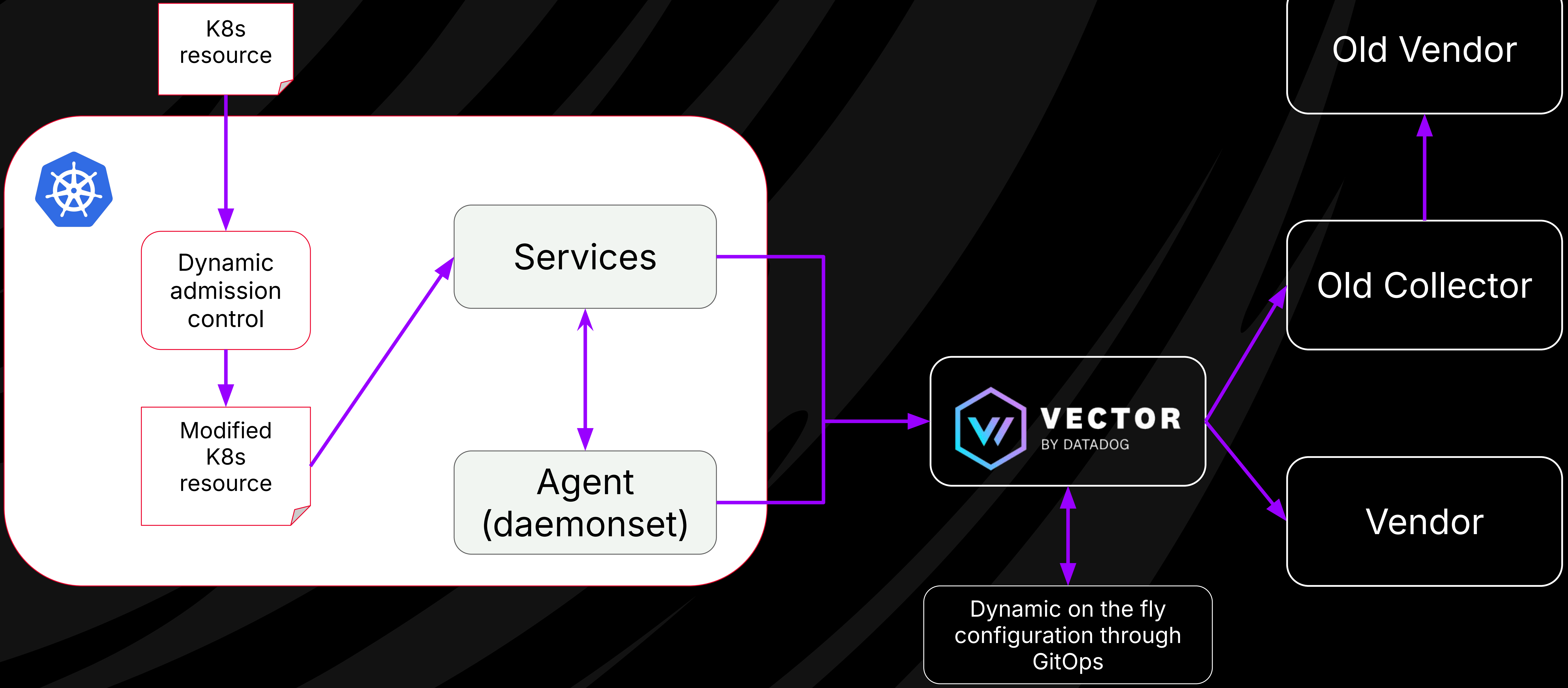
Allowlist for metrics

Introduced Allowlists for metrics

- To counter tag cardinality and metrics names with variables/labels in it.
- Metrics are mostly allowlisted. Metrics from kafka, mysql, our internal frameworks and others are allowed by default.
- Allowlists are stored dynamically using GitOps and all changes to them are applied almost instantly.



Tech overview



Additional changes

Encouraged all teams to move to monorepo with IaC files for alerts and dashboards
This is to provide the SRE teams the ability to apply cross cutting modifications to monitors and reduce burden of maintenance of such repositories for teams.

Data corrections and vendor specific attributes
We extract tags from metric names, filter high cardinality tags, normalize environment names and add vendor specific attributes.

Introduced tighter controls on API keys and Vendor functionality
API keys are now issued via special ticketing system and require a review for few specific exceptions. Vendor specific tech is vetted and tracked by us to ensure governance and compliance.

Robust CICD
We have hundred of units and integration tests. Canary applications emitting telemetry constantly being verified. Constant load tests. A/B deployments and automatic promotions.

With the tech part covered, how to approach teams and their services though?

How did we migrate the teams?

Two different approaches

Whiteglove migration

5-20 engineers embedded in specific teams with high impact services.

Helped us gather deep insight on multiple diverse technologies Riot uses.

Most of the work was carried out by us.The scope of work varied a lot from team to team, and work kept changing.

Teams got cold feet to pull the trigger as they acquired little knowledge of the vendor during the transition.

Self-service

Service teams were responsible for migrating themselves.

A lot of work on documentation was necessary with a good explanation of tools, examples, and best practices.

On-call rotation with SLAs for PR reviews and question-and-answer channels

Required switching to whiteglove along the way for some teams/services.

Teams were more confident to pull the trigger as they migrated themselves.

06 Challenges and key results

Learnings and tough spots

Sampling
Initially it was too aggressive and we underestimated how much teams relied too much on individual logs. People start thinking about workarounds to bypass it. We eased a lot of the constraints and in some cases optimized their logs, removing waste. Traces also reduced the reliances on logs.

Gatekeeping is complicated
We started with a manual process to add metrics to allowlist metrics and to fine-tune sampling. It required lengthy PR reviews that sometimes went across time zones. GitOps, documentation and Quotas have made this easier.

Vendor lock in
Teams were using a lot of different vendor specific libraries and data structures, which required to emulate it on the new platform or transition to standard tools/data structures. We dropped all vendor specific in code bases.

Learnings and tough spots

Wastage control is a constant battle
Having tools to control waste allows us to have conversations with service teams and bring data about costs. Metric cardinality and log data were and still are some of the most significant cost factors, and we can gate-keep bad practices.

Right tool for the right job
We generate a lot of data, not all of it require to be accessed in real-time manner. Some can be optimized by using logs instead of metrics, or metrics instead of logs. While others can be sent only when there is an interesting event instead of constantly being sent.

Seriously, documentation is very important
Documentation, backlinks, references and examples for different technologies and our frameworks greatly improved migration for many teams.

Some key results

We achieved 30% cost reduction in our total Vendor bills. $5M in cost avoidance. We are also using much more extensively the vendor capacities. Finished the project in 1 year. 350 TB logs/month only. 90% drop of custom metrics ingestion drop. Down to 300k synthetic runs from 3 million

Tracing is widely used. This helps us understand the web of services and the full player journeys without much domain knowledge, improving MTTD and accuracy of first escalation.

With 1 year of operation we had less than a handful of major incidents for the ingestion pipeline. At high peak it can scale up to 2000 cores with 3 TB of memory.

Easy to integrate. Support industry formats, plus internals. We support all major languages and major platforms. 0 reliance on shared frameworks.

Questions?

RIOT GAMES